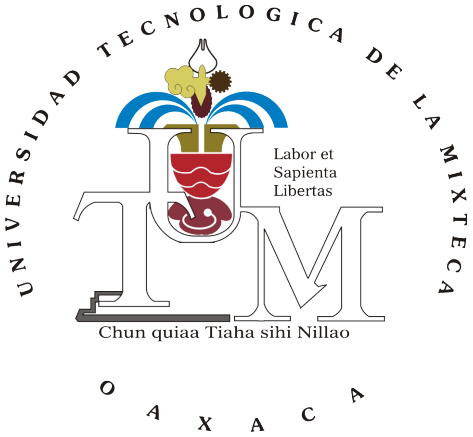

UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"DISEÑO E IMPLEMENTACIÓN DE MÓDULO DE DIRECCIÓN ASISTIDA TIPO STEER-BY-WIRE BASADA EN CAN BUS"

TESIS

PARA OBTENER EL TÍTULO DE:

INGENIERO MECÁNICO AUTOMOTRIZ

PRESENTA:

JUAN DANIEL SORIANO COLMENARES

DIRECTOR DE TESIS:

DR. ARTURO HERNÁNDEZ MÉNDEZ

HCA. CD. DE HUAJUAPAN DE LEÓN, OAXACA.
MAYO DE 2024

Para mis queridos papás, Josefina Colmenares y Jacinto Soriano Martínez.
Que nunca me falte su apoyo y cariño.

Agradecimientos

Le agradezco a Dios por darle las fuerzas necesarias a mis papás para que me apoyaran de manera incondicional y así pudiera lograr mi meta.
Muchas gracias papás, hermanas y hermano... los quiero mucho.
Abuelo Mele hice lo que un día me aconsejó... esforzarme en la escuela. Gracias abuelo Onecimo por alegrarse tanto de que yo estudiara.

Un enorme agradecimiento al Dr. Arturo Hernández Méndez, por su constante asesoramiento y por darme la oportunidad de desarrollar este tema de tesis. Además, quiero agradecerle a todos mis profesores por sus enseñanzas y a mis amigos por los momentos divertidos que ayudaban a ser menos estresantes los días de exámenes y trabajos finales.

Resumen

En este trabajo de tesis se desarrolla un prototipo de dirección steer-by-wire. El cual está integrado por dos motores DC interconectados. Su funcionamiento requiere de dos esquemas de control PID, enlazados mediante CAN Bus. Su arquitectura se divide en dos subsistemas: subsistema lado volante (SLV) y subsistema lado rueda (SLR). En el primer subsistema se desarrolla un control de corriente, para contar con retroalimentación de par en el volante. Mientras tanto, el control del SLR es de posición angular. Para comprobar el funcionamiento de estos controladores, es necesario una plataforma experimental. Su desarrollo abarca desde el diseño conceptual hasta la fabricación. El proceso de diseño asistido por computadora (CAD) se realiza mediante el software de SolidWorks. Con respecto a la parte electrónica se desarrollan dos unidades centrales, una para el SLV (subsistema lado volante) y la otra para el SLR (subsistema lado rueda). Cada unidad central está integrada, principalmente por una tarjeta de evaluación Tiva C (EK-TM4C123GXL). Para la etapa de potencia se emplea una fuente de alimentación conmutada y dos puentes H comerciales. Por otro lado, el software del sistema se desarrolla en el entorno de programación de Code Composer Studio. Los códigos son desarrollados en lenguaje C. Para evitar el uso de registros en la configuración de periféricos se utiliza un conjunto de librerías denominadas TivaWare. También se realiza una simulación del sistema en el softaware Simulink de MatLab, su finalidad es determinar la lógica de control. En este documento se presentan los resultados de la simulación, así como los resultados experimentales. Cabe señalar que estos últimos resultados, pertenecen a pruebas en condiciones normales y en algunos escenarios de falla. Finalmente, se presenta una interfaz gráfica que fue diseñada para el despliegue de información. Esta interfaz es desarrollada en App Designer de MatLab.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En el diseño del sistema de dirección automotriz se mantiene de forma predominante la utilización de elementos mecánicos e hidráulicos. Sin embargo, recientemente se ha incluido la asistencia eléctrica con el objetivo de alcanzar una mayor eficiencia en su funcionamiento [1, 2]. A pesar de los avances tecnológicos relacionados con el procesamiento de señales [3, 4], dispositivos de potencia [5, 6] y protocolos de comunicación [7-9], en general, no se tiene la confianza para desacoplar mecánicamente el sistema de dirección, ya que se tiene una mayor percepción de las desventajas que de sus beneficios.

La idea de una dirección sin acoplamiento mecánico surge del desarrollo de sistemas drive-by-wire, cuya característica principal es eliminar las conexiones mecánicas entre ciertos elementos de sistemas automotrices. Hasta la fecha se tiene el desarrollo de al menos tres sistemas que son: el acelerador por cable, el freno por cable y la dirección por cable. El primero ha sido totalmente aceptado en los vehículos a comparación de los otros dos sistemas, los cuales tienen poca presencia en el mercado. El sistema de acelerador por cable se encuentra en los modelos Mercedes-Benz Clase E y SL, y en el Toyota Estima. Mientras que, la dirección por cable (steer-by-wire) se encuentra implementada en automóviles como el Infiniti Q50 de Nissan y el Hy-wire de General Motors, entre otros [10]. Tanto el sistema de dirección como el de freno por cable deben reforzarse y mejorarse con respecto a su confiabilidad, el diagnóstico de fallas y la tolerancia a fallas.

Este trabajo se enfoca en el desarrollo e implementación de un sistema de dirección tipo steer-by-wire. La dirección steer-by-wire es un sistema automotriz completamente electrónico para el control de la dirección de un automóvil, que elimina la conexión mecánica entre el volante para conducir y las ruedas delanteras del vehículo. Este tipo de dirección debe implementar las mismas funciones de manejo que un sistema de dirección convencional, tanto en el comportamiento de manejo como en el control dinámico del vehículo. El sistema se encarga de interpretar las acciones del conductor con respecto al volante para conducir y así mantener una sincronización con las ruedas. La sensación de dirección en el volante es una simulación de los pares de torsión que la carretera aplica a los neumáticos, esto se hace con el fin de lograr una conducción realista proporcionando información valiosa sobre el estado de la carretera y la dinámica de las ruedas [11]. Los componentes

principales de una dirección steer-by-wire son: Unidades de control electrónico (ECUs), motores eléctricos (corriente directa y síncronos) y sensores, siendo estos últimos de posición, de corriente y de par, tanto en el lado volante como en el lado rueda, con la posibilidad de estar conectados a través de una red de comunicación [10]. El protocolo de comunicación más usado en la industria automotriz es el protocolo CAN Bus.

Las fallas en uno de los componentes de este sistema, errores de software o errores humanos resultan en efectos de dirección no deseados, que deben ser manejados por la arquitectura y los protocolos de seguridad [12]. La seguridad en este sistema de dirección es un aspecto muy importante.

1.1. Antecedentes

Las direcciones eléctricas se han integrado en los automóviles para generar mejores prestaciones, donde una es la evolución de otra, con el objetivo de solventar sus deficiencias y generar un sistema más completo. Al mencionar este tipo de direcciones automotrices, se tiene la dirección EPS y a la dirección steer-by-wire que comparten ciertas características que las hacen más eficientes que una con un sistema más convencional, como lo es la hidráulica. Estas primeras tienen mayores ventajas tanto en cuestiones ambientales como en aspectos de funcionalidad y arquitectura [11]. Las cuales se enlistan de la siguiente manera:

- Menor consumo de combustible.
- Posibilidad de implementar funciones de asistencia al conductor.
- Mayor estabilidad al conducir y una reducción de vibraciones en el volante.
- Menor número de componentes que reduce el peso y el espacio requerido.

Sin embargo, lograr estos beneficios no fue algo inmediato. Para ello, ocurrió una completa evolución que originó la conversión de sistemas mecánicos convencionales a sistemas electrónicos de dirección automotriz. Con tal motivo, a continuación se presenta esta transición junto con una recopilación de trabajos enfocados a la dirección tipo steer-by-wire.

1.1.1. Evolución del sistema de dirección automotriz

Eckstein, Hesse y Klein [13], describen a la dirección automotriz como el sistema encargado de traducir las acciones del conductor sobre el volante en cambios de dirección del vehículo. Al mismo

tiempo, que proporciona una realimentación adecuada del camino y la dinámica del automóvil mediante la transferencia mecánica de fuerzas y pares.

De acuerdo con la Figura 1.1, se observa que la dirección mecánica ha estado presente desde la invención del automóvil en 1886 por Carl Benz. Sin embargo, la eficiencia y la seguridad de los automóviles, así como el aumento de sus masas y sus velocidades de conducción llevaron a innovar el sistema de dirección convencional. Esta evolución trajo consigo sistemas de dirección electromecánicos compatibles con sistemas de estabilidad del vehículo y asistencia al conductor. Además de sistemas de dirección por cable donde la comunicación electrónica sustituye el vínculo mecánico entre el volante y las ruedas.

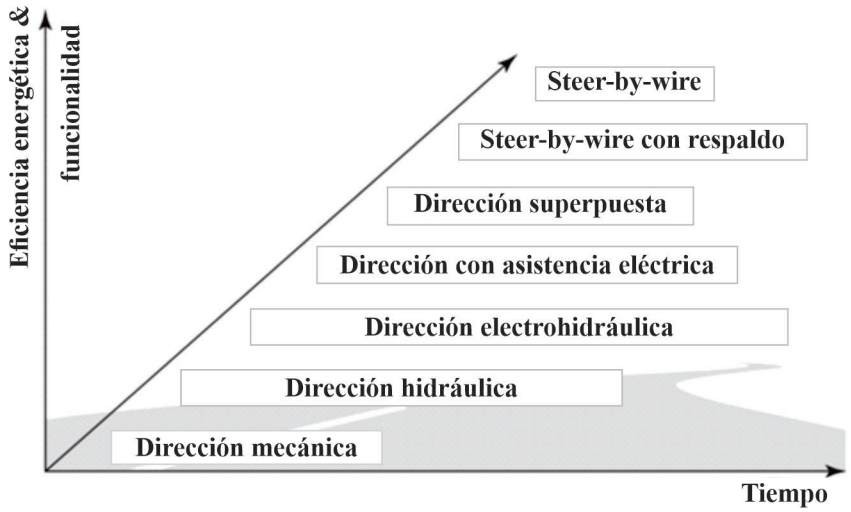

Figura 1.1: Cronología de los sistemas de dirección. Adaptada de [13].

Después de la dirección mecánica se creó la dirección hidráulica (HPS), donde se proporciona asistencia a la dirección con el propio motor del vehículo que acciona constantemente una bomba hidráulica mediante una transmisión por correa. En este sistema el par aplicado por el conductor en el volante controla una válvula que dicta la cantidad de asistencia que se aplica a la dirección [14]. La primera producción en masa comenzó en 1951 con los modelos New Yorker e Imperial de Chrysler [13].

Posteriormente, a principios de la década de 1990 surgió el desarrollo de la dirección electro-hidráulica (EHPS) equipada con motores eléctricos para accionar las bombas hidráulicas (véase Figura 1.2). A partir de este tipo de sistema de dirección se consiguió una mejor eficiencia debido a una disminución de demanda de energía [15].

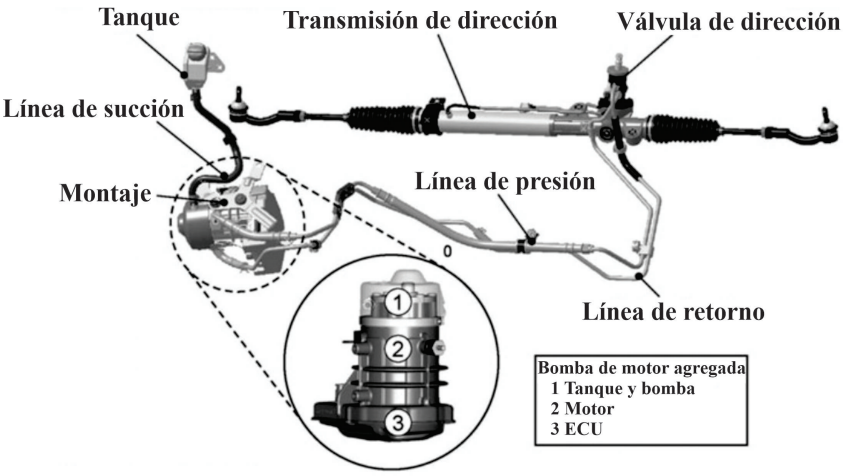

Figura 1.2: Dirección hidráulica accionada electrónicamente. Adaptada de Gessat et al. en [15].

En el año de 1988 apareció el primer vehículo que incorporaba una dirección con asistencia eléctrica (EPS). Se trataba del Suzuki "Cervo", el cual implementaba un motor eléctrico y un sensor de torque en la columna de dirección que reemplazaba a la bomba hidráulica [2]. En este sistema el par de asistencia proporcionado por el motor se transmite por toda la columna hasta llegar al mecanismo de caja de dirección, como se muestra en la Figura 1.3.

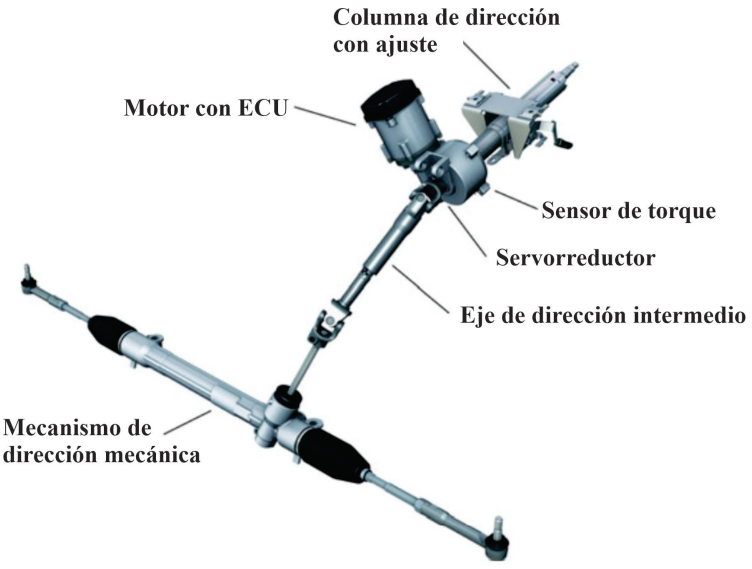

Figura 1.3: Componentes de una dirección EPS. Adaptada de Gaedke et al. en [2].

La siguiente evolución fue el sistema de dirección superpuesta que llegó a una producción en serie hasta el año 2002 por parte de BMW, así como de Toyoda Machinery Works, ZF-Lenksysteme y Lexus. Este tipo de dirección se basa en la superposición angular que añade un ángulo adicional a

la entrada del ángulo de dirección del conductor. Para ello, este sistema integra un engranaje de superposición que hace posible el control de la relación de dirección, con respecto al desplazamiento angular del volante y a la velocidad del vehículo. Ocasionando una mejor maniobrabilidad y una mayor estabilidad, a bajas y a altas velocidades, respectivamente [16].

En la Figura 1.4, se logra apreciar como una caja de engranaje de superposición se encuentra conectada directamente al motor eléctrico de la dirección mediante un mecanismo de tornillo sin fin.

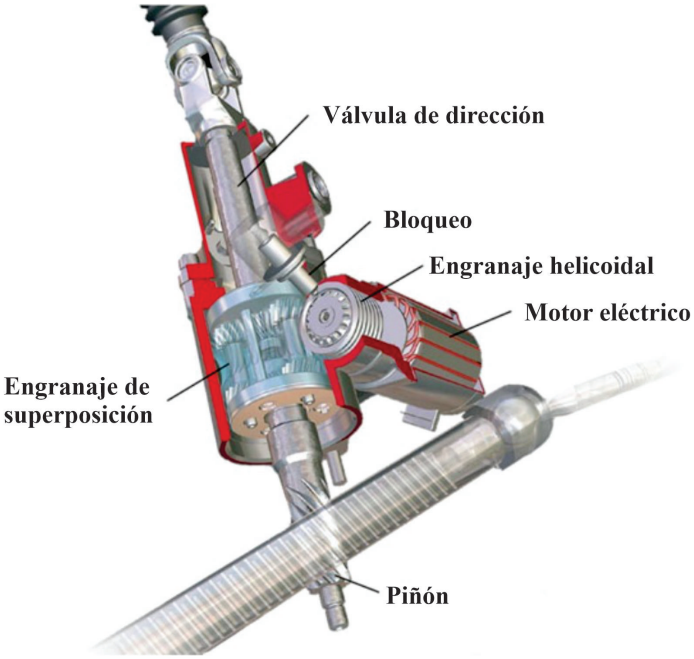

Figura 1.4: Componentes de una dirección superpuesta activa. Adaptada de [16].

Con respecto a los sistemas de dirección por cable, su aparición se ha dado en la última década del Siglo XX con el desarrollo de varios prototipos y autos conceptuales. Estos vehículos se caracterizaban por sustituir el volante de conducir con joysticks y palancas que controlaban su dirección, como ejemplo se tiene al automóvil Saab 9000 diseñado en el año de 1991 [13].

1.1.2. Arquitecturas steer-by-wire

El sistema de dirección steer-by-wire básico consiste en un sensor para detectar la entrada de posición del volante y otro para medir la posición del actuador del lado rueda, una unidad de control electrónico que interprete la información de los sensores y un actuador que modifique el ángulo de dirección de los neumáticos [13]. De este sistema parten diseños más sofisticados como el desarrollado por Huang y Pruckner [17], que se muestra en la Figura 1.5.

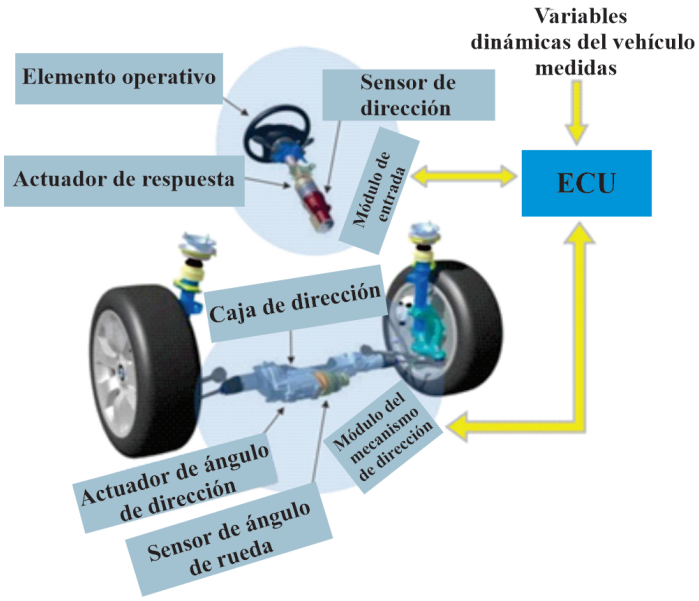

Figura 1.5: Dirección electrónica steer-by-wire. Adaptada de [17].

Se observa cómo este esquema de dirección steer-by-wire está integrado por un módulo de entrada que incluye un elemento operativo, un motor de respuesta y un sensor de posición angular, responsables de dar una sensación de par y de monitorear el ángulo del volante. Por otro lado, el módulo de mecanismo de dirección consta de un motor y un sensor de posición de las ruedas. Tanto el módulo del lado volante como el del lado rueda se encuentran controlados eléctricamente por una ECU interconectada con otros sistemas del vehículo.

Hayama et al. en [18] presentan una arquitectura muy similar a la anterior. Su sistema cuenta con retroalimentación de par y con recepción de información externa sobre la dinámica del vehículo. También integran una red de comunicación y una redundancia en el controlador, que lo hacen más completo y más seguro para el conductor (véase Figura 1.6).

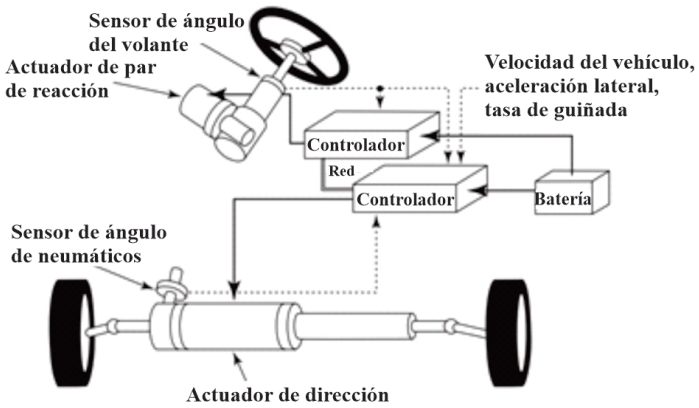

Figura 1.6: Dirección steer-by-wire con controlador dual. Adaptada de Hayama et al. [18].

1.1.3. Sistemas steer-by-wire tolerantes a fallas

Dado que los sistemas de dirección steer-by-wire se consideran críticos en cuestiones de seguridad, Eckstein et al. [13] mencionan que se necesita de cierto grado de redundancia. Su objetivo es una operación segura de la dirección en cualquier escenario de falla. Esta situación ocasiona que los investigadores propongan varias arquitecturas de sistemas steer-by-wire tolerantes a fallas.

En 2006, Wallentowitz y Reif en [19] desarrollaron una arquitectura a prueba de fallas que funciona de forma segura, incluso en caso de error. Pretendiendo que falle en silencio, es decir, que el sistema no genere efectos adversos aún estando activo. Su estrategia para garantizar la integridad de la dirección steer-by-wire se basa en redundancias locales para los subsistemas (véase Figura 1.7).

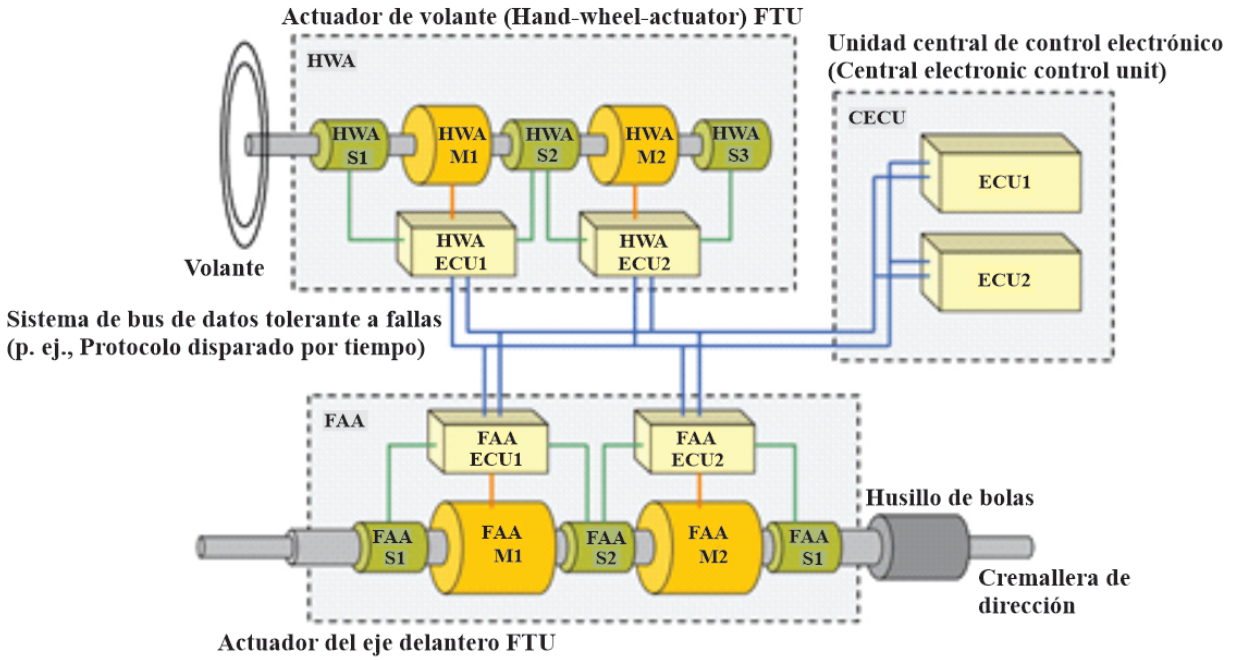

Figura 1.7: Sistema steer-by-wire tolerante a fallas. Adaptada de [19].

El subsistema lado volante y el subsistema lado rueda están diseñados con dos motores, tres sensores en un solo eje y dos unidades de control electrónico. Cada unidad procesa los datos de dos sensores diferentes a la vez que controla un actuador. Se implementan dos ECUs adicionales que representan la unidad de control electrónico central, las cuales están conectadas al resto mediante un bus de comunicación en tiempo real. El propósito de este diseño es permitir ciertos fallos electrónicos y eléctricos en cada uno de sus subsistemas sin que el conductor pierda el control.

Hayama et al. [18] también proponen una arquitectura steer-by-wire bastante compleja (Figura 1.8). Este sistema hace uso de dos actuadores de dirección redundantes y varias unidades de control.

Además, de un enlace de comunicación con otros sistemas del vehículo para implementar una asistencia de conducción.

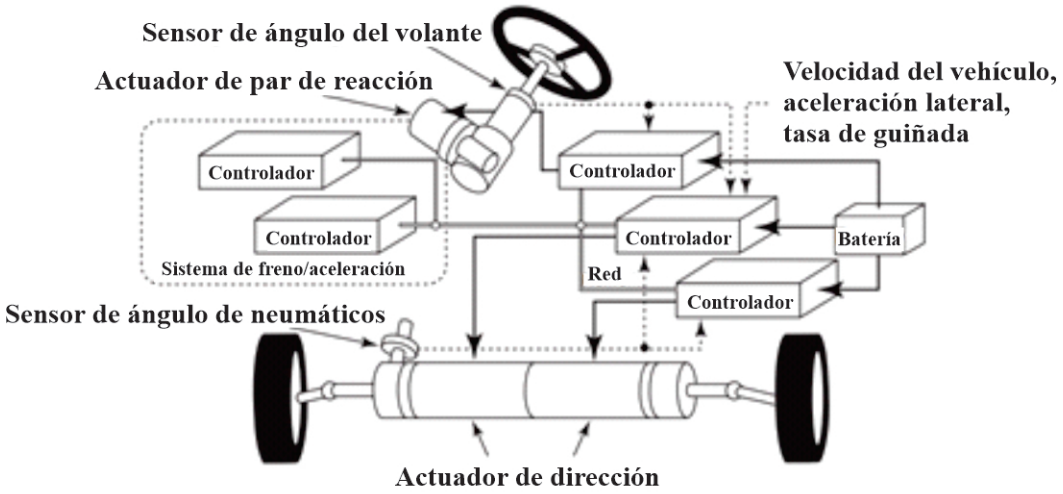

Figura 1.8: Arquitectura steer-by-wire con actuador dual. Adaptada de Hayama et al. en [18].

El objetivo de la redundancia de hardware para el actuador de dirección es evitar la falla del sistema que podría causar el uso de un único motor. Sin embargo, hay que considerar que el hecho de tener dos motores en el subsistema lado rueda lo hace más complejo, ya que esto conlleva a la necesidad de poder controlar el movimiento de ambos simultáneamente sin generarse algún conflicto y que su sistema de control logre una transferencia suave cuando se pase de tener dos motores a uno solo.

Como información adicional, en [10] se menciona que debido a la complejidad que tienen las direcciones steer-by-wire, éstas requieren del mayor nivel de tolerancia a fallas que sea posible otorgarles. También se describen dos tipos de sistemas de control tolerante a fallas: los sistemas activos y los pasivos.

El sistema de control tolerante a fallas activo se adapta de acuerdo con la información del error en tiempo real, mediante un esquema de detección y aislamiento de fallas, y un mecanismo de reconfiguración. Mientras tanto, en el sistema pasivo se consideran un conjunto de fallas potenciales conocidas y establecidas como fallas de base de diseño.

1.1.4. Sistemas steer-by-wire con mecanismo de embrague

Para mejorar la seguridad y la confiabilidad de este tipo de sistema de dirección, Onoda et al. [20] incluyen un mecanismo de embrague entre el volante y las ruedas del vehículo que funciona como respaldo mecánico. En la Figura 1.9, se observa que esta arquitectura propuesta también consta de

un motor de retroalimentación, dos motores de dirección y dos unidades de control electrónicas, que hacen que este tipo de diseño se considere como una forma efectiva de promover la confianza por parte de los conductores.

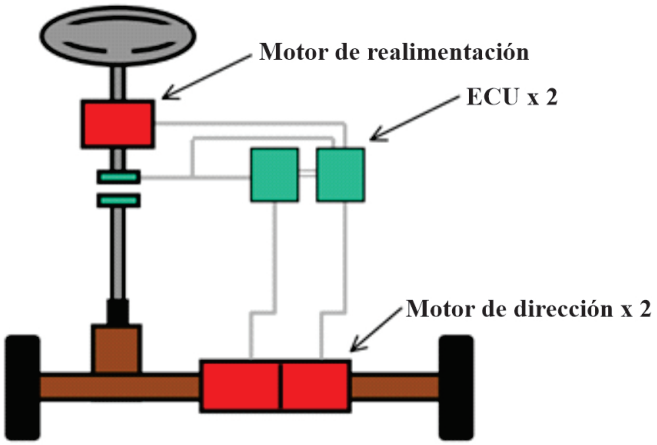

Figura 1.9: Sistema steer-by-wire con mecanismo de embrague. Adaptada de Onoda et al. [20].

Siguiendo este enfoque, Huang et al. [10] presentan un esquema de dirección steer-by-wire muy completo, con el mismo sistema de respaldo mecánico y redundancia de hardware (véase Figura 1.10).

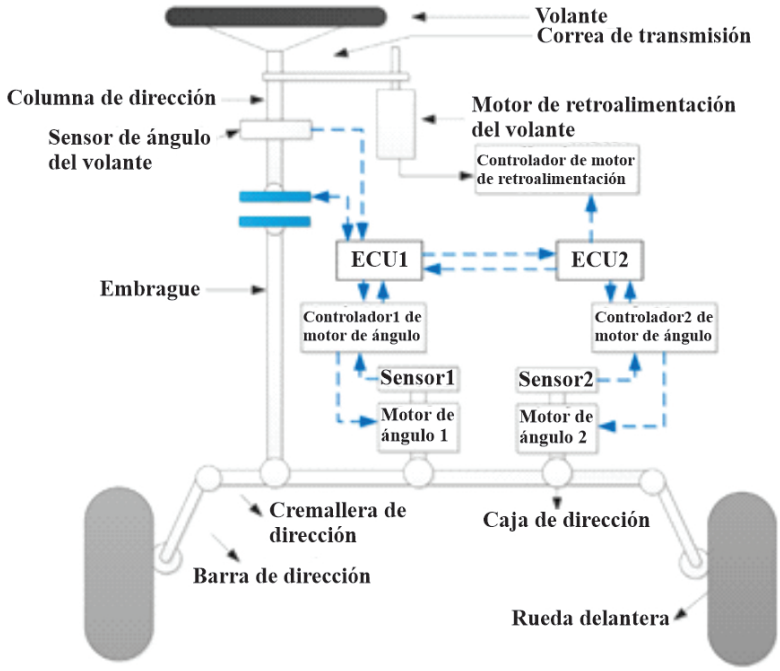

Figura 1.10: Dirección steer-by-wire con respaldo. Adaptada de [10].

1.1.5. Protocolos de comunicación en sistemas steer-by-wire

En una dirección steer-by-wire no hay enlaces mecánicos entre el subsistema lado volante y el subsistema lado rueda, los canales de comunicación son el único enlace entre los nodos de estos dos subsistemas. En la actualidad han surgido varios protocolos de comunicación compatibles con este tipo de dirección como lo son: FlexCAN, TTCAN y TTP/C. Sin embargo, existen dos protocolos que son los más utilizados por investigadores: el CAN y el FlexRay. El primero por ser la tecnología de bus dominante en vehículos y el segundo por tener velocidades de datos más altas que lidian con el retraso en el tiempo de comunicación (10 Mbps), así como de su capacidad de tolerancia a fallas [10].

El objetivo de implementar un protocolo de comunicación es un correcto intercambio de información entre las distintas unidades de control. Enseguida, se muestra la integración de estas redes de comunicación en las arquitecturas steer-by-wire.

Zheng y Anwar [21] describen una arquitectura con realimentación de par y conformada por un conjunto de dualidades en actuadores, microcontroladores, controladores de motor, canales de comunicación CAN y sensores de posición del lado rueda. Agregan un bus de arbitraje entre ambos microcontroladores para una reconfiguración automática de operación maestro-esclavo (Figura 1.11).

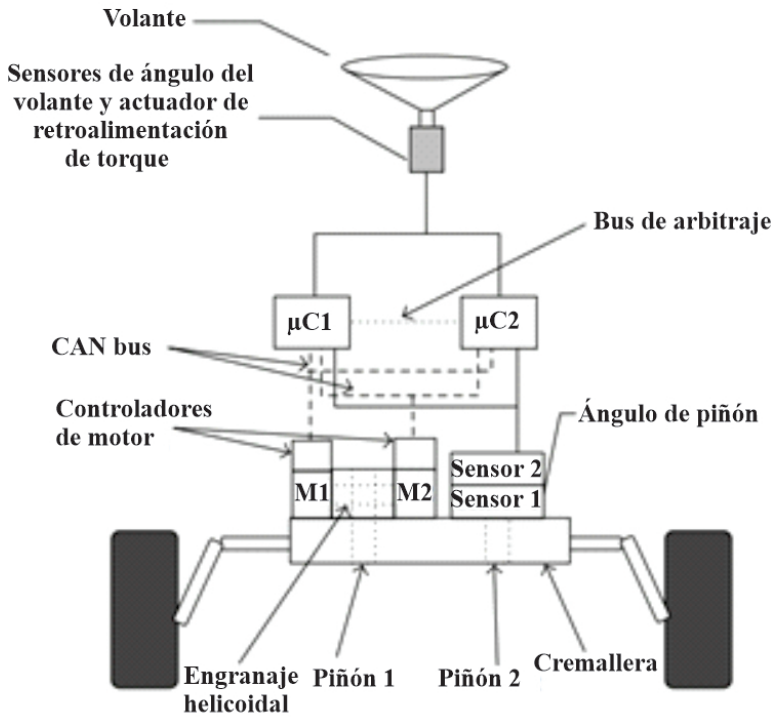

Figura 1.11: Sistema steer-by-wire basado en CAN Bus. Adaptada de [21].

El microcontrolador maestro realiza sus operaciones con la información de los sensores de posición y envía las órdenes a los controladores de motor a través de los buses CAN. Sin embargo, el esclavo también realiza sus propios cálculos para estar sintonizado y en caso de que el maestro falle, pueda asumir la tarea de control enviando sus comandos de la misma manera en los canales de comunicación.

Los investigadores indican que la información de control enviada por cualquiera de los dos microcontroladores tendrá el mismo identificador de mensaje CAN, debido a que el maestro y el esclavo nunca enviarán comandos al mismo tiempo.

Por otro lado, Zong et al. en [22] proponen un sistema steer-by-wire basado en bus FlexRay (véase Figura 1.12). En él se implementa un controlador de algoritmo que se encarga de generar las señales de dirección, de acuerdo con el ángulo de entrada del volante. Este controlador hace uso de la información de estado del vehículo y al ser la unidad central, también controla el motor de retroalimentación de par. Todas sus órdenes son enviadas a los demás controladores a través de los dos canales que proporciona el protocolo de comunicación.

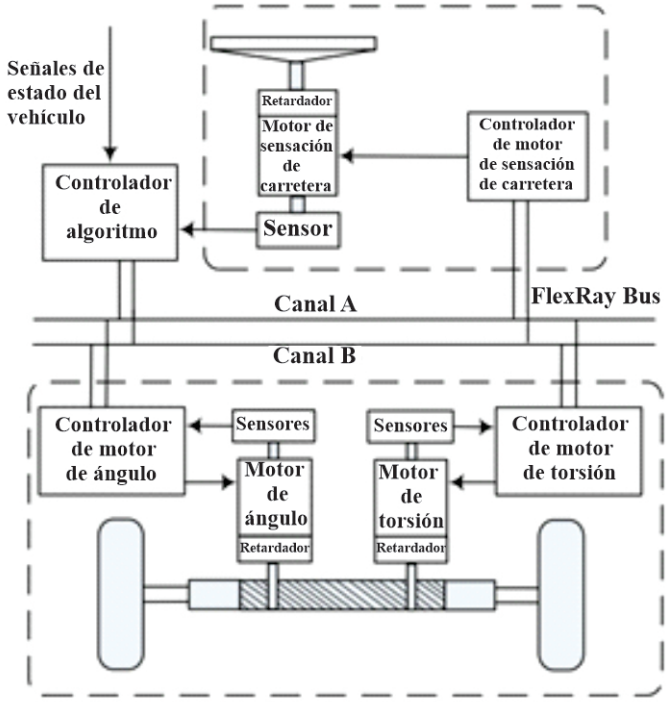

Figura 1.12: Dirección steer-by-wire basada en FlexRay. Adaptada de Zong et al. [22].

Los investigadores indican que las razones por las que seleccionaron FlexRay es debido a que es un bus de comunicación disparado por tiempo, que ofrece una alta velocidad de transmisión de datos, una arquitectura de procesamiento distribuido y una tolerancia a fallas. Sus características hacen posible el mejoramiento del desempeño y la reducción de riesgos de error en el sistema.

1.1.6. Esquemas de control implementados en sistemas steer-by-wire

Mohd Tumari et al. [23] presentan un esquema de control PID, que consideran confiable y robusto para el control direccional y la sincronización de las ruedas en una dirección steer-by-wire. Emplean dos controladores PID, uno para compensar el error producido por el ángulo del volante y el otro para controlar el desplazamiento angular de los neumáticos delanteros (Figura 1.13). Para ajustar los parámetros del controlador utilizan el método de Ziegler Nichols y como entorno de programación el software Matlab/Simulink. En su investigación, los resultados obtenidos demuestran que el controlador PID tiene un buen seguimiento de la posición angular.

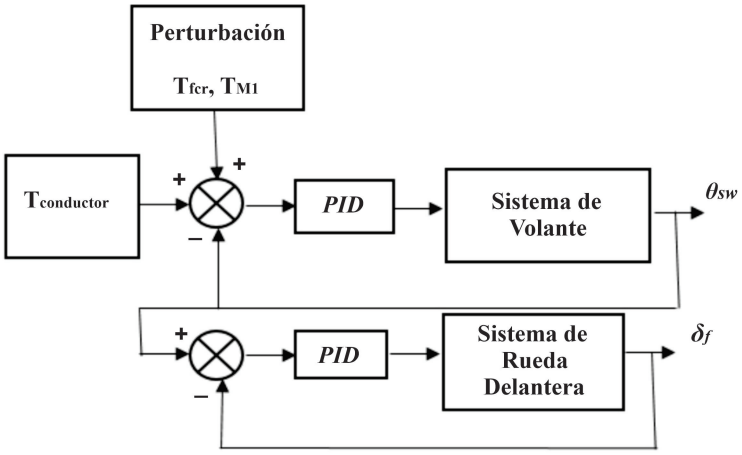

Figura 1.13: Diagrama de control PID para una dirección steer-by-wire. Adaptada de Mohd Tumari et al. [23].

En [24] se hace una comparación entre el controlador PID con optimización del hisopo de partículas (PSO) y el control de aprendizaje iterativo (ILC). Con ellos se controla el ángulo de dirección de un sistema steer-by-wire. La optimización PSO es una técnica para ajustar los parámetros de los controladores, en este caso son: la ganancia proporcional (K_p), la integral (K_i) y la derivativa (K_d) del control PID. Por otro lado, el esquema de control ILC tiene la característica de utilizar la naturaleza repetitiva del proceso para lograr un seguimiento de posición de alta precisión. Sin embargo, el estudio realizado determina que el controlador PSO-PID tiene un mejor rendimiento y es más preciso en el seguimiento de dirección.

En [25] se diseña y se simula un sistema de dirección steer-by-wire basado en un modelo que combina el control PID y el control de lógica difusa (Fuzzy-PID). El de lógica difusa se utiliza como estrategia para obtener los parámetros K_p, K_i y K_d del controlador PID. Estos parámetros son ajustados automáticamente de acuerdo con las diferentes condiciones operativas de la planta. Esta acción provoca un buen rendimiento en el seguimiento de posición y una mejor estabilidad del sistema.

El esquema de control Fuzzy-PID tiene como entradas el error y la tasa de cambio del error. Mientras tanto, sus salidas son las tres ganancias del control PID anteriormente mencionadas. A continuación, en la Figura 1.14 se tiene un diagrama general de este modelo de control combinado.

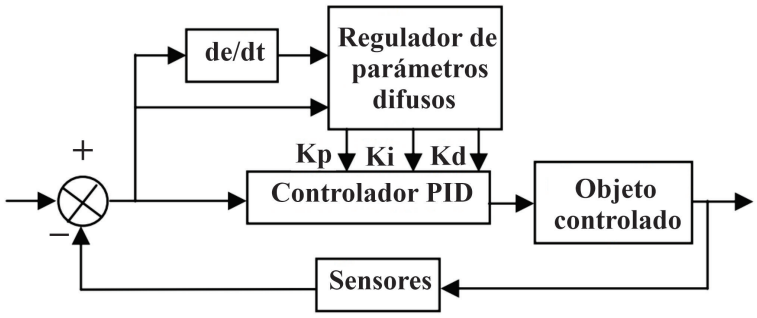

Figura 1.14: Esquema de control Fuzzy-PID. Adaptada de [25].

Chen et al. [26] realizaron el estudio de un controlador robusto para una dirección steer-by-wire basado en el algoritmo H∞ mejorado y en el tradicional de sensibilidad mixta. Para ello, hicieron un análisis completo del modelo dinámico y cinético del sistema. Al momento de simularlos, las respuestas de los controladores demostraron que el esquema de control H∞ mejorado era superior al tradicional.

Sun et al. [27] presentan el control de modo deslizante adaptativo (ASM) para una dirección electrónica steer-by-wire. Demuestran que este tipo de control tiene la capacidad de hacer frente a las incertidumbres paramétricas en el modelo de la planta. Además, mantiene la estabilidad y robustez en diferentes condiciones del camino, con errores de seguimiento de dirección significativamente pequeños.

En este controlador se utiliza el método adaptativo para obtener el coeficiente de par de autoalineación, el cual tiene efectos sobre el sistema de dirección y debe ser compensado. Mientras tanto, el modo deslizante funciona como un controlador de realimentación que hace frente a las variaciones de los parámetros del sistema.

Cetin et al. [28] mencionan que en este tipo de dirección electrónica se puede emplear una estrategia de control de referencia del modelo (MRC). Donde la fuerza y los torques externos son medidos a través de sensores, siendo estas mediciones las entradas del sistema. En esta clase de control no hay necesidad de estimar los parámetros de la planta, se hace frente a las perturbaciones y a las variaciones paramétricas, siempre y cuando se encuentren dentro de ciertos límites. En caso de que las variaciones no se encuentran dentro de los límites, se requiere de un control adaptativo de referencia del modelo (MRAC) que estime los parámetros del sistema de dirección desarrollada. Para este segundo controlador se mencionan tres métodos de estimación adaptativos que son: el de error de salida, el de error de ecuación y el método de mínimos cuadrados recursivos modificados.

Finalmente, con los resultados obtenidos de su experimentación, los investigadores demostraron que se logra un rendimiento satisfactorio con el control MRAC. Siendo más útil el método de error de salida.

1.1.7. Vehículos con dirección steer-by-wire

Eckstein et al. [13] mencionan algunos modelos de vehículos donde se implementaron sistemas de dirección tipo steer-by-wire. Uno de los primeros es el DaimlerChrysler F200 Imagination desarrollado en 1996. Se trataba de un automóvil conceptual con un conjunto de sistemas drive-by-wire y dos palancas laterales. Las palancas sustituían al volante y a los pedales: del acelerador y el freno. Con base a este modelo, en 1998 se desarrolló el Mercedes-Benz SL 500 (R129). Un prototipo de vehículo donde las tareas de control se llevaban a cabo con dos joysticks en los extremos del conductor (véase Figura 1.15).

Figura 1.15: Vehículo Mercedes-Benz R129 con palancas laterales. Imagen tomada de la red en [29].

Además, señalan que en el año 2002 General Motors presentó su modelo Hy-wire. En la Figura 1.16 se observa que este vehículo presentaba una dirección steer-by-wire con un concepto de volante bastante innovador.

Figura 1.16: Vehículo conceptual de GM con dirección steer-by-wire. Imagen tomada de la red en [30].

Su diseño constaba de un volante con centro fijo y dos palancas a los extremos. Estas palancas podían trasladarse hacia arriba y hacia abajo, creando un movimiento similar al de un volante convencional.

En la investigación realizada por Huang et al. [10] se menciona como la empresa Nissan desarrolló en 2013, el modelo Infiniti Q50 equipado con un sistema steer-by-wire de respaldo triple y un sistema de varillaje de dirección. Este vehículo se muestra en la Figura 1.17.

Figura 1.17: Modelo Infiniti Q50 de Nissan. Imagen tomada de [31].

Su sistema de respaldo consta de tres unidades de control electrónicas individuales que tienen como objetivo el comparar sus resultados para detectar errores y en caso de que las tres unidades fallen, el sistema activa un embrague para contar con una conexión de dirección mecánica y así el conductor pueda dirigir el automóvil.

Hace poco tiempo, en abril de 2022, se presentó el Lexus RZ 450e completamente eléctrico con tecnología steer-by-wire. El clásico volante redondo es sustituido por un diseño rectangular muy sofisticado conocido como “One Motion Grip” (Figura 1.18).

Figura 1.18: Dirección steer-by-wire “One Motion Grip” del RZ 450 e. Imagen tomada de página oficial de Lexus [32].

Este volante no genera incomodidad al momento de realizar las maniobras de conducción, debido a la capacidad del sistema para cambiar la relación de dirección. Cuenta con un giro máximo de 150° tanto a la izquierda como a la derecha, evitando que el conductor cruce las manos al momento de girarlo. Además, el fabricante menciona que este automóvil utiliza redundancia en la fuente de alimentación de su sistema de dirección, en caso de emergencia [32].

Por otro lado, en el mismo año la empresa automotriz Toyota lanzó su vehículo eléctrico bZ4X con dirección electrónica steer-by-wire y una arquitectura de volante similar a la del Lexus RZ 450e (véase Figura 1.19) [33]. Dando a entender que la tecnología "One Motion Grip", podría convertirse en una verdadera tendencia para las demás compañías de automóviles.

Figura 1.19: Volante rectangular del Toyota bZ4X. Imagen tomada de página oficial de Toyota [33].

1.2. Planteamiento del problema

El acople mecánico, entre el volante y las ruedas, presenta algunos inconvenientes. En primer lugar, siempre hay limitaciones en cuanto a peso y espacio, considerando que por lo general el motor se ubica en la parte frontal del vehículo. Esto complica el diseño, la disposición y distribución de los diferentes elementos del acoplamiento mecánico de la dirección e incluso de algunos componentes del motor [17]. Además, el acoplamiento mecánico de la dirección es un elemento crítico en el tema de seguridad ante posibles colisiones frontales. En este tipo de colisiones, la columna de dirección puede entrar peligrosamente a la cabina con posibles consecuencias graves para el conductor [13]. Para resolver este problema, se han implementado complejos y costosos sistemas pasivos para colapsar la dirección en caso de impacto [34].

Sin lugar a duda, el sistema de dirección con asistencia eléctrica (EPS) presenta importantes ventajas con respecto a la asistencia hidráulica, tales como: menor cantidad de partes móviles, menor peso, menor ruido, mayor eficiencia, sistemas de aviso y registro de fallas, capacidad de

variar el nivel de la asistencia, entre otros. Sin embargo, se mantiene el acoplamiento mecánico entre el volante y las ruedas con las desventajas ya mencionadas [2].

Debido al avance tecnológico en dispositivos de procesamiento de datos y de manejo de potencia, se considera factible el enfoque sin acoplamiento mecánico, denominado también steer-by-wire. Además de las ventajas del sistema de dirección con asistencia eléctrica, se agregan las relacionadas con la ausencia de la columna de dirección, facilidad para integrar sistemas de asistencia de manejo (control de estabilidad, estacionamiento automático, evasión de obstáculos, manejo autónomo, entre otros) y de diagnóstico. Sin embargo, los sistemas steer-by-wire tienen algunos inconvenientes, tal como la sensación de realimentación que debe tener el conductor al volante sobre el tipo de terreno por el cual se circula y la sensación ligada a la confianza y seguridad del sistema. Lo anterior, significa que se debe confiar completamente en sistemas electrónicos y de potencia, en un sistema de vital importancia [17]. Dicha confianza en la implementación de sistemas by-wire, se está alcanzando con los avances tecnológicos y un cambio en la forma de pensar de los clientes. Actualmente ya existen algunos modelos de automóvil en el mercado que ya cuentan con este tipo de tecnología: Infiniti Q50 y Q60 de Nissan, Lexus RZ 450e y Toyota bZ4X.

Tal como ya se ha mencionado, el sistema steer-by-wire requiere estrictas condiciones de rendimiento y seguridad, como una alta capacidad de tolerancia a fallas de dispositivos electrónicos e incluso del esquema de control [10]. Por lo tanto, en el diseño de una dirección steer-by-wire se debe emplear un esquema de control robusto apoyado por un sistema de comunicación de uso automotriz, como lo es el protocolo de comunicación CAN. Con el objetivo de crear una arquitectura con seguridad crítica, que permita el correcto intercambio de información entre los elementos del sistema de dirección [12]. Los sensores enviarán información de posición y corriente a las unidades de control, y éstas lo interpretarán para dar órdenes al actuador correspondiente, como se observa a continuación en la Figura 1.20.

Es lógico que los sistemas steer-by-wire tengan estrictos requerimientos de seguridad (modo a prueba de fallas, redundancia, alertas y diagnóstico), capacidad de control (error de seguimiento de asistencia, realimentación de sensación de manejo, compensación de vibraciones) [10] y una adecuada integración con el resto de los sistemas automotrices (intercambio de información vía protocolo seguro de comunicación) [18].

Bajo este contexto, se propone el diseño de una dirección electrónica tipo steer-by-wire basada en un control clásico PID. El sistema propuesto debe contar con un adecuado desempeño de asistencia de conducción y con una capacidad de respuesta aceptable en el seguimiento del ángulo de ataque de las ruedas. Además, de contar con realimentación de sensación de manejo y brindar seguridad ante ciertos escenarios de falla que lo harán confiable para el conductor.

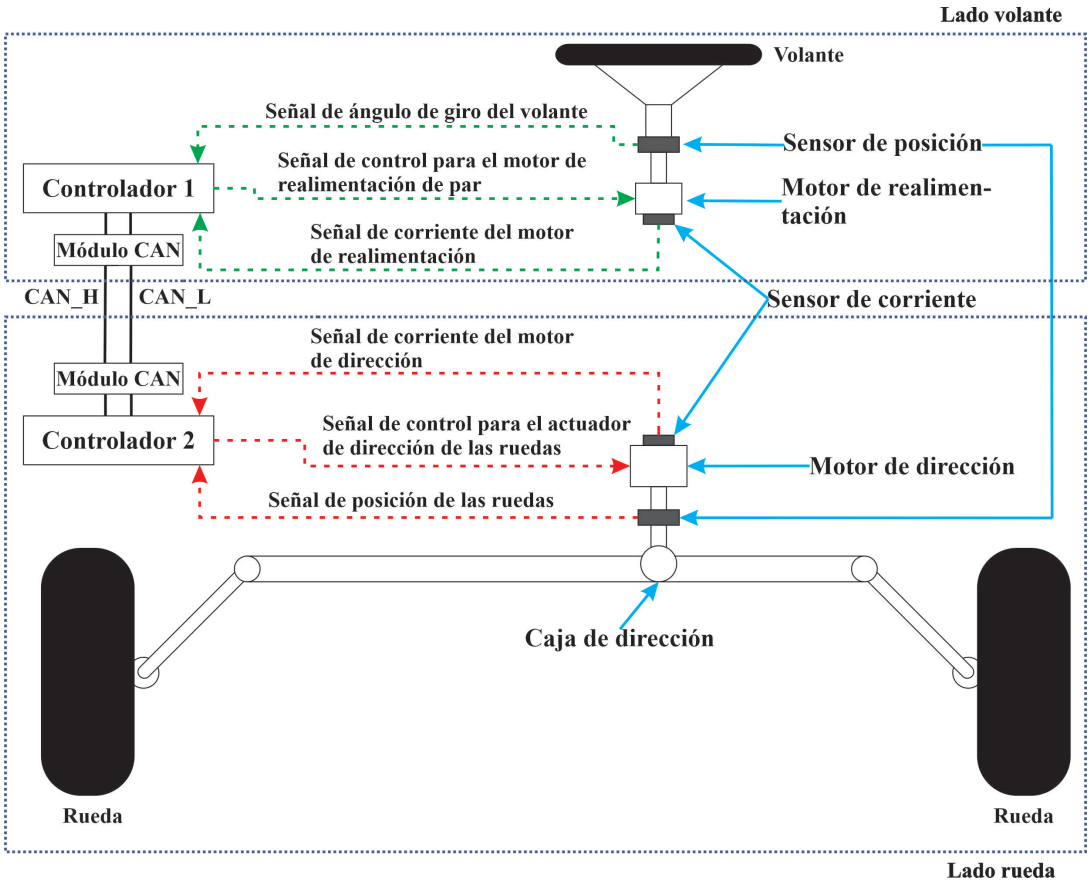

Figura 1.20: Esquema del sistema de dirección propuesto.

Una dirección steer-by-wire con un alto grado de tolerancia a fallas, requiere de un conjunto de redundancias en unidades de control, sensores, canales de comunicación y hasta en actuadores. Sin embargo, contar con más componentes en este sistema ocasiona un aumento significativo en el costo del trabajo propuesto. Por lo tanto, se tiene como límite emplear solo la cantidad necesaria de componentes para que la dirección funcione de manera adecuada y con las características establecidas.

Con respecto al banco de pruebas, se tiene ciertas limitaciones en el módulo del lado rueda. El motor DC no está conectado a una caja de dirección debido a que esto lo hace más complejo y costoso. Los elementos de este subsistema se colocan en una plataforma que forma parte del banco experimental.

1.3. Justificación

La característica principal de la dirección steer-by-wire, que la distingue de los sistemas de dirección convencionales y de la dirección EPS, es el hecho de que este sistema no cuenta con una

columna de dirección que conecte al volante para conducir con las ruedas del automóvil. La ausencia de este enlace mecánico trae consigo una serie de ventajas, tanto para la parte de diseño del vehículo como para algunos aspectos de seguridad.

Al implementar un sistema steer-by-wire, los fabricantes de automóviles tienen mayor libertad para el diseño de la parte delantera. La ausencia de la columna de dirección facilita la ubicación del volante y deja disponible un valioso espacio en el compartimiento del motor. Además, brinda una ventaja en situaciones de colisión frontal debido a que se tiene menor probabilidad de que el volante invada el espacio de sobrevivencia del conductor [11]. Por otro lado, la principal desventaja en este tipo de dirección causada por la falta de una conexión mecánica es que se debe confiar en un sistema enteramente electrónico, donde una falla en el sistema de control o en algún componente puede generar graves accidentes. Esta es la razón de los problemas de aceptación con estos sistemas por parte de las personas, que desconfían de la seguridad que las direcciones steer-by-wire pueden brindar.

Bajo este contexto, y siguiendo la tendencia de innovación tecnológica, son importantes los estudios relacionados con tecnologías by-wire. Por otro lado, en el ámbito local es necesario contar con un prototipo experimental con la capacidad de probar diferentes estrategias de control y seguridad en sistemas steer-by-wire en la UTM. Con base en lo antes mencionado, a continuación, se describen los aspectos que justifican este trabajo.

1.3.1. Pertinencia

- Es un trabajo que contribuye a la tendencia actual de transformar sistemas mecánicos en electrónicos en la industria automotriz.
- Investigaciones y desarrollos de este tipo de sistemas eficientes en cuanto al consumo energético, son importantes dada la tendencia hacia tecnologías sustentables como los vehículos eléctricos.
- Las características de desempeño que brinda este enfoque denominado by-wire, lo hacen compatible con el desarrollo de vehículos autónomos modernos.

1.3.2. Relevancia

- La necesidad de la implementación y comprobación de desempeño de esquemas de control clásico en sistemas que requieren estabilidad y seguridad.
- El desarrollo de este trabajo de tesis deriva en una serie de contribuciones que se describen a

continuación.

1.3.3. Contribuciones

Durante el desarrollo del presente trabajo, se comprueban las hipótesis planteadas, así como el desempeño del sistema de forma experimental. En el proceso se realizaron las siguientes contribuciones:

- Análisis del modelo matemático del esquema interconectado y del controlador descentralizado propuesto, en un ambiente de simulación.
- Se implementan algunos mecanismos de seguridad propuestos y se comprueba su desempeño ante determinados escenarios de falla.
- Se diseña y fabrica una plataforma experimental para comprobar el desempeño del esquema de control propuesto, que además servirá para el desarrollo de trabajos futuros.
- También se contribuye al estudio e implementación de redes de comunicación basadas en CAN integradas en sistemas de control descentralizado, en aplicaciones automotrices.

1.4. Hipótesis

En función de lo establecido anteriormente, en este trabajo de tesis, se desarrolla un sistema de control de asistencia de par sin columna de dirección, lo suficientemente robusto, con mínimo error de seguimiento de posición y tolerante a determinados escenarios de falla. Para lo cual, se tienen las siguientes hipótesis:

- El control de asistencia de par y de seguimiento de posición, se llevará a cabo de forma robusta y estable con un esquema de control clásico PID.
- El esquema steer-by-wire propuesto, es un sistema interconectado de dos motores DC, un motor lado volante y otro lado rueda (véase Figura 1.20). Por lo tanto, se diseñará un esquema de control clásico descentralizado para cada uno, con un enlace de comunicación basado en CAN Bus. Se espera que el tiempo de respuesta y la integridad de la información sean los adecuados para llevar a cabo la tarea de control.
- La validación continua de corriente consumida por el motor del subsistema lado rueda, el monitoreo de su temperatura y la implementación de un modo seguro permitirán un estado

de alerta de las condiciones del sistema ante determinados escenarios de falla y un funcionamiento adecuado al menos para el control de seguimiento de posición.

- Se implementará una función de retorno a cero artificial, que hará que el volante regrese a su posición central, como si se tratara de un vehículo con acoplamiento mecánico.

1.5. Objetivos

1.5.1. Objetivo general

Diseño y desarrollo de una dirección steer-by-wire con sistema de control robusto y tolerante a ciertos escenarios de falla, basada en protocolo de comunicación CAN.

1.5.2. Objetivos específicos

I. Diseño conceptual del sistema de dirección steer-by-wire propuesto.

II. Simulación y validación del esquema de control propuesto.

III. Diseño del banco de pruebas para la dirección steer-by-wire, usando el software de SolidWorks.

IV. Fabricación del banco de pruebas y montaje de los componentes del sistema steer-by-wire.

V. Diseño y armado de las placas electrónicas para el subsistema lado volante y el subsistema lado rueda.

VI. Control del seguimiento de posición y de realimentación de par.

VII. Conexión entre el lado rueda y el lado volante del sistema de dirección, mediante un enlace de comunicación CAN.

VIII. Desarrollo e implementación de escenarios de falla.

IX. Obtención y análisis de resultados del sistema de dirección desarrollado.

1.6. Metas

I. Diseño de un banco de pruebas en SolidWorks. Para finalmente, fabricarlo e integrarle todos los elementos del sistema de dirección steer-by-wire propuesto.

II. Desarollo de un sistema de control clásico de seguimiento de posición y de realimentación de par, usando la información obtenida de los sensores de posición y de corriente implementados en el sistema.

III. Control del motor de dirección y del motor de realimentación de par de manera descentralizada, mediante un enlace de comunicación tipo CAN Bus.

IV. Visualización de las señales importantes para la evaluación del desempeño de la dirección, en ambos esquemas de control.

V. Prueba del prototipo de dirección ante los escenarios de falla propuestos. Las alertas generadas se comparten entre los subsistemas junto con el resto de información, a través del canal de comunicación CAN.

1.7. Metodología

La metodología para la elaboración de este proyecto de tesis se basa del proceso de desarrollo de hardware y software para sistemas electrónicos embebidos y de seguridad crítica, presentado por Perez et al. [35]. En la figura 1.21 se muestra el esquema de la metodología, su ciclo de desarollo consta de 7 fases descritas a continuación:

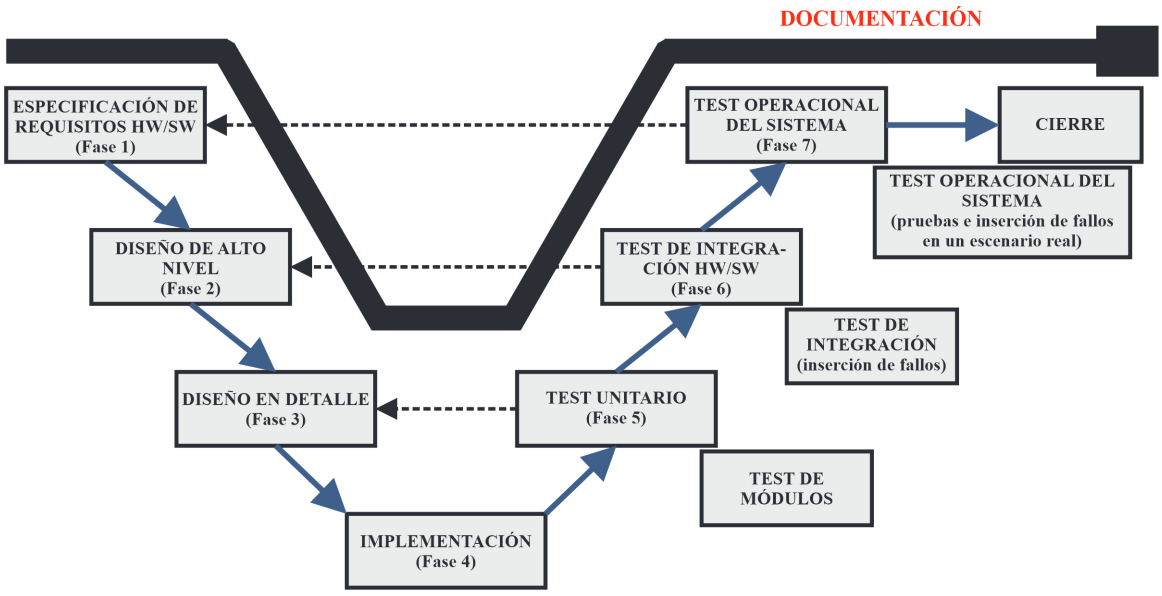

Figura 1.21: Ciclo de desarrollo clásico en V para sistemas electrónicos embebidos y de seguridad crítica. Adaptada de Perez et al. [35].

Definición de especificaciones. Consiste en establecer los requisitos y características del

sistema, tanto a nivel de software como de hardware. Para establecer las especificaciones del sistema embebido se toma en cuenta el estándar IEEE-830.

En esta etapa se plantean las funciones del sistema y se realiza su diseño conceptual. Se selecciona el dispositivo lógico programable y el entorno de programación. Para las placas de circuito impreso, la plataforma experimental y la interfaz gráfica, se determina el software de diseño.

Diseño global. Tiene como principal objetivo la obtención de un diseño que brinde la visión general del sistema.

Diseño en detalle. En esta fase se hace una descripción detallada de cada bloque o subsistema que integra el trabajo propuesto.

En este punto se realiza el diseño virtual del prototipo y de las placas electrónicas. En el caso del software, se crean diagramas de flujo con la lógica de control. Además, se realizan simulaciones del sistema.

Implementación. Consiste en la materialización del hardware y la programación del software planteados en el diseño en detalle.

En el caso del hardware se fabrican las placas de circuito impreso y se construye la plataforma experimental del sistema. Mientras tanto, para el software se realizan los códigos para los sensores, los algoritmos de control, el bus de comunicación, entre otros. También se desarrolla la interfaz gráfica para despliegue de información.

Test unitario. Esta fase tiene como tarea verificar de manera aislada, que el módulo de hardware y el módulo de software funcionen correctamente.

Por lo tanto, en este punto se deben revisar los códigos. Verificando las tareas de control, los tipos de variables, los valores devueltos por las funciones y la configuración de los periféricos. En los circuitos electrónicos se deben corroborar las conexiones y para la plataforma experimental su integridad con pruebas de uso.

Integración. En esta etapa se integra el módulo de software y hardware, de tal manera que funcionan a la par e interactúan entre sí. Una vez obtenido el sistema completo se comprueba su funcionamiento, ante situaciones normales y en determinadas fallas. Finalmente, se registran los resultados en un documento.

Test operacional del sistema. Consiste en la realización de una serie de pruebas en un escenario real y como en los casos anteriores, tener una documentación que lo evidencie y muestre los datos obtenidos.

El sistema desarrollado es un prototipo para experimentación, no se integra en un automóvil. Por lo tanto, esta última fase de la metodología no se realiza.

1.8. Organización de la tesis

El **Capítulo 1** comienza con el contexto general del proyecto de tesis. Seguido de un conjunto de antecedentes que dan muestra de la evolución de la dirección automotriz y de algunos trabajos de investigación realizados con respecto al tema abordado. Además, contiene el planteamiento del problema, la justificación, las hipótesis, los objetivos y la metodología elegida. En el **Capítulo 2** se proporciona los aspectos teóricos relacionados con el protocolo de comunicación CAN, el control PID y el control de posición angular de motores DC. Posteriormente, en el **Capítulo 3** se describe el diseño preliminar del sistema steer-by-wire, donde se presentan sus especificaciones y características generales. Después de tener un diseño conceptual del sistema, en el **Capítulo 4** se describe el diseño virtual y la manufactura del banco de pruebas.

Por otro lado, en el **Capítulo 5** se presentan los módulos de electrónica que conforman el prototipo (potencia, instrumentación, lógica programable y datos). Además, se presentan esquemas de la plataforma experimental de hardware y software. En el **Capítulo 6** se describe el desarrollo de los bloques lógicos de la dirección steer-by-wire. Se presenta la simulación del sistema y la lógica de control de los programas desarrollados. A continuación, en el **Capítulo 7** se presentan los resultados de la etapa experimental, donde también se proporcionan las conclusiones y la propuesta de los trabajos futuros. Finalmente, el apéndice incluido en este documento es el siguiente, **Apéndice A:** Programación del sistema steer-by-wire.

Capítulo 2

Marco teórico

En este capítulo se describen algunos aspectos teóricos relacionados con el desarrollo del prototipo de dirección steer-by-wire. La información presentada se enfoca en el protocolo de comunicación CAN y en el control de motores DC (control de corriente y de posición angular). Además, se presentan ciertos fundamentos teóricos relacionados con el esquema de control PID. A continuación, se desarrollan los temas anteriormente mencionados.

2.1. Protocolo de comunicación CAN

La integración de los sistemas electrónicos en la arquitectura de los vehículos causó que existiera una gran cantidad de sensores, actuadores y unidades de control electrónicas. Todos estos elementos ayudaron a que los automóviles sean más eficientes y cumplan con las regulaciones ambientales. Sin embargo, el aumento de cables significó un problema en la gestión de datos y en el peso del vehículo. Como consecuencia, se establecieron un conjunto de reglas para la transmisión de datos en los sistemas automotrices llamados protocolos de comunicación. Entre ellos destaca el protocolo de comunicación CAN (Controller Area Network), que fue desarrollado en el año de 1983 por Robert Bosch GmbH en Alemania [7].

Las características de este protocolo lo han colocado como el más usado en la industria automotriz, especialmente para la gestión del tren de potencia [36]. Es catalogado como una red de clase C debido a que alcanza velocidades de transmisión de datos de hasta 1 Mbps [7]. Además, de ser un protocolo de comunicación estandarizado en cuestiones de velocidades y capa física, por la Sociedad de Ingenieros Automotrices (SAE) y la Organización Internacional para la Estandarización (ISO) [36].

Existen dos tipos de comunicación: la comunicación síncrona y la asíncrona. En la primera de ellas el receptor lee el bus cada determinado tiempo, porque sabe cuándo está disponible cada dato. Mientras tanto, la comunicación asíncrona es todo lo contrario debido a que solo se usa el bus cuando es necesario. A su vez, Shrinath y Emadi [7] mencionan que la comunicación síncrona es una comunicación disparada por tiempo y la asíncrona es disparada por eventos. Disparada por

tiempo significa que existe una señal de reloj que hace que cada usuario tenga un rango de tiempo del total del ancho de banda y la transmisión de datos se dé en intervalos específicos. Por otro lado, que sea disparado por eventos es que el canal de comunicación se encuentra ocupado solo cuando hay datos que transmitir. El protocolo de comunicación CAN es asíncrono y disparado por eventos.

Siguiendo con su descripción, se tiene que es un protocolo orientado al mensaje donde los datos que se transmiten en la red son enviados a todos los nodos conectados. Estos tienen la obligación de mirar dentro del paquete de datos y determinar si el mensaje era para ellos o simplemente deben ignorarlo, esto lo hacen mediante un identificador de mensaje [7]. Además, Hernández et al. [36] describen que el protocolo CAN cuenta con una estructura multi-maestro, donde cada nodo conectado al bus tiene la capacidad de mandar o recibir información de manera bidireccional.

Con respecto a la prioridad de mensajes, se establece que solo un nodo puede transmitir mientras que todos los demás escuchan. Cuando ocurre el caso de que varios quieren enviar datos al mismo tiempo se recurre al uso de un identificador único que determina la prioridad de cada mensaje, esto con el objetivo de que el nodo con el mensaje más importante siga transmitiendo y los demás dejen de hacerlo. La prioridad está basada en el valor binario del identificador; es decir, entre más bajo sea el valor binario mayor es la prioridad y viceversa [7, 37].

Las capas del modelo OSI del protocolo CAN importantes para el desarollo de este trabajo son: la capa física y la capa de enlace de datos. Dichas capas se describen a continuación.

2.1.1. Capa física

La topología que se usa para el protocolo CAN de acuerdo con el estándar ISO 11898-2 es de tipo bus o lineal [8]. Se caracteriza por contar con dos resistencias a los extremos de 120 Ohms y un par de cables trenzados en los cuales se lleva a cabo la transmisión de señales. Estas señales se denominan CAN-H (CAN-HIGH) y CAN-L (CAN-LOW), como se muestra a continuación en la Figura 2.1.

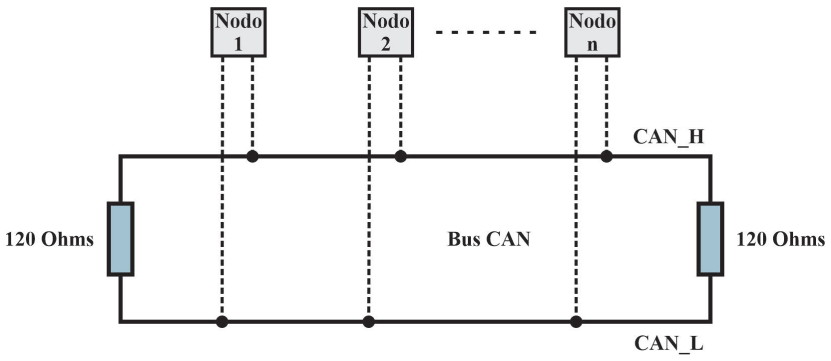

Figura 2.1: Topología del bus CAN de acuerdo al estándar ISO 11898-2.

Andrango en [8], resume las características eléctricas principales de una red de comunicación CAN establecidas en el estándar ISO 11898-2, algunas de ellas son:

- Velocidad máxima de transmisión de 1 Mbps.
- Bus de dos cables.
- Impedancia de línea de 120 Ohms.
- Número total de nodos limitado por la carga eléctrica del bus, pudiendo conectarse hasta una cantidad máxima de 110 nodos.
- Longitud máxima del bus de 40 m a 1 Mbps.
- Longitud máxima de stub de 30 cm a 1 Mbps.

Además, este mismo estándar establece los niveles nominales del bus en forma de tensión diferencial (véase Figura 2.2).

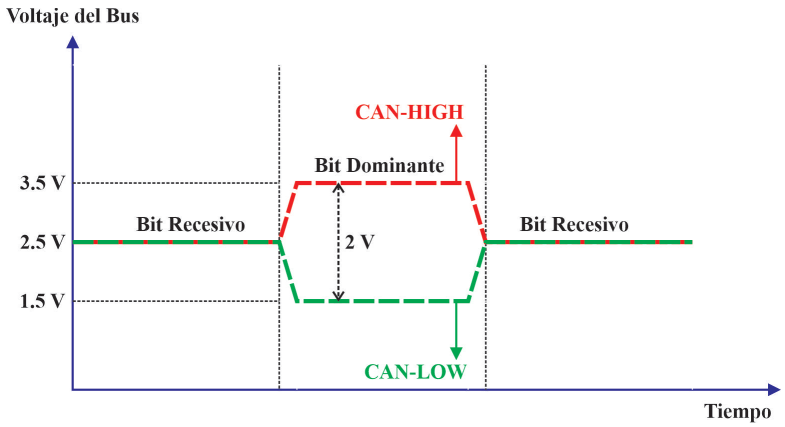

Figura 2.2: Niveles nominales del bus CAN. Adaptada de [37].

El bus tiene dos estados: estado dominante y estado recesivo. En el primero de ellos existe una diferencia de tensión entre CAN-H y CAN-L de 2 V, mientras que en el estado recesivo los dos cables que conforman el bus se encuentran al mismo nivel de tensión, es decir, existe una diferencia de tensión de 0 V [37].

2.1.2. Capa de enlace de datos

En las redes de comunicación, los nodos compiten entre sí por el acceso al bus. Esta acción puede generar problemas serios en cuestiones de colisiones de datos. Existen algunos métodos de arbitraje

del bus para evitar esta situación, que son: CSMA/CD (Acceso Múltiple del Sensor de Portadora/ Detección de Colisión) y CSMA/CA (Acceso Múltiple del Sensor de Portadora/ Prevención de Colisiones). El tipo de acceso al medio que utiliza el protocolo CAN es de prevención de colisiones. En este tipo de acceso, los nodos primero revisan que el bus se encuentre libre y en caso de que dos o más nodos transmitan al mismo tiempo prevalece el de mayor prioridad. Así, dicho mensaje no se pierde y el resto de los nodos intentan transmitir nuevamente [7, 38].

En [8, 38] se indican cuatro tipos de tramas que componen el protocolo de comunicación CAN:

- *Trama de datos.* Su principal función es poner información en el canal de comunicación y así los nodos receptores puedan recibirla.
- *Trama remota.* Es utilizada por los nodos receptores para solicitarle a un nodo transmisor que envíe el mensaje con el identificador dado. El nodo al que pertenezca ese identificador tiene la obligación de transmitir la información en una trama de datos.
- *Trama de error.* Se genera cuando un nodo detecta algún error definido.
- *Trama de sobrecarga.* Se genera como consecuencia de que algún nodo requiere más tiempo para procesar los mensajes recibidos.

El prototipo de dirección steer-by-wire desarrollado consta de dos subsistemas interconectados mediante este protocolo de comunicación. El subsistema lado volante requiere la información del consumo de corriente en el subsistema lado rueda, al mismo tiempo que proporciona la posición angular de referencia a este otro. Por lo tanto, el tipo de trama que se utiliza es de datos. Lokman et al. en [39], describen cada una de las partes que forman esta trama CAN. A continuación se presentan algunas de ellas:

- *Campo de datos:* Contiene de 0 a 8 bytes y en él se encuentran todos los datos transferidos dentro de una trama CAN.
- *Campo CRC:* Se llama código de redundancia cíclica, consiste en un conjunto de 15 bits y un bit delimitador. Sus funciones son: que el receptor pueda verificar y determinar la validez de la trama recibida.
- *Campo de acuse de confirmación (ACK):* Consta de 2 bits que tienen como función informar al nodo transmisor que el receptor ha recibido correctamente una trama, si no es así el transmisor debe enviar nuevamente los paquetes de datos.

El protocolo de comunicación CAN tiene la capacidad de manejar diferentes tipos de errores, motivo suficiente para integrarlo en el desarrollo de este trabajo. Shrinath y Emadi [7], mencionan que sus mecanismos de detección de errores son:

- Chequeo de bit.
- Chequeo de regla de relleno.
- Verificación de redundancia cíclica (CRC).
- Chequeo de trama.
- Verificación de errores de reconocimiento (ACK).

2.2. Control de motores de corriente directa

El prototipo de dirección steer-by-wire está integrado por dos motores DC interconectados. En el subsistema lado volante se requiere un control PID de corriente. El actuador acoplado al volante brinda una retroalimentación de par, proporcional a la corriente consumida por el motor del lado rueda. Mientras tanto, en el subsistema lado rueda se realiza un control PID de posición angular, donde el actuador debe igualar la posición de referencia del volante. A continuación, se describen los esquemas de control a seguir para desarrollar este trabajo de tesis.

2.2.1. Esquema de control para la posición de un motor DC

La simplicidad y el basto conocimiento que se tiene de los motores DC, hacen que este tipo de máquinas eléctricas se consideren para aplicaciones donde se necesite controlar la posición angular. Lo que a su vez conlleva a requerir de una etapa de potencia, un dispositivo lógico programable y de instrumentos de medición que brinden la posición del eje del motor. Algunas de las opciones más comunes son: puentes H, microcontroladores, tarjetas de adquisición de datos y encoders (absoluto e incremental), respectivamente [40].

El esquema que se muestra en la Figura 2.3, es una adaptación del desarrollado por Rairán-Antoniles et al. en [40] y tiene por objetivo mostrar de manera general los elementos y su interacción para llevar a cabo la tarea de control.

El algoritmo de control se alimenta por una señal de error (e). Esta señal resulta de la diferencia entre los valores de referencia (r) y los valores de posición angular medida en el motor (b). La aplicación del algoritmo de control produce una señal de salida (u), en formato de modulación por

ancho de pulsos (PWM). Las señales PWM se suministran al puente H con el propósito de hacer funcionar al motor con cierta dirección de giro y potencia, hasta alcanzar los valores de referencia.

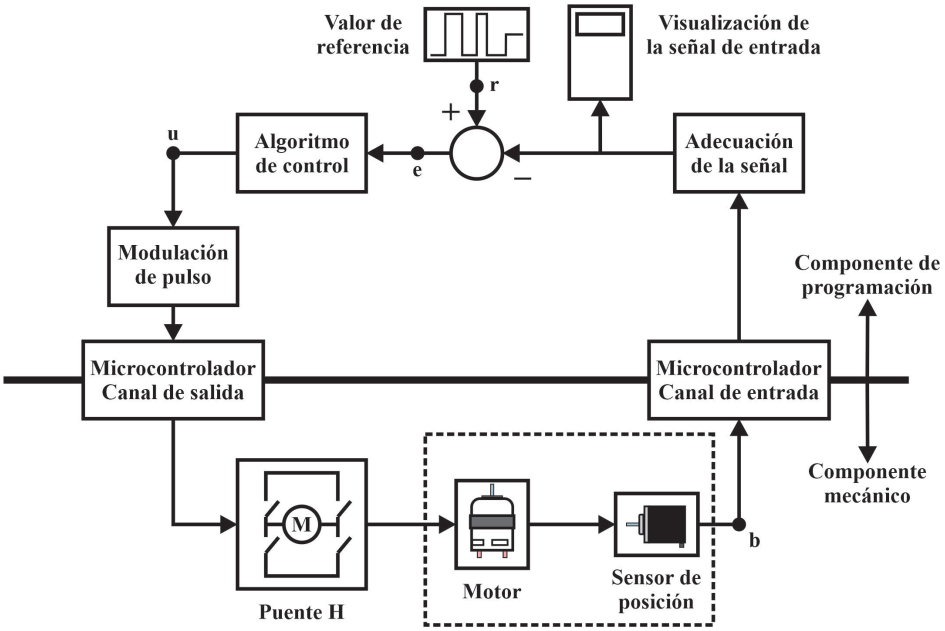

Figura 2.3: Esquema para controlar la posición angular de un motor DC. Adaptada de [40].

La señal de control (u) puede estar conformada por valores tanto negativos como positivos, dependiendo de si el error, es menor ó mayor a cero. Por lo tanto, esta señal se debe someter a un sencillo proceso de adecuación. Para que se suministre una conmutación PWM unipolar al puente H, con la corrección que hace el algoritmo de control. En la Figura 2.4, se observa que este proceso consiste en multiplicar a u por menos uno, siempre y cuando esta señal sea negativa.

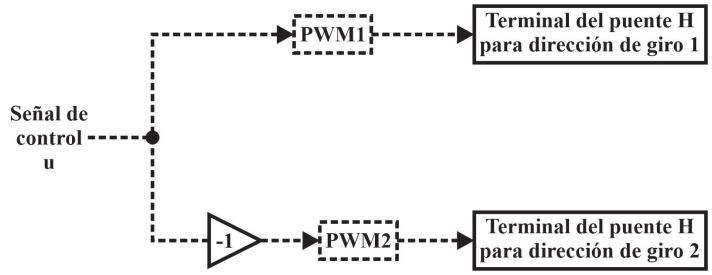

Figura 2.4: Manejo de la salida de control. Basada en [40].

2.2.2. Esquema de control de corriente para un motor DC

La teoría mostrada anteriormente, también puede ser llevada al control de corriente en motores DC, la lógica de control sigue siendo la misma. La única diferencia que existe está en el tipo de sensor que se utiliza (véase Figura 2.5).

El objetivo de este control es hacer girar al actuador en una determinada dirección y con cierto consumo de corriente. Para ello, es necesario el uso de sensores bipolares que tengan la capacidad de medir corriente eléctrica tanto negativa como positiva, proporcionando una señal de salida analógica ó digital (b). Este tipo de medición se debe realizar en serie, se coloca el sensor entre el elemento de potencia (puente H) y el motor.

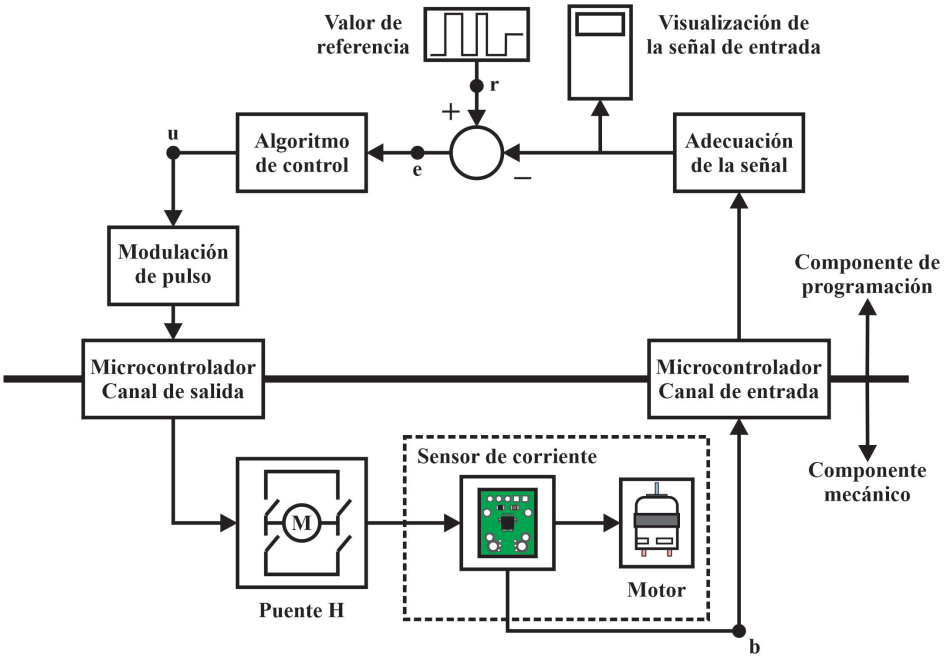

Figura 2.5: Esquema para realizar el control de corriente de un motor DC. Basada en Rairán-Antoniles et al. [40].

2.3. Algoritmo de control PID

El control PID es simple y robusto [41], es muy útil en plantas donde no se conoce su modelo matemático. Aporta un control satisfactorio de los sistemas [42], pero su sencillez limita el rango de las aplicaciones [43].

Su algoritmo de control es:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{2.1}$$

donde $u(t)$ es la señal de control, $e(t)$ es el error (valor de referencia - valor medido), K_p es la ganancia proporcional, K_i es la ganancia integral y K_d es la ganancia derivativa. En la Figura 2.6 se observa que la señal de control es la suma de un término proporcional (P), un término integral (I) y un término derivativo (D). Estas tres acciones operan de manera paralela y no interactuante [41].

Por otro lado, la forma estándar de la ec. 2.1 es:

$$u(t) = K(e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}) \tag{2.2}$$

donde K es la ganancia proporcional, T_i es el tiempo integral y T_d es el tiempo derivativo. En la forma estándar se factoriza la constante proporcional y se realiza un ajuste de las ganancias K_i y K_d (véase ecs. 2.3 y 2.4). Además, en la ec. 2.5 se observa al algoritmo de control PID en función de transferencia.

$$K_i = \frac{K}{T_i} \tag{2.3}$$

$$K_d = KT_d \tag{2.4}$$

$$u(s) = K(1 + \frac{1}{T_i s} + sT_d) \tag{2.5}$$

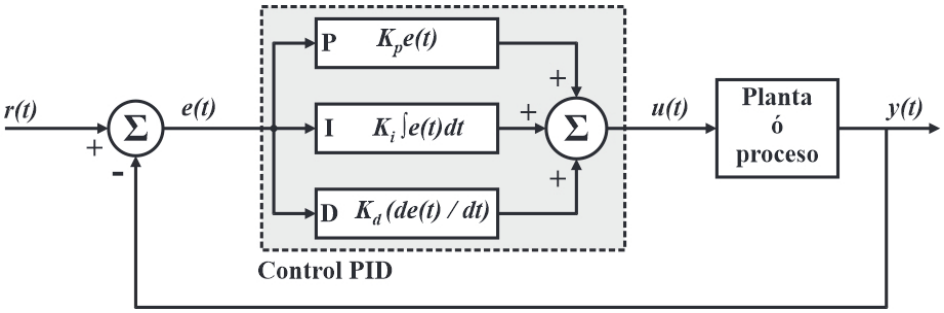

Figura 2.6: Composición en paralelo del esquema de control PID.

A continuación se describe la acción proporcional, la acción integral y la acción derivativa. Así como los efectos de sus ganancias en la respuesta de control. También se presenta la implementación del algoritmo de control en dispositivos lógicos programables.

2.3.1. Acción proporcional

Esta acción de control es proporcional al valor instantáneo del error. Cuando existe un error grande, la acción del control también lo es [43]. Por otro lado, aumentar la ganancia proporcional (K) conlleva a efectos positivos y negativos en el control del sistema. La parte positiva radica en el aumento de la velocidad de respuesta y en la disminución del error en régimen permanente.

Mientras tanto, los aspectos negativos se refieren a que conforme la ganancia proporcional va en aumento, también lo hace la inestabilidad del sistema. Por lo tanto, se debe encontrar un punto de equilibrio donde se tenga la mayor velocidad de respuesta y el más mínimo error, sin que el sistema sea demasiado inestable. Estas características se observan en la Figura 2.7, un ejemplo desarrollado por Åström y Hägglund en [41].

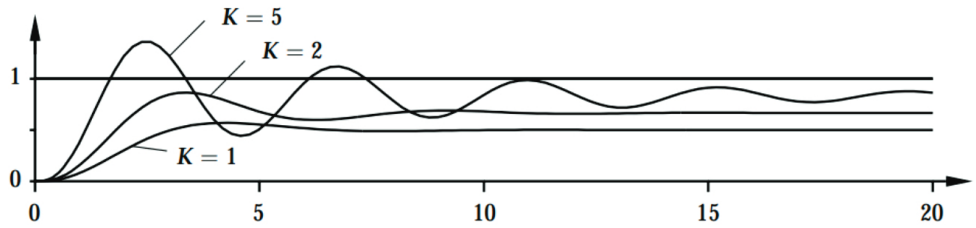

Figura 2.7: Respuesta de control variando la ganancia proporcional (K). Tomada de [41].

2.3.2. Acción integral

La accion integral brinda una señal de control que es proporcional al error acumulado [43]. Se asegura de que la salida del proceso coincida con el punto de referencia, reduciendo el error en estado estacionario del sistema.

Aumentar el tiempo integral (T_i) en valores finitos genera ciertos efectos en el control de un sistema. Se disminuye el error en estado estacionario, la inestabilidad del sistema y la velocidad de respuesta. Este último efecto es una desventaja de la acción integral. En la Figura 2.8 se observa que con valores pequeños se tiene una respuesta más rápida, pero su oscilación aumenta.

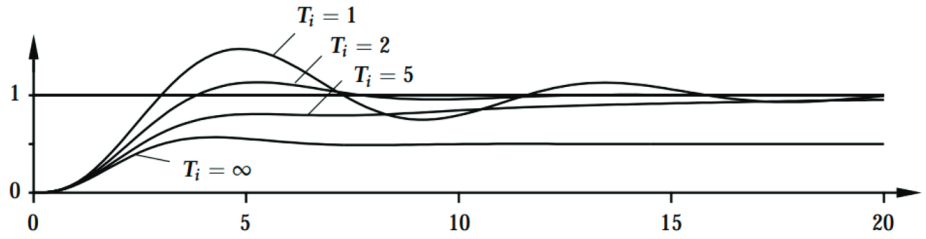

Figura 2.8: Respuesta de control variando el tiempo integral (T_i). Tomada de [41].

2.3.3. Acción derivativa

La acción derivativa trabaja sobre la tasa de cambio del error y mejora la estabilidad del sistema. Se basa en una predicción del error futuro [43]. Cuando el sistema tiene una velocidad de respuesta alta y se dirige al valor de referencia, se produce un sobrepulso y oscilaciones en torno a este valor.

Para evitarlo se emplea la acción derivativa. Al aumentar el tiempo derivativo T_d, se mejora la estabilidad del sistema hasta cierto punto, pero su velocidad de respuesta disminuye.

En la Figura 2.9 se observa que la acción derivativa sirve de amortiguamiento para la respuesta de control. Sin embargo, esta estabilidad disminuye si el tiempo derivativo T_d toma valores muy pequeños ó cada vez más grandes [41].

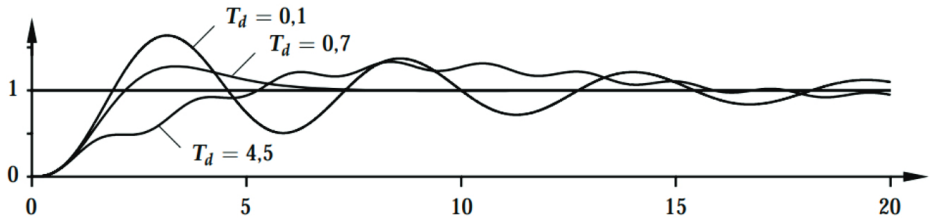

Figura 2.9: Respuesta de control variando el tiempo derivativo (T_d). Tomada de [41].

2.3.4. Sintonización del control PID

La sintonización de controladores PID consiste en encontrar valores para los parámetros de la ganancia proporcional (K), del tiempo integral (T_i) y del tiempo derivativo (T_d). Este ajuste se basa en las especificaciones de comportamiento requeridas y en la operación estable del sistema.

Para realizar el ajuste de estos parámetros existen diferentes tipos de reglas de sintonización. Las más comunes son las reglas de Ziegler-Nichols, que son: el método basado en la curva de reacción y el método de oscilación. Normalmente son usados cuando no se conocen los modelos matemáticos de las plantas. Estos métodos empíricos utilizan la respuesta al escalón unitario de la planta y proporcionan un punto de partida. Se complementan con una sintonía heurística menor [42]. En este trabajo de tesis solo se utiliza el método heurístico para encontrar las ganancias K_p, K_i y K_d, en los dos esquemas de control PID implementados (de corriente y de posición angular).

2.3.5. Implementación del algoritmo de control PID en microcontroladores

El algoritmo de control PID que se muestra en la ec. 2.1 se desarrolla en tiempo continuo. Para poder implementarse en un dispositivo lógico programable se necesita discretizar. En la etapa de discretización, una señal se debe representar en bloques ó intervalos de tiempo (Δt).

La integral del error se discretiza de la siguiente manera:

$$\int_0^{t_k} e(t)dt = \sum_{i=0}^{k} e(t_i)\Delta t \tag{2.6}$$

La derivada del error se aproxima con el método Backward Euler de la siguiente forma:

$$\frac{\mathrm{d}e(t_k)}{\mathrm{d}t} = \frac{e(t_k) - e(t_{k-1})}{\Delta t} \tag{2.7}$$

donde $e(t_k)$ es el error actual, $e(t_{k-1})$ es el error anterior y Δt es el tiempo de muestreo. El término proporcional se conserva igual (ganancia proporcional por el error instantáneo). A continuación, se presenta el pseudocódigo general que se implementa en los microcontroladores [44].

```
previous_error := 0
cum_Error := 0
loop:
    error := setpoint-measured_value
    // Integral del error
    cum_Error := cum_Error + error*dt
    // Derivada del error
    rate_Error := (error-previous_error) / dt
    // Salida de control
    output := Kp*error + Ki*cum_Error + Kd*rate_Error
    previous_error := error
    wait(dt)
    goto loop
```

En el pseudocódigo anterior se observa la obtención del error, el error acumulado (integral) y la tasa de cambio del error (derivada). Se considera un tiempo de muestreo (dt) y cada parámetro se multiplica con su respectiva ganancia (K_p, K_i y K_d). Los tres términos del controlador PID se suman para obtener la señal de control. Finalmente, en cada iteración se debe actualizar el error, como error pasado.

Capítulo 3

Diseño preliminar del sistema

En este capítulo se describe el análisis preliminar de diseño del sistema steer-by-wire. Se presentan sus especificaciones, características generales y de desempeño mediante una adaptación del estándar IEEE-830. Los aspectos a considerar del estándar con respecto al trabajo son:

- Perspectiva del sistema steer-by-wire.
- Funcionalidad del prototipo.
- Requerimientos específicos.
- Características de los usuarios.
- Diseño conceptual.
- Restricciones.

3.1. Perspectiva del sistema steer-by-wire

El presente sistema trata de un prototipo de dirección steer-by-wire. El objetivo del prototipo es: implementar dos esquemas de control PID descentralizados e interconectados mediante CAN Bus, que logren el funcionamiento de este tipo de dirección electrónica. Además, al ser un sistema de experimentación también se prueban mecanismos de seguridad.

La arquitectura de la dirección steer-by-wire se divide en dos subsistemas: subsistema lado volante (SLV) y subsistema lado rueda (SLR). En el primer subsistema se desarrolla un esquema de control de par con la corriente eléctrica medida. Mientras que, en el subsistema lado rueda se implementa un control de posición. A continuación, se presentan las características generales del prototipo de dirección electrónica steer-by-wire.

- **Nombre del sistema:** Módulo de dirección asistida tipo steer-by-wire basada en CAN Bus.

- **Qué hará el sistema:** Realiza un seguimiento de posición, usando como referencia el desplazamiento angular del volante. También simula una sensación de conducción empleando el seguimiento de corriente eléctrica. La retroalimentación de par más un error de posición añadido, ocasionan que el volante tienda a regresar a su posición central. El sistema entra en un estado seguro, cuando detecta que la corriente consumida supera el rango establecido durante cierto tiempo. Adicionalmente, envía los datos por puerto serial a una interfaz gráfica de despliegue de información y las condiciones de su funcionamiento son observadas a través de un led indicador.

- **Qué no hará el sistema:** No intercambia datos ni códigos de falla con otros sistemas. No incluye un sistema de diagnóstico estandarizado como es el OBD II. Las fallas detectadas son indicadas en la interfaz gráfica. Esta interfaz no es desarrollada para interactuar con el usuario. Además es un prototipo preliminar, por lo tanto es posible optimizar tamaño, funcionalidad y costos. Se trata de un sistema sin redundancia en sensores, actuadores, enlaces de comunicación y fuentes de alimentación como en el caso de los sistemas comerciales.

3.2. Funcionalidad del prototipo

Las funciones del sistema steer-by-wire se enlistan a continuación, con su respectiva descripción:

- **Control de dirección:** El subsistema lado rueda controla el ángulo de giro de las ruedas y envía información proporcional al par aplicado al lado volante. La posición de referencia viaja en el bus de comunicación CAN para ser utilizada por el esquema de control PID del SLR. Este seguimiento de posición angular necesita de dos encoders, uno en cada subsistema.

- **Retroalimentación de par:** El subsistema lado volante simula una sensación de conducción mediante un control indirecto de par. Las cargas aplicadas en el actuador del SLR aumentan y hacen variar su consumo de corriente eléctrica. Estas variaciones de corriente se ven reflejadas en el actuador conectado al volante.

- **Retorno a cero:** El sistema simula el regreso del volante a su posición central, como lo hace una dirección con acoplamiento mecánico.

- **Monitoreo de temperatura:** En el subsistema lado rueda se incluye un sensor de temperatura LM35, este sensor es colocado en la carcasa del actuador. Cuando existe una medición elevada, el controlador del subsistema determina la falla de sobrecalentamiento del motor. La información del monitoreo de temperatura es puesta a disposición en el bus de comunicación CAN y en el puerto serial.

- **Detección de fallas para la corriente eléctrica:** El sistema utiliza el monitoreo constante de la corriente eléctrica de los motores para la detección de fallas. En cada controlador se implementa un algoritmo, que detecta las lecturas de corriente superiores al rango de funcionamiento establecido. El algoritmo mide el tiempo en que se superan los umbrales para descartar los picos de corriente eléctrica.

- **Modo seguro:** En caso que las mediciones del sensor de corriente del SLV se encuentren fuera de los valores nominales, el controlador recurre a inhabilitar la realimentación de par. Esta desactivación no tiene repercusiones en el seguimiento del ángulo de dirección.

- **Graficación de señales para evaluar el desempeño:** El subsistema lado volante envía por puerto serial las señales de: seguimiento de corriente, ciclo de trabajo y compensación de retorno a cero. Por otro lado, las señales enviadas por el subsistema lado rueda son: seguimiento de posición, ciclo de trabajo y monitoreo de temperatura. Todas estas señales son importantes para el análisis de la dirección steer-by-wire.

- **Interfaz gráfica de datos y diagnóstico:** Los códigos de falla, la temperatura, la posición del volante, entre otros, son mostrados en la interfaz. Esta interfaz gráfica recibe la información desde el puerto serial del SLR. El controlador envía los datos cada 40 milisegundos.

- **Testigo led:** El sistema implementa este mecanismo de seguridad, para que el conductor tenga noción de las condiciones de funcionamiento. El led se ilumina en color azul si no existen fallas, en color amarillo para las alertas del SLR y en rojo para el modo seguro.

- **Enfriamiento por aire forzado:** Los ventiladores del sistema están encendidos constantemente para evitar el sobrecalentamiento de los actuadores.

3.3. Requerimientos específicos

Los requerimientos del sistema steer-by-wire se describen en la Tabla 3.1. En el hardware se proporcionan las especificaciones de los componentes electrónicos y mecánicos. También se presentan aspectos relacionados con el diseño de la plataforma experimental. Los requerimientos de software se enfocan en los esquemas de control y en el protocolo de comunicación.

Cada requerimiento es calificado de acuerdo a su prioridad. El valor de prioridad se indica en la Tabla 3.1, siendo 10 la prioridad más alta.

N°	Nombre del requerimiento	Descripción	Prioridad
		HARDWARE	
1	**Dimensionamiento del sistema**	La plataforma exprimental debe ser compacta. Dimensiones de la estructura del SLV: 245 *mm* de largo, 250 *mm* de ancho y 114 *mm* de altura. Para el chasis del SLR, las dimensiones consideradas son: 245 *mm* (largo) x 265 *mm* (ancho) x 200 *mm* (altura).	9
2	**Elemento operativo**	Volante estándar de 14 in. Colocado con un ángulo de inclinación de 14°.	9
3	**Elementos de transmisión**	Componentes estándar en poleas y engranes. Las poleas y bandas dentadas de tipo htd 3M serán utilizadas para transmitir potencia. Mientras que, las GT2 para transmitir movimiento.	9
4	**Simulación de par**	El sistema requiere de una parte mecánica para poder aplicar par. Por lo tanto, se colocará en el SLR un freno de disco manual.	9
5	**Tope mecánico**	El desplazamiento angular del volante debe estar limitado a 1440°. Se adaptará un mecanismo de dos engranes (engrane mayor modificado), en el espacio libre generado por el ángulo de inclinación.	10
6	**Etapa de potencia**	La corriente eléctrica consumida por cada motor DC, supera los 12 A. Por lo tanto, la mejor opción es una fuente de alimentación de 12 V, 50 A y 600 W. En el caso de los puentes H, se requiere un driver para alta potencia como el BTS7960.	10
7	**Dispositivo lógico programable**	Las tareas de control requieren de un microcontrolador con alto rendimiento. La placa de evaluación EK-TM4C123GXL cuenta con velocidad de procesamiento de 80 MHz y con los periféricos necesarios: 2 módulos CAN, 12 canales ADC, 2 módulos PWM, 8 UART e interfaz de encoder incremental.	10
8	**Módulos de comunicación**	Convertidor de niveles de voltaje para interfazar el bus CAN. Las tarjetas Tiva C requieren transceivers CAN SN65HVD230. Para la comunicación de la placa con el encoder absoluto, se usará un driver MAX3485 que convierte serial a RS-485 y viceversa.	10
		SOFTWARE	
1	**Lenguaje de programación**	El software del sistema será desarrollado en el lenguaje de programación C, en el entorno de Code Composer Studio. Con el soporte de un conjunto de librerías para los periféricos, proporcionadas en el kit de desarrollo de TivaWare.	10
2	**Esquemas de control**	Control PID de posición con un error máximo de $\pm$ 1 %. Control PID de par que no supere un error de $\pm$ 4 %. Este segundo control consistirá de un seguimiento de corriente eléctrica.	10
3	**Periodo de control**	Los algoritmos de control serán programados para realizarse cada 5 milisegundos.	10
4	**Enlace de comunicación**	Interconexión de los esquemas de control con un protocolo de comunicación seguro y rápido, como el CAN Bus. Velocidad de transmisión del CAN configurada a 1 Mbps.	10
5	**Señales PWM**	Debido a que los actuadores del sistema son motores DC, la frecuencia para las señales PWM se establece en 10 KHz.	10
6	**Interfaz gráfica**	La interfaz desarrollada en App Designer, debe desplegar información y funcionar como herramienta de diagnóstico. Especialmente, en las pruebas de los escenarios de falla. La cantidad de muestras a tomar es de 200 (opcional). La velocidad de transmisión serial debe ser configurada a 115200 bps.	9

Tabla 3.1: Requerimientos específicos de hardware y software para el sistema steer-by-wire.

3.4. Características de los usuarios

La Tabla 3.2 muestra las características generales de los usuarios. Incluye el nivel educacional y las áreas de desarrollo que aborda el sistema.

Tipo de usuario	Profesores y estudiantes.
Actividades	Desarrollo de esquemas de control para el seguimiento de posición y de par. Desarrollo de sistemas enfocados a mejorar la tolerancia a fallas.
Formación	Ingeniería y posgrado.

Tabla 3.2: Características generales de los usuarios.

3.5. Diseño conceptual

La Figura 3.1 presenta el bosquejo inicial del sistema de dirección steer-by-wire. La información que se muestra, es la interacción de los controladores con los sensores y los actuadores, además del enlace de comunicación CAN. Los datos que se transmiten en el bus de comunicación son: la posición medida en el SLV y la corriente eléctrica consumida por el actuador del SLR. Estos datos son leídos y usados como referencia en el controlador respectivo.

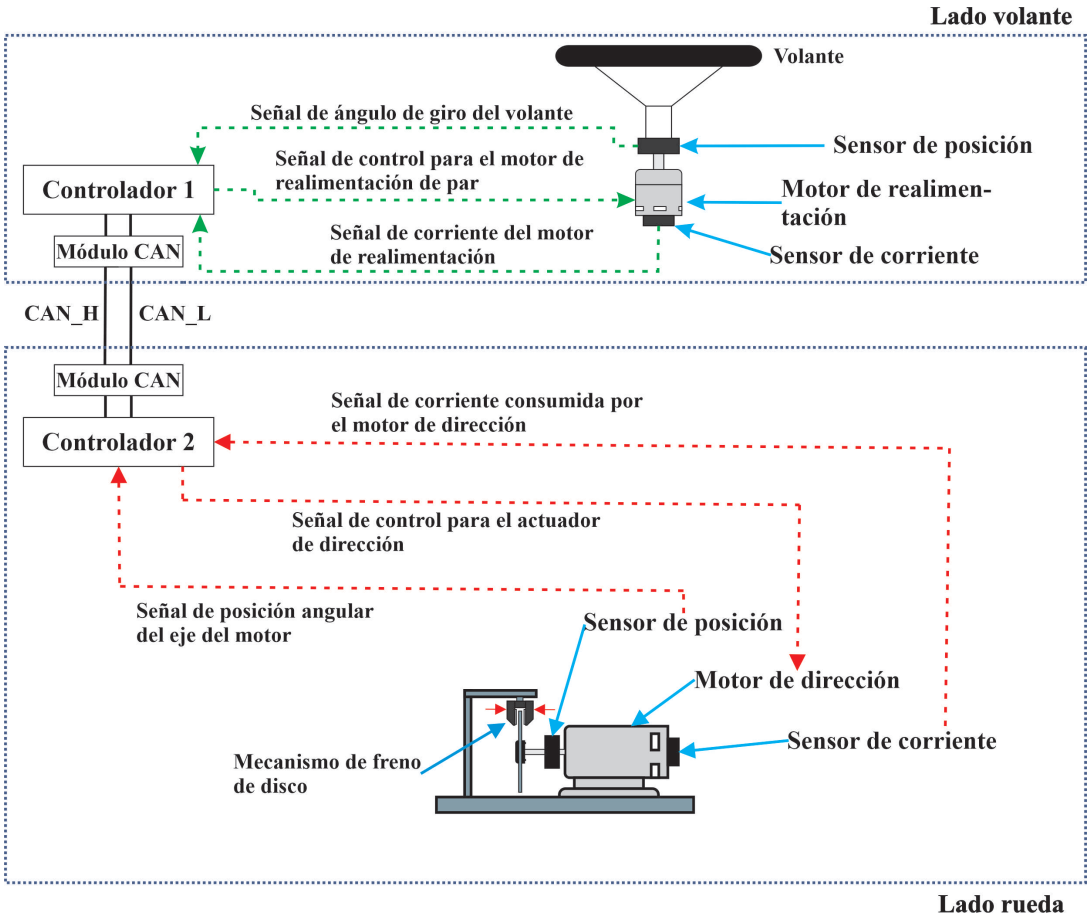

Figura 3.1: Esquema del diseño a seguir en el prototipo de dirección steer-by-wire.

El esquema anterior es el punto de partida para diseñar la plataforma experimental. Previo al diseño de las piezas y la realización de los ensambles en SolidWorks, se desarrolla el diseño conceptual del prototipo en cuestión. Resultado de ello son los dos subsistemas mostrados en las Figuras 3.2 y 3.3. Estos presentan la forma de cada estructura, así como la ubicación y distribución de elementos principales (actuador, sensor de posición y electrónica).

3.5.1. Subsistema lado volante (SLV)

Este subsistema se basa en la tendencia de volante electrónico para juegos de video, mejor conocido como *diy force feedback gaming steering wheel*. En este subsistema se distribuyen adecuadamente los elementos mecánicos, sensor, actuador y electrónica dentro de un cubo de 9738 cm^3 aproximadamente. Además el volante se coloca con una inclinación de 14°, la cual es similar a la inclinación de los sistemas comerciales destinados al *gaming*.

El ángulo del volante permite incorporar en el espacio inferior un tope mecánico capaz de limitar su desplazamiento angular similar a una dirección convencional. El mecanismo del volante, se compone de dos engranes rectos acoplados con relación 1:4. El engrane (engrane mayor) es modificado estructuralmente, con un área que no incluye espacios libres entre tres dientes. Esta zona del engrane bloquea el movimiento rotatorio de los ejes primario y secundario al entrar en contacto con los dientes del piñón.

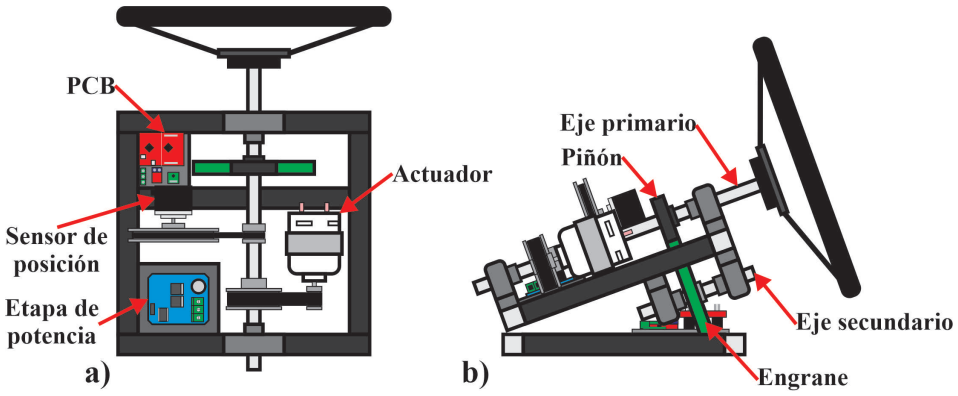

Figura 3.2: Boceto del subsistema lado volante a) vista superior y b) vista lateral.

3.5.2. Subsistema lado rueda (SLR)

El subsistema lado rueda es compacto, no es diseñado para integrar elementos de una dirección automotriz (ruedas, brazos y caja de dirección). La forma de su estructura permite colocar un eje y un freno de disco manual tipo bicicleta. La distribución de estos elementos, más la del actuador, el sensor y la electrónica, ocasionan que el SLR abarque un espacio de 18595 cm^3 aproximadamente.

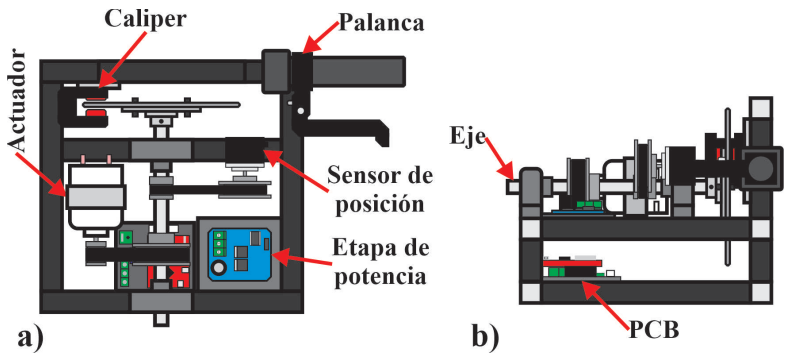

Figura 3.3: Boceto del subsistema lado rueda a) vista superior y b) vista lateral.

Este subsistema hace posible implementar el esquema de control de posición y simular de forma manual fuerzas de oposición que apliquen carga al motor.

3.6. Restricciones

A continuación, se describen los puntos que limitan el proceso de diseño y desarrollo del sistema, a nivel de software y de hardware.

- El despliegue de información en la interfaz gráfica no es en tiempo real.
- El par aplicado al volante (que se opone al movimiento del usuario) está limitado por el tipo de actuador utilizado.
- El mecanismo desarrollado para el lado rueda no representa un sistema de dirección real. Sin embargo, permite realizar pruebas de seguimiento de posición y retroalimentación de par.
- La medición de posición de giro en el lado rueda se realiza con un encoder incremental, lo cual implica que la posición se establece al encender el sistema. En un sistema real es necesario instalar un encoder absoluto.
- Las medidas de tolerancia a fallas instaladas son limitadas. El desarrollo de este tipo de mejoras quedan como trabajos futuros.
- Se trata de un sistema a nivel prototipo. Por lo tanto, es posible mejorar las características de sensores y actuadores. Esto puede mejorar el desempeño obtenido del presente diseño.
- Los motores DC del sistema desarrollado, están limitados a un consumo de corriente eléctrica de 15.5 A.

- Los motores seleccionados no están diseñados para aplicaciones de seguimiento de posición, el criterio de haberlos elegido es por su accesibilidad.
- Los sensores de posición utilizados no son de aplicación automotriz, sin embargo cumplen de sobra la resolución que se necesita.

Capítulo 4

Desarrollo de sistemas mecánicos

En este capítulo se describe el diseño y manufactura de la plataforma experimental. Se presenta el ensamblado físico de los materiales, mediante imágenes y explicaciones. El proceso de diseño asistido por computadora (CAD) es llevado a cabo con la herramienta de software de SolidWorks. El prototipo diseñado corresponde a una dirección steer-by-wire, que cuenta con dos subsistemas mecatrónicos interconectados mediante un enlace de comunicación basado en CAN Bus. Uno de estos módulos corresponde al subsistema lado volante (SLV) y el otro corresponde al subsistema lado rueda (SLR).

A continuación se describen las etapas de desarrollo del prototipo arriba mencionado:

- Diseño de subsistemas con la herramienta CAD.
- Manufactura de los subsistemas.

4.1. Diseño de subsistemas con la herramienta CAD

Esta etapa describe el diseño virtual del prototipo con el software de SolidWorks. Se muestran los ensambles de los subsistemas, con las dimensiones de sus estructuras y las ubicaciones de los elementos mecánicos y eléctricos que los integran. También se proporciona información específica del tope mecánico que distingue al SLV.

El SLV y el SLR son diseñados con base a un mismo elemento estructural, un perfil de aluminio 2020 con ranurado en V(ver Figura 4.1). Las ranuras con las que cuenta facilitan su ensamblaje y unión con otros elementos. Además de ser ligero y resistente.

A continuación se describe el proceso de creación de diseño 3D del prototipo steer-by-wire.

4.1.1. Creación de diseño 3D del SLV

El chasis de este subsistema es compacto, integrado por una plataforma inferior y una superior. En la Figura 4.2 se observa que sus medidas generales son: 245 *mm* de largo y 250 *mm* de ancho.

La colocación de las plataformas hace que el chasis tenga una altura superior de 113.86 *mm*, una altura inferior de 53.57 *mm* y un ángulo de 14.28°.

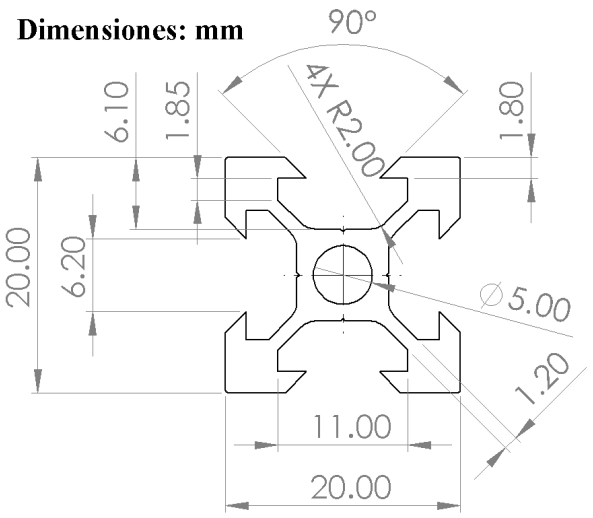

Figura 4.1: Dimensiones del perfil de aluminio 2020.

La estructura del SLV contiene dos soportes en forma de L y dos bisagras para perfil de aluminio 2020. Los soportes son colocados en las caras laterales, con una distancia de 32.61 *mm*. Esta medida determina el ángulo de inclinación del volante y solo es necesario desplazar los soportes para modificarlo.

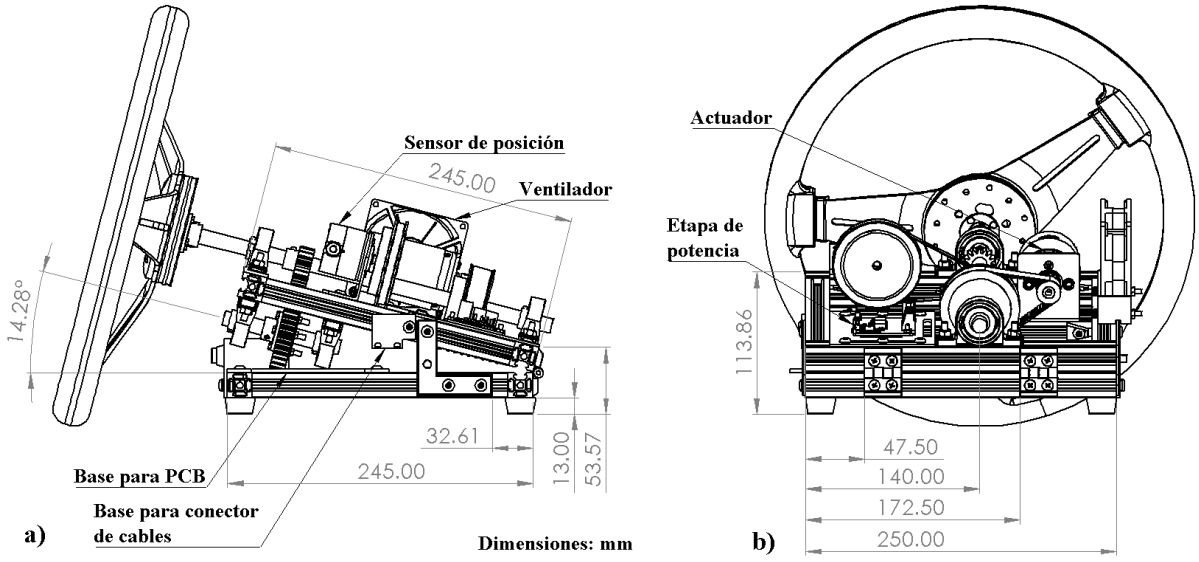

Figura 4.2: Distribución de elementos mecánicos y electrónicos en la estructura del SLV a) vista lateral izquierda y b) vista frontal.

A continuación se describen de forma separada las plataformas inferior y superior del SLV.

La plataforma inferior del SLV está integrada por 5 segmentos de perfil de aluminio 2020 (véase Figura 4.3). Su estructura es rectangular con las dimensiones mencionadas anteriormente (250 *mm* x 245 *mm*). Desde la vista superior se observa que en el interior del marco se coloca un segmento de perfil de 210 *mm* de largo. Este mismo segmento sirve de soporte para ubicar la electrónica (digital, lógica y de comunicación) en el cuadrante superior izquierdo.

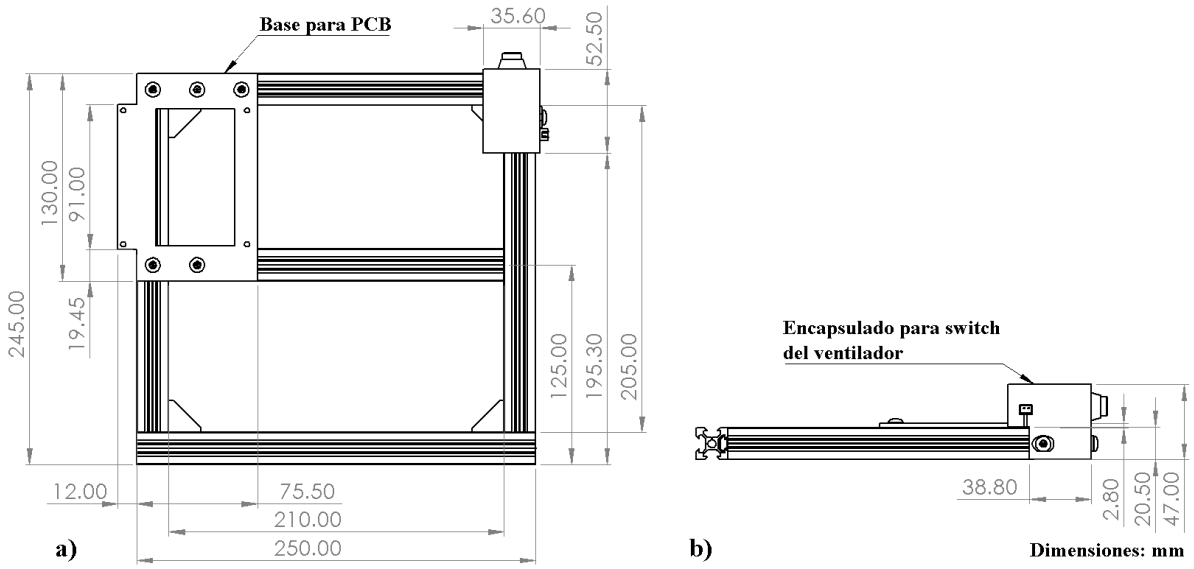

Figura 4.3: Dimensiones de la plataforma inferior del SLV y ubicación de la placa electrónica junto con el switch del ventilador a) vista superior y b) vista lateral derecha.

El switch para el enfriamiento del actuador necesita de un pequeño encapsulado de 49.5 cm^3. Esta pieza es ubicada de manera coincidente con la esquina superior derecha de la plataforma.

La plataforma superior del SLV conserva la misma forma rectangular que la inferior. Sus dimensiones de ancho y de largo son similares, a excepción de la cantidad de segmentos de perfil (véase Figura 4.4). En esta plataforma se integran tres segmentos interiores, con longitudes de 210 *mm* y 75 *mm*.

Esta parte del chasis contiene la mayor cantidad de elementos electrónicos y mecánicos del subsistema. Desde la vista superior de la Figura 4.4, se observa que el mecanismo del volante se ubica a 140 *mm* de la esquina inferior izquierda. El sensor de posición y el actuador son acoplados al eje primario del volante mediante poleas y bandas dentadas. Sus distancias con respecto a este eje son de 82 *mm* y 55 *mm*, respectivamente.

La etapa de potencia es colocada sobre una base de 84.5 *mm* de ancho por 108 *mm* de largo. Esta base cuenta con dos ranuras para sujetarse a la estructura y una más para enganchar los cables que se conecten al driver de potencia.

En la vista lateral se observa una base para sostener un conector de cables, es colocada a una

distancia de 105 *mm* para estar cerca de la zona del PCB en el ensamble completo (Figura 4.2). Existen otras 2 bases en la plataforma superior, la del led (cuadrante superior izquierdo) y la del ventilador ubicado al costado derecho del actuador.

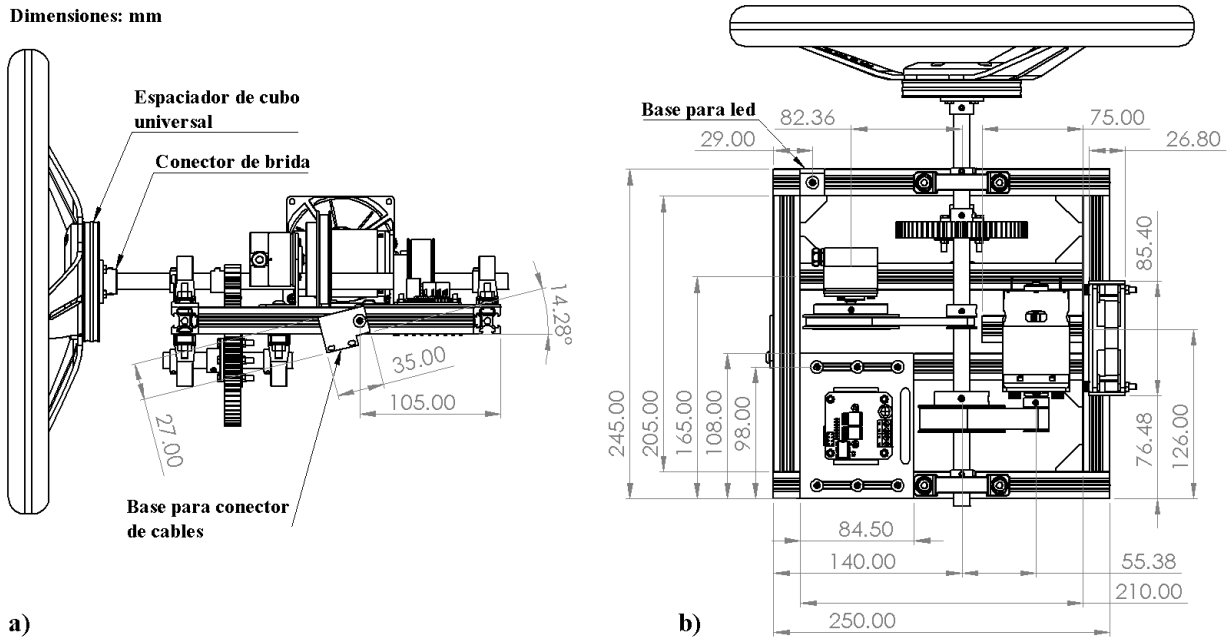

Figura 4.4: Dimensiones de la plataforma superior del SLV y ubicación de elementos del subsistema a) vista lateral izquierda y b) vista superior.

Por otro lado, la Figura 4.5 muestra de manera más detallada el mecanismo del volante, el cual está conformado por dos ejes de 12 *mm* de diámetro, dos poleas (htd 3M y GT2) y un par de engranes rectos. El eje primario tiene una longitud de 304 *mm* y el secundario de 96 *mm*. Para que sean montados en la plataforma superior se necesita de las chumaceras que se ven a los extremos.

El mecanismo del volante debe estar limitado a girar 1440°, es decir, cuando se posiciona en un punto medio solo puede girar dos vueltas a la derecha y dos a la izquierda.

La cantidad de vueltas permitida justifica la relación de engranaje establecida en el diseño conceptual del SLV (relación de engranes 1:4). Este dato junto con el valor del módulo y la distancia entre ejes mostrada en el ensamble de la Figura 4.6 son el punto de partida para el diseño del tope mecánico.

La distancia entre ejes es equivalente a la distancia entre centros de los dos engranes acoplados. Sin embargo, la distancia de 58 *mm* es una medida de referencia. La distancia entre centros se obtiene con el diámetro de paso del piñón y del engrane. En el caso del parámetro módulo, éste es seleccionado de manera arbitraria siendo de 1.5 $mm/diente$.

La fórmula del módulo es la siguiente:

$$m = \frac{d}{N} \tag{4.1}$$

donde m es el módulo $[mm/diente]$, d es el diámetro de paso $[mm]$ y N es el número de dientes.

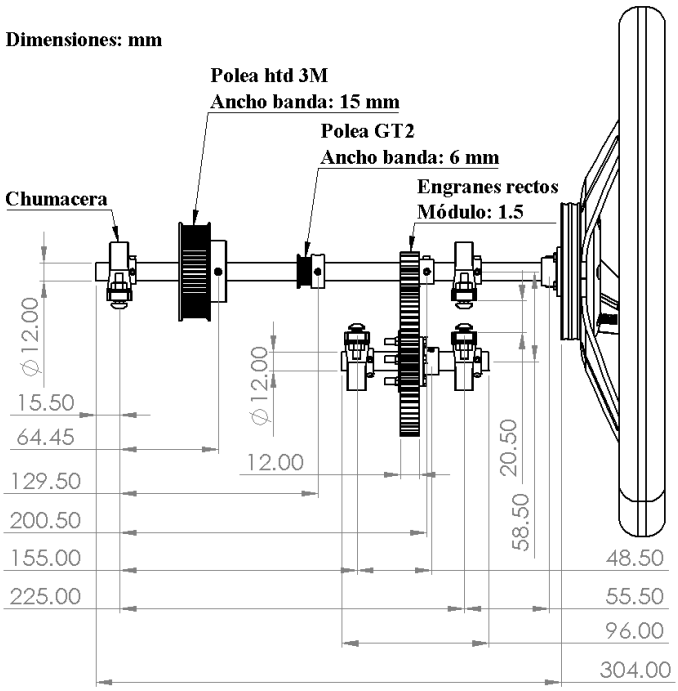

Figura 4.5: Ubicación de elementos mecánicos en el mecanismo del volante (vista lateral derecha).

Al despejar el diámetro de paso de la ec. 4.1 y considerando que el diámetro es dos veces el radio ($d = 2r$), se tiene que:

$$r = \frac{m \cdot N}{2} \tag{4.2}$$

donde r es el radio de paso $[mm]$. La cantidad de dientes del piñón (N_P) se determina de forma eurística. Al sustituir m y $N_P = 15$ *dientes* en la ec. 4.2 se obtiene el radio de paso del piñón, $r_P = 11.25$ *mm*.

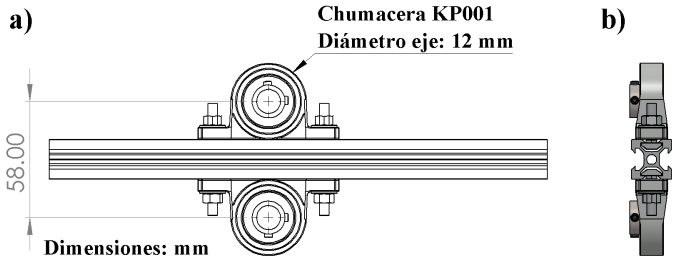

Figura 4.6: Distancia entre ejes usando como soporte dos chumaceras y un perfil de aluminio 2020 a) vista frontal y b) vista lateral derecha.

La distancia entre centros de un piñón y un engrane acoplado es igual al radio de paso del primero más el radio de paso del segundo, como se muestra en la siguiente ecuación.

$$D = r_P + r_G \tag{4.3}$$

donde D es la distancia entre centros [mm] y r_G es el radio de paso del engrane [mm]. Despejando r_G de la ec. 4.3 y sustituyendo valores, se obtiene que $r_G = 46.75\ mm$. En seguida se calcula el número de dientes del engrane (N_G). Para ello, se despeja este parámetro de la ec. 4.2. Cuando se sustituye r_G y el módulo m, resulta que $N_G = 62.33\ dientes$. Es evidente que este valor se debe tomar ya sea como 62 ó 63 dientes.

Con $N_G = 62\ dientes$ se obtiene un radio de paso $r_G = 46.5\ mm$ que sumado a r_P da una distancia entre centros $D = 57.75\ mm$. Mientras que con $N_G = 63\ dientes$, el radio de paso es igual a 47.25 mm y la distancia entre centros de 58.5 mm.

La elección que favorece al diseño y fabricación del subsistema es la de mayor cantidad de dientes, ya que solo se requiere aumentar 0.5 mm la distancia entre los ejes paralelos de los engranes para igualar su distancia entre centros.

En la Figura 4.7, se observa cómo este elemento mecánico al ser modificado queda con 60 dientes completos permitiendo que el piñón tan solo pueda dar 4 vueltas. Al ser un engrane con una característica tan especial, no se encuentra en el mercado y por lo tanto se debe fabricar. Las opciones para su manufactura son: maquinado CNC e impresión 3D. La impresión 3D es la opción más accesible y menos costosa.

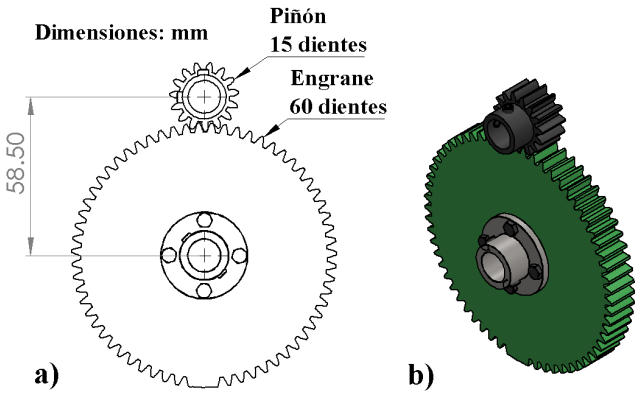

Figura 4.7: Tope mecánico mediante dos engranes rectos acoplados a) vista frontal y b) vista isométrica.

Finalmente, en la Figura 4.8 se observa el diseño completo del subsistema lado volante desde una vista isométrica.

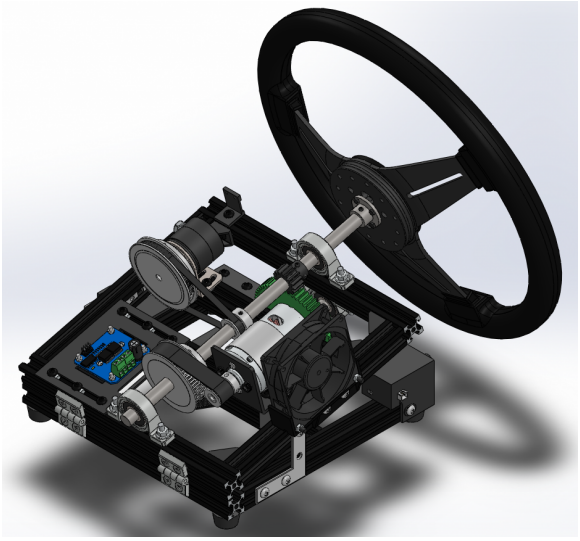

Figura 4.8: Diseño del SLV con SolidWorks.

4.1.2. Creación de diseño 3D del SLR

El subsistema lado rueda es diseñado para seguir una referencia de posición angular en un disco giratorio, al mismo tiempo permite aplicar par que se opone a su movimiento. Su chasis está dividido en dos niveles: nivel inferior y nivel superior. Las Figuras 4.9 y 4.10 muestran que el SLR consta de una estructura en forma de prisma rectangular y un marco de montaje para los elementos del freno de disco manual (palanca de freno y caliper).

Las dimensiones de la estructura prismática son: 245 *mm* de largo, 265 *mm* de ancho y 90 *mm* de alto. La colocación del marco de montaje en el nivel superior hace que la altura total del chasis sea de 213 *mm*.

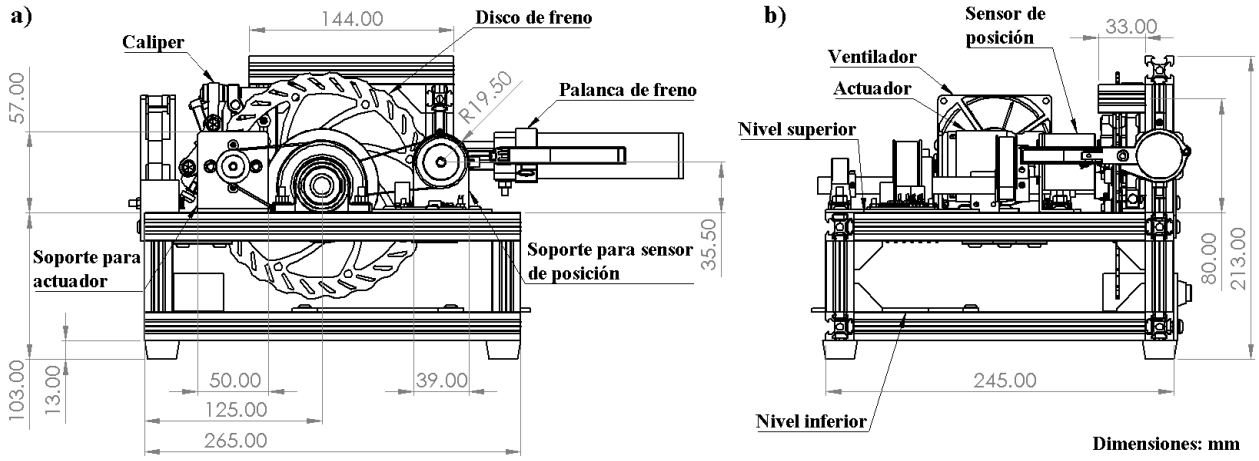

Figura 4.9: Distribución de elementos del SLR en los niveles inferior y superior del chasis a) vista frontal y b) vista lateral derecha.

El marco de montaje está integrado por tres segmentos de perfil 2020 de 90 *mm* y de 144 *mm* de longitud. Se atornilla al resto de la estructura mediante dos soportes en forma de L. La ubicación de uno de los soportes es de 84 *mm* con respecto a la esquina inferior derecha (véase Figura 4.10). La sujeción empleada permite modificar esta distancia e ir ajustando el contacto entre el disco de freno y el caliper.

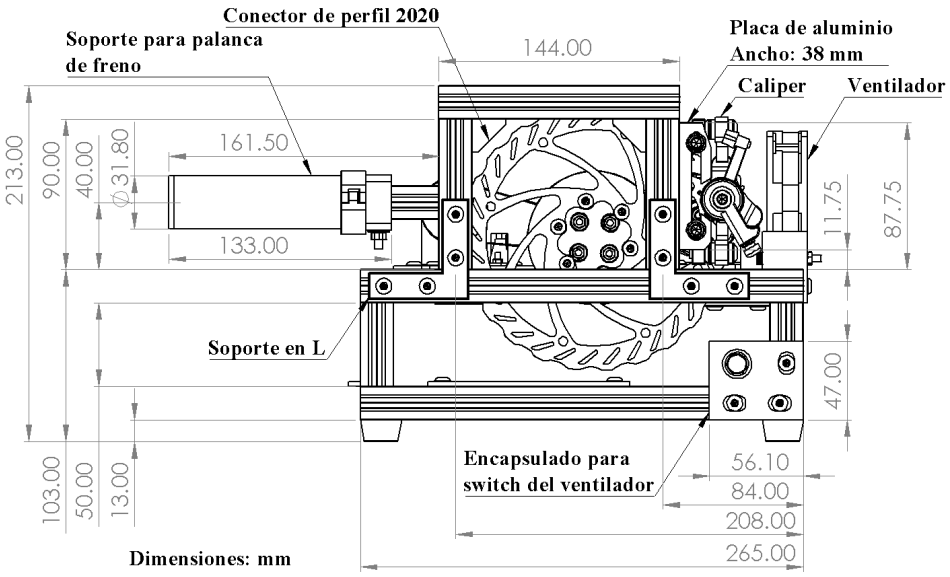

Figura 4.10: Vista posterior del subsistema lado rueda.

La Figura 4.10 muestra que el soporte para la palanca de frenos es empotrado al marco de montaje. Su altura sobre la superficie del nivel superior es de 40 *mm*. Por otro lado, el caliper es atornillado a una placa de aluminio ubicada al extremo derecho del marco con una altura de 11.75 *mm*.

A continuación se describen de forma separada los niveles inferior y superior del subsistema lado rueda.

El nivel inferior del SLR está constituido por una estructura rectangular. Sus dimensiones de largo y de ancho son las mismas que las descritas anteriormente (245 *mm* x 265 *mm*). La vista superior de la Figura 4.11 muestra que en el interior se ubica un segmento de perfil de 225 *mm* de longitud. Este elemento estructural es colocado horizontalmente justo en el centro de esta base del chasis.

La electrónica del SLR (digital, lógica y de comunicación) es ubicada en el cuadrante inferior derecho a una distancia de 40 *mm* de la esquina. En esta zona también se tiene una base para un conector de cables. El único tornillo con el que se sujeta esta pequeña plataforma se coloca a 102 *mm* de la misma esquina de referencia.

El encapsulado para el switch del ventilador es similar en volumen al implementado en el SLV.

Se ubica en el cuadrante superior izquierdo de la estructura rectangular y es atornillado al chasis cómo se muestra en la Figura 4.10.

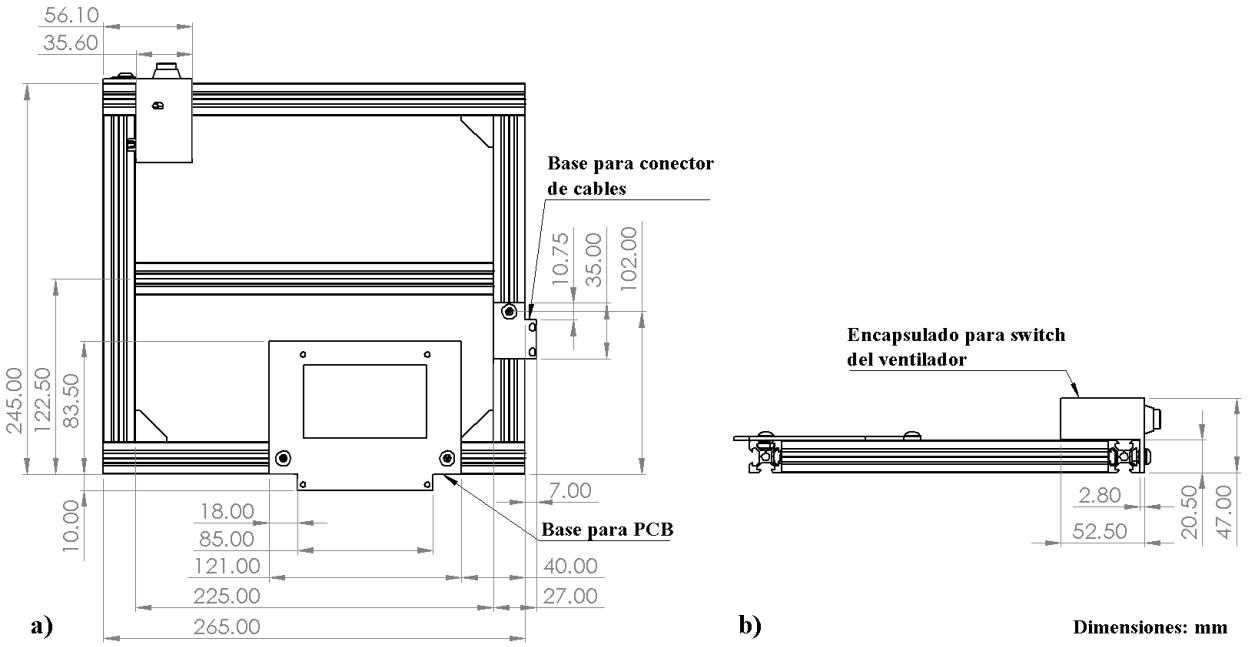

Figura 4.11: Dimensiones del nivel inferior y ubicación de ciertas bases para elementos electrónicos del SLR a) vista superior y b) vista lateral derecha.

Por otro lado, el nivel superior es la parte más completa del SLR. El contorno de su estructura es similar al del nivel inferior (véase Figura 4.12). Sin embargo, la cantidad de segmentos de perfil que la integran aumenta considerablemente. En el interior de la estructura se colocan tres segmentos, con 225 *mm* y 95 *mm* de longitud.

La Figura 4.12 muestra un eje con un disco de freno en su extremo. Esta flecha mecánica es ubicada a 140 *mm* de la esquina inferior derecha. El sensor de posición y el actuador del SRL son acoplados al eje mediante poleas y bandas dentadas. Sus distancias con respecto a este último son de 83.89 *mm* y 62.56 *mm*, respectivamente.

La base que sostiene la etapa de potencia se ubica en el cuadrante inferior derecho. Sus dimensiones y forma geométrica son exactamente igual a la del subsistema anterior. Las otras bases son: la del ventilador y la del sensor de temperatura LM35. El ventilador se posiciona al costado izquierdo del actuador, mientras que el sensor de temperatura al otro extremo. La base para el sensor de temperatura ocupa una superficie de 7.68 cm^2. Sus dimensiones son las adecuadas para colocarlo en el espacio libre que existe entre el actuador y el eje.

El disco de freno se sujeta al eje del SLR a través de una maza. En la Figura 4.13 se observa que esta parte es formada por la unión de un conector de brida y una placa circular de aluminio de 80

mm de diámetro.

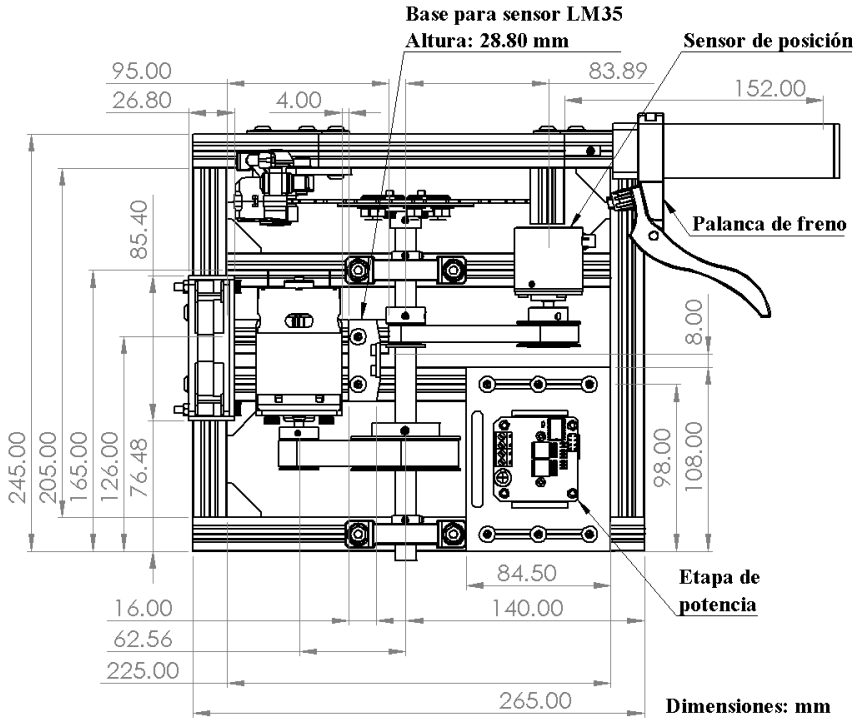

Figura 4.12: Dimensiones del nivel superior y ubicación de elementos del SLR (vista superior).

El eje tiene una longitud de 208 *mm* y es montado en la estructura del nivel superior mediante dos chumaceras colocadas a 155 *mm* una de la otra. Las poleas que se colocan en el eje son del mismo tipo que las utilizadas en el subsistema lado volante.

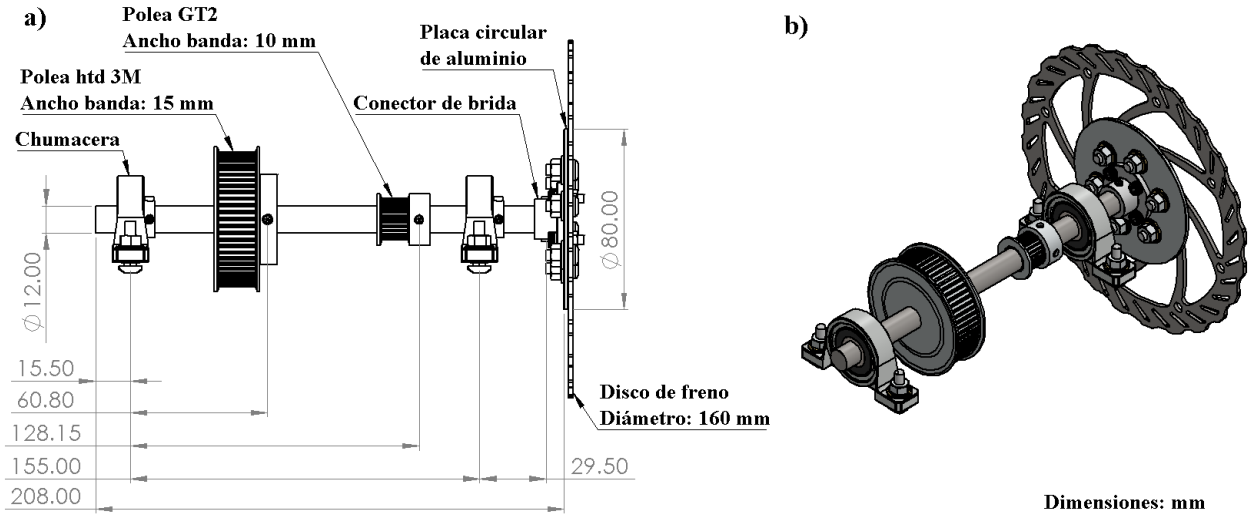

Figura 4.13: Ubicación de elementos mecánicos en el eje del SLR a) vista lateral derecha y b) vista isométrica.

En la Figura 4.14 se muestra el diseño final del subsistema lado rueda desde una perspectiva isométrica.

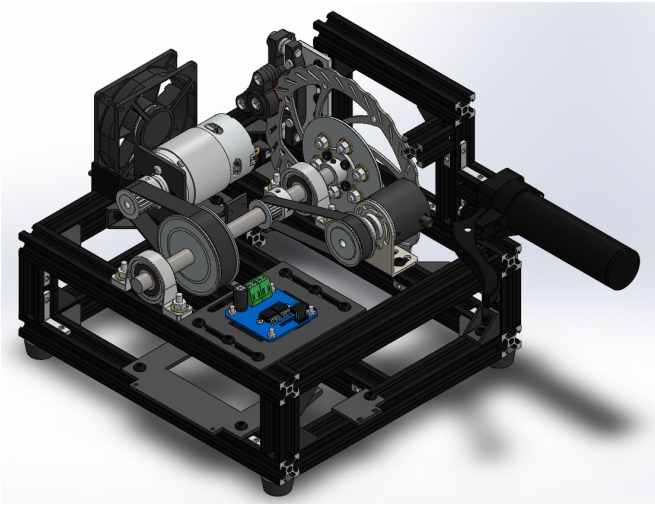

Figura 4.14: Diseño del SLR con SolidWorks.

4.2. Manufactura de los subsistemas

Esta etapa de desarrollo describe la construcción física de la plataforma experimental. Se presentan sus materiales, los ensamblajes físicos de cada chasis y la colocación de los elementos mecánicos y electrónicos.

Las estructuras de ambos subsistemas contienen como elementos de sujeción: tornillos y tuercas M5, acompañados de conectores y soportes para perfiles de aluminio 2020. Se emplean dos tipos de conectores: internos y de esquina (ver Figura 4.15). El tipo de unión que tienen los segmentos estructurales y las bases de los elementos electrónicos, facilitan el mantenimiento futuro del banco de pruebas.

Figura 4.15: Conectores de perfil de aluminio ranurado 2020 a) de esquina y b) interno.

A continuación, se describe el proceso de fabricación de la plataforma experimental.

4.2.1. Construcción física del diseño del SLV

En la sección anterior se muestran los elementos necesarios para la fabricación de los subsistemas que conforman a la plataforma experimental. Por ejemplo, en el subsistema lado volante se requieren 4 chumaceras KP001 para ejes de 12 *mm* de diámetro.

Los ejes que se utilizan son segmentos de varilla de acero inoxidable. Estos segmentos junto con los de perfil de aluminio 2020, son cortes de piezas más largas. La Tabla 4.1 resume las cantidades de segmentos y sus longitudes.

Cantidad (uds)	Material	Longitud [*mm*]
4	Perfil de aluminio ranurado 2020	250
4	Perfil de aluminio ranurado 2020	205
3	Perfil de aluminio ranurado 2020	210
1	Perfil de aluminio ranurado 2020	75
1	Varilla de acero inoxidable (diá. 12 *mm*)	96
1	Varilla de acero inoxidable (diá. 12 *mm*)	304

Tabla 4.1: Segmentos de perfil 2020 y de varilla de acero inoxidable para el SLV.

La construcción del SLV comienza por su chasis. En el diseño 3D se mencionan dos plataformas que lo integran: la inferior y la superior. La plataforma inferior (base de la estructura) es la primera en ser construida. En ella se colocan los dos soportes en forma de L y las dos bisagras para perfil de aluminio 2020 que sujetan a la otra plataforma. La Figura 4.16 muestra a la plataforma inferior de forma física.

Figura 4.16: Construcción física de la plataforma inferior del SLV.

Los segmentos de perfil de aluminio que integran el marco rectangular requieren de 4 conectores de esquina para su unión. Mientras tanto, el segmento central se sujeta al resto de la estructura con 4 conectores internos.

En la Figura 4.17 se observa la construcción de la plataforma superior. En ella están colocadas las dos chumaceras del eje primario, el soporte para el sensor de posición y el del actuador. Los dos soportes se encuentran en el mercado, siendo innecesaria su fabricación.

La estructura de esta plataforma requiere de un total de 8 conectores de esquina y 4 conectores internos. Además, las chumaceras se fijan a ella mediante tornillos M5 tipo T ó cabeza de martillo. La forma de estos tornillos les permite anclarse a los perfiles de aluminio cuando son introducidos en las ranuras.

Figura 4.17: Construcción física de la plataforma superior del SLV.

El subsistema lado volante contiene ciertas piezas que requieren fabricarse con impresión 3D. Estos elementos son las bases de colocación para: etapa de potencia, ventilador, led indicador, placa de circuito impreso (PCB), conector de cables e interruptor de enfriamiento. El engrane del tope mecánico también es fabricado de la misma manera. Sin embargo, esta pieza necesita elaborarse con la mayor cantidad de relleno de material, para ser resistente a las fuerzas inflingidas por el movimiento del volante.

La Figura 4.18 muestra que el engrane fabricado no tiene un cubo propio. Este cubo es la adaptación de un conector de brida de acoplamiento. El material con el que está fabricado es acero y los prisioneros con que cuenta hacen que el engrane se sujete de forma rígida al eje secundario del SLV, esto no sería posible con un cubo hecho de plástico.

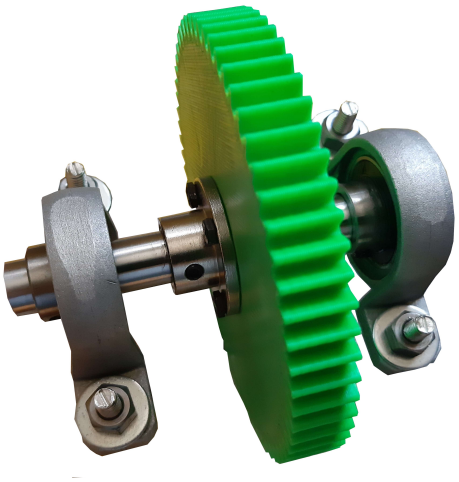

Figura 4.18: Colocación de elementos mecánicos en el eje secundario del SLV.

Una vez obtenido el ensamblaje físico del eje secundario con las chumaceras y el engrane recto, se coloca por debajo de la plataforma superior. En la Figura 4.19 se observa al chasis completo del SLV con algunos de los elementos mecánicos y de soporte que integran el subsistema lado volante.

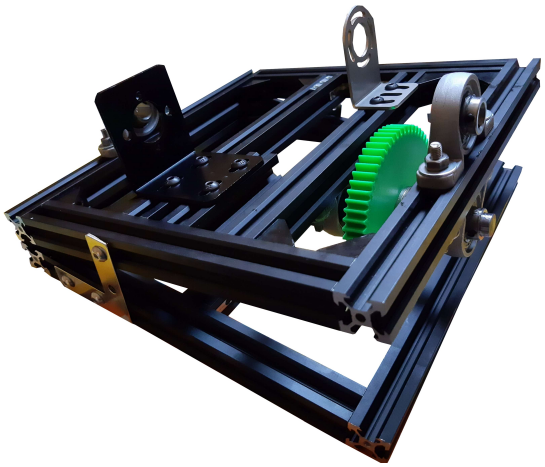

Figura 4.19: Colocación del eje secundario en el chasis del SLV.

Los siguientes elementos que se incluyen son: la etapa de potencia, el sensor de posición (encoder absoluto) y el actuador (motor DC 895). Las dos poleas centrales y el piñon del tope mecánico deben ser ubicados en el eje primario de acuerdo a las distancias indicadas anteriormente (ver Figura 4.5). Las demás poleas son colocadas en el eje del actuador y en el eje del sensor de manera alineada con las otras. La Figura 4.20 muestra el resultado de integrar en el chasis del SLV todos los elementos mencionados.

La banda y las poleas instaladas para aplicar par al volante son de tipo htd 3M. El tamaño de sus dientes las hacen adecuadas para la transmisión de potencia. El ancho y el perímetro de la banda

es de 15 *mm* y 240 *mm*, respectivamente. Las poleas son de 15 y 60 dientes, estas cantidades se seleccionaron considerando el tamaño del mecanismo y valores estándar en el mercado. Además, la polea conductora tiene el diámetro menor para formar un mecanismo reductor de velocidad, que amplifique el par motor transmitido.

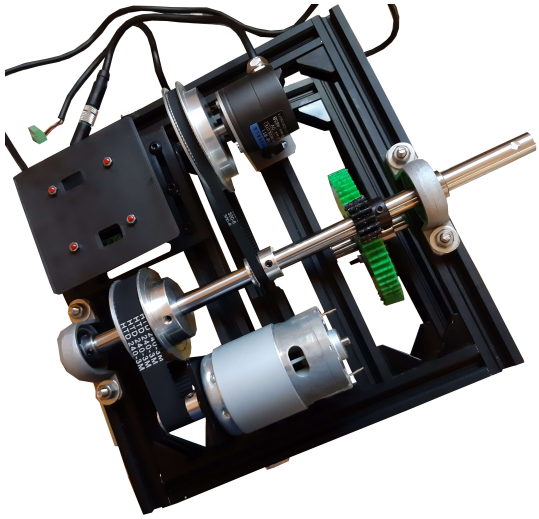

Figura 4.20: Integración del eje primario junto con el resto de elementos en el chasis del SLV.

Por otro lado, la banda y las poleas que conectan al eje primario con el sensor de posición son de tipo GT2. Este mecanismo de transmisión no se somete a grandes esfuerzos. El ancho de la banda es de 6 *mm* y su perímetro de 300 *mm*. La relación entre las cantidades de dientes de las poleas coincide con la relación de engranaje 1:4. Esto permite que el desplazamiento angular del mecanismo del volante (1440°) sea interpretado en el rango de medición del sensor de posición (360°). La cantidad de dientes seleccionada para la polea conductora y para la polea conducida es de 25 y 100 dientes, respectivamente.

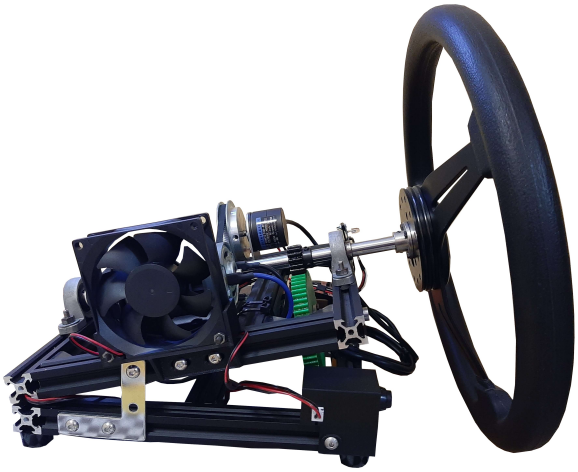

Figura 4.21: Resultado final de la construcción física del subsistema lado volante.

La Figura 4.21 muestra la construcción del SLV terminada. El volante necesita una adaptación para ser colocado en el eje primario. Se utiliza un conector de acoplamiento atornillado a un espaciador de cubo universal. En la vista lateral izquierda de la Figura 4.4 se indican los elementos de esta adaptación.

4.2.2. Construcción física del diseño del SLR

Las planos mostrados en la creación de diseño 3D del SLR, indican las longitudes de corte de los segmentos de perfil de aluminio 2020 y del segmento de varilla de acero inoxidable. La Tabla 4.2 presenta las cantidades de los segmentos y sus longitudes. El número de elementos estructurales es mayor que en el subsistema anterior.

Cantidad (uds)	Material	Longitud [*mm*]
4	Perfil de aluminio ranurado 2020	265
4	Perfil de aluminio ranurado 2020	205
4	Perfil de aluminio ranurado 2020	50
3	Perfil de aluminio ranurado 2020	225
1	Perfil de aluminio ranurado 2020	95
2	Perfil de aluminio ranurado 2020	90
1	Perfil de aluminio ranurado 2020	144
1	Perfil de aluminio ranurado 2020	33
1	Perfil de aluminio ranurado 2020	152
1	Varilla de acero inoxidable (diá. 12 *mm*)	208

Tabla 4.2: Segmentos de perfil 2020 y de varilla de acero inoxidable para el SLR.

El ensamblaje físico del chasis comienza por el nivel inferior. Su construcción consiste en armar un marco rectangular con un segmento de perfil en su centro. De forma vertical, en cada esquina se coloca un elemento estructural de 50 *mm* de longitud. Estos 4 segmentos funcionan de pilares para colocar el nivel superior de la estructura (véase Figura 4.22).

La estructura rectangular contiene 4 conectores de esquina y requiere de 4 conectores internos para sujetar el segmento central. Los segmentos de soporte se anclan al nivel inferior combinando los conectores para perfil de aluminio 2020 (4 conectores de cada tipo).

La Figura 4.23 muestra la construcción del nivel superior del SLR. Esta estructura integra la misma cantidad de conectores que la plataforma superior del SLV (8 conectores de esquina y 4 internos). Las tuercas y los tornillos M5 distribuidos en la estructura son el medio de sujeción para

el soporte del actuador, el soporte del sensor de posición y la base de la etapa de potencia.

Figura 4.22: Ensamblaje físico del nivel inferior con los segmentos de soporte del SLR.

El ensamble del nivel superior con el nivel inferior origina la estructura en forma de prisma rectangular, mencionada anteriormente en el diseño 3D. En ella se posiciona el marco de montaje para los elementos del freno de disco manual. Su unión requiere de 2 conectores internos y de 2 soportes en forma de L.

Figura 4.23: Ensamblaje físico del nivel superior del SLR.

En la Figura 4.24 se observa el resultado de colocar la palanca de freno y el caliper. También son integrados los soportes del actuador y del sensor de posición, las dos chumaceras KP001 del eje y la base para la etapa de potencia. Esta base y el soporte para la palanca de freno son fabricados mediante impresión 3D. La forma cilíndrica del soporte es para la abrazadera de la palanca y para brindar comodidad al accionarla. Además, la sección vacía en su interior permite adaptarle un segmento de perfil 2020 de 152 *mm* de longitud, que funciona como elemento de anclaje.

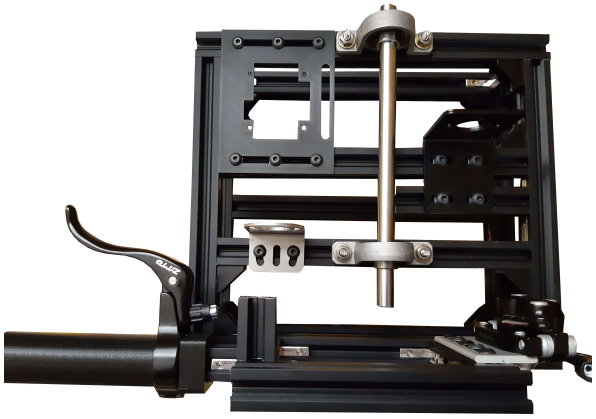

Figura 4.24: Colocación de los elementos del freno de disco manual en el chasis del SLR.

El segmento de perfil de aluminio más corto es integrado como medida de protección en caso que el disco de freno deje de estar sujetado al eje. Este segmento junto con el soporte para palanca de freno son empotrados en el marco de montaje con 1 y 2 conectores internos, respectivamente.

Los siguientes elementos mecánicos y electrónicos que se incluyen en el SLR se observan en la Figura 4.25. La etapa de potencia, el actuador (motor DC 895) y el sensor de posición (encoder incremental) son colocados en sus bases correspondientes.

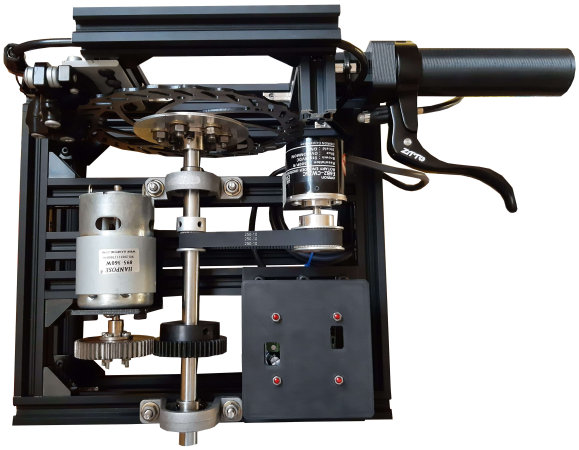

Figura 4.25: Colocación de un mecanismo de engranaje en el subsistema lado rueda.

El freno de disco manual requiere de un cable de accionamiento, que es introducido en una de las ranuras longitudinales del marco de montaje. El disco de freno se coloca en el eje mediante una pequeña maza fabricada. Los elementos que forman esta maza se encuentran indicados en la vista lateral derecha de la Figura 4.13.

En el eje también se integran los elementos mecánicos que lo conectan con el sensor de posición y el actuador. Las características de estos elementos son seleccionadas conforme a las dimensiones

del SLR y valores estándar en el mercado. La banda y las poleas dentadas son de tipo GT2. El ancho de la banda es de 10 *mm* y su perímetro de 250 *mm*. Las poleas son de 25 y 50 dientes. La relación de transmisión de las poleas permite ampliar el rango de desplazamiento angular del eje.

El subsistema lado rueda incorpora un mecanismo de engranaje de prueba para acoplar al eje con el actuador. Cada engrane es de 48 dientes y el valor de su módulo es de 1. La distancia entre centros es de 48 *mm*, suficiente para que la superficie del actuador no entre en contacto con el eje. El cubo del piñón es una adaptación con un conector de brida, similar al mostrado en el SLV.

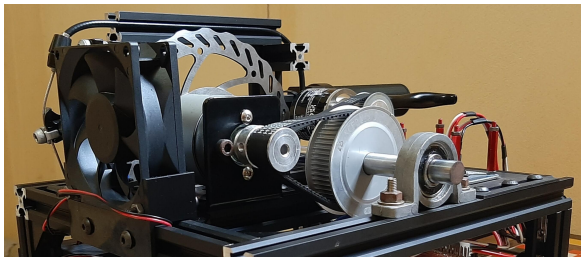

Figura 4.26: Transición del mecanismo de engranaje al mecanismo de poleas en el SLR.

La transición al mecanismo de poleas dentadas tiene como propósito: disminuir el ruido generado por el movimiento rotatorio y mejorar la precisión de la posición angular. La Figura 4.26 muestra el cambio en el acoplamiento. La banda y las poleas son de tipo htd 3M, similares a las utilizadas en el subsistema lado volante. A excepción, del perímetro de la banda (255 *mm*) y la cantidad de dientes de la polea conductora (20 dientes).

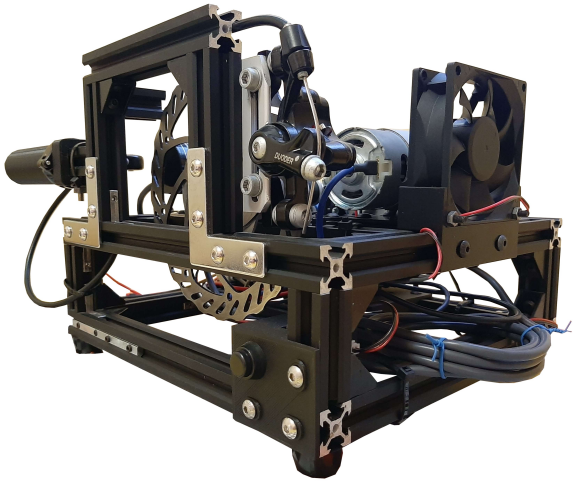

Figura 4.27: Resultado final de la construcción física del subsistema lado rueda.

Los últimos elementos en ser incorporados al SLR son: el ventilador, el sensor de temperatura LM35, el switch de enfriamiento, la base para la placa de circuito impreso (PCB) y la base del conector para cables. La construcción finalizada de este subsistema se observa en la Figura 4.27.

Capítulo 5

Desarrollo de sistema electrónico

El subsistema lado volante (SLV) y el subsistema lado rueda (SLR) cuentan con módulos de electrónica, los cuales corresponden a:

- Potencia
- Lógica programable y datos
- Instrumentación

En la etapa de potencia están los drivers de puente H y la fuente de alimentación conmutada. En lógica programable se encuentran las tarjetas de desarrollo y en datos, los módulos de comunicación CAN y RS-485. Mientras tanto, la instrumentación tiene que ver con las interfaces de acoplamiento de voltaje. A continuación, se describe cada uno de los módulos de electrónica del sistema de dirección steer-by-wire. También se presenta el diseño y desarrollo de las placas de circuito impreso que integran los elementos antes mencionados.

5.1. Potencia

El diagrama eléctrico del sistema steer-by-wire se muestra en la Figura 5.1. La fuente de alimentación brinda un voltaje de 12 V y GND. Se adapta un regulador de voltaje, para que cada subsistema sea alimentado también con 5 V. Estos niveles de voltaje y GND son distribuidos al SLV y al SLR mediante un bloque de conectores de cables.

En la Figura 5.2 se presentan las conexiones eléctricas realizadas en ambos subsistemas. Los elementos principales de este circuito eléctrico son: el motor DC, el puente H (driver BTS7960), el sensor de corriente (ACS711EX) y el ventilador. El bloque empalme une GND de la fuente con la tierra del dispositivo lógico programable. La placa de circuito impreso es alimentada de forma externa por los niveles de voltaje de la fuente. El ventilador se conecta a la salida de 12 V de la placa y las terminales del módulo de puente H (VCC, R_EN y L_EN) a la salida de 5 V. El

nivel de voltaje aplicado en R_EN y L_EN habilita la función de avance y retroceso del motor, respectivamente.

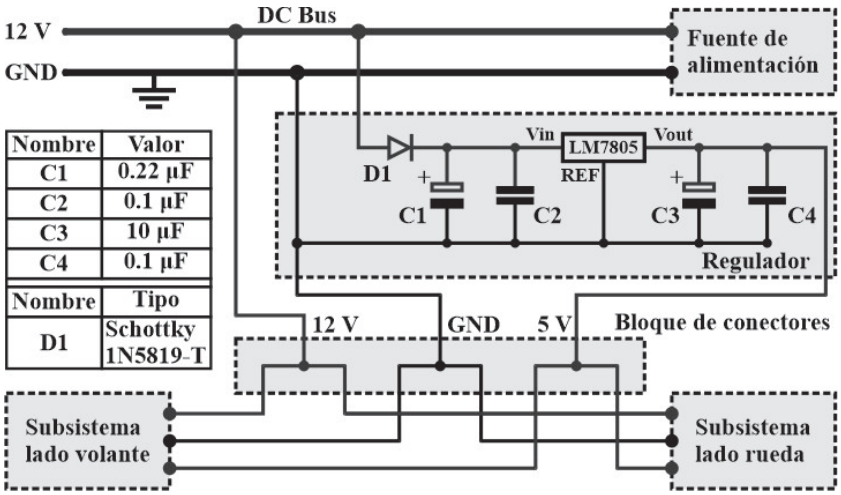

Figura 5.1: Diagrama eléctrico del prototipo de dirección steer-by-wire.

El driver de puente H recibe la alimentación de potencia en sus borneras B+ (12 V) y B- (tierra común). Esta potencia es entregada al motor en las terminales restantes (M+ y M-). Además, se coloca un sensor en serie para medir la corriente consumida por el actuador.

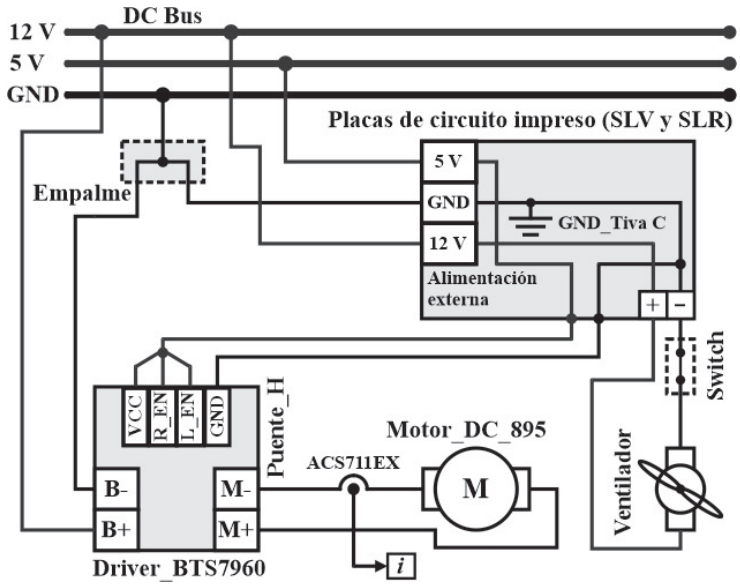

Figura 5.2: Circuito eléctrico del SLV y del SLR.

El actuador de cada subsistema es un motor DC modelo 895, con potencia de 360 W y alimentación de 12 V. Al ser un motor de alta potencia, su consumo de corriente eléctrica es elevado. El rango de medición del sensor de corriente analógico (ACS711EX) es de $\pm$ 15.5 A y su sensibilidad

de 90 *m*V/A. En el caso del puente H BTS7960 (véase 5.2), soporta un consumo de corriente de 43 A y voltajes de entrada de 6 a 27 V. Su capacidad de frecuencia PWM alcanza los 25 KHz y sus niveles de entrada de control son de 3.3 y 5 V.

Por otro lado, la fuente de alimentación es de 12 V, 600 W y 50 A. Las características de la fuente están basadas en el consumo de los motores. En la práctica, el funcionamiento simultáneo de ambos no supera los 31 A. Los niveles de voltaje requeridos en el sistema son de 5 y 12 V, las adaptaciones de la fuente son para facilitar su conexión. En la Figura 5.3 se observa el trabajo realizado en la fuente de alimentación. La placa de circuito impreso del regulador, el acoplamiento de las salidas y la caja del interruptor, están montados en una pieza fabricada con impresión 3D.

Figura 5.3: Adaptaciones en la fuente de alimentación conmutada.

5.2. Lógica programable y datos

El controlador de cada subsistema se integra por una *LaunchPad* de *Texas Instruments*. Esta tarjeta es la Tiva C TM4C123G, que cuenta con una alta velocidad de procesamiento (80 MHz) y con los periféricos necesarios para desarrollar el sistema steer-by-wire. Los módulos que se utilizan son: CAN, UART, QEI (encoder en cuadratura), PWM y ADC. A continuación, se describe la interacción de los controladores con los elementos electrónicos y de potencia. Además, de la interconexión de los subsistemas con los módulos de comunicación CAN.

5.2.1. Controlador del SLV

El diagrama electrónico y de control que se observa en la Figura 5.4, representa el funcionamiento del SLV. Algunas de las tareas principales del controlador son: la lectura, el procesamiento y la

transmisión de la posición angular del volante. El encoder absoluto se basa en la comunicación RS-485. La tarjeta de desarrollo no cuenta con el periférico para este protocolo. Por lo tanto, se utiliza un módulo convertidor bidireccional (MAX3485). Este driver permite que el controlador obtenga la información del encoder desde un módulo UART. La posición angular medida (*Pos_SLV*) es un valor de 15 bits (0 a 32767), que se debe adecuar al mecanismo. El proceso de Adecuación$_1$, divide a la señal *Pos_SLV* en 2 bytes para ser transmitida por CAN Bus al SLR. También escala y centra la lectura para tener ángulo de giro positivo y negativo (Pos_{SLV}). La señal Pos_{SLV} se utiliza en la función de retorno a cero del volante. El nuevo intervalo de valores para la posición medida es de -5000 (-720°) a 5000 (720°).

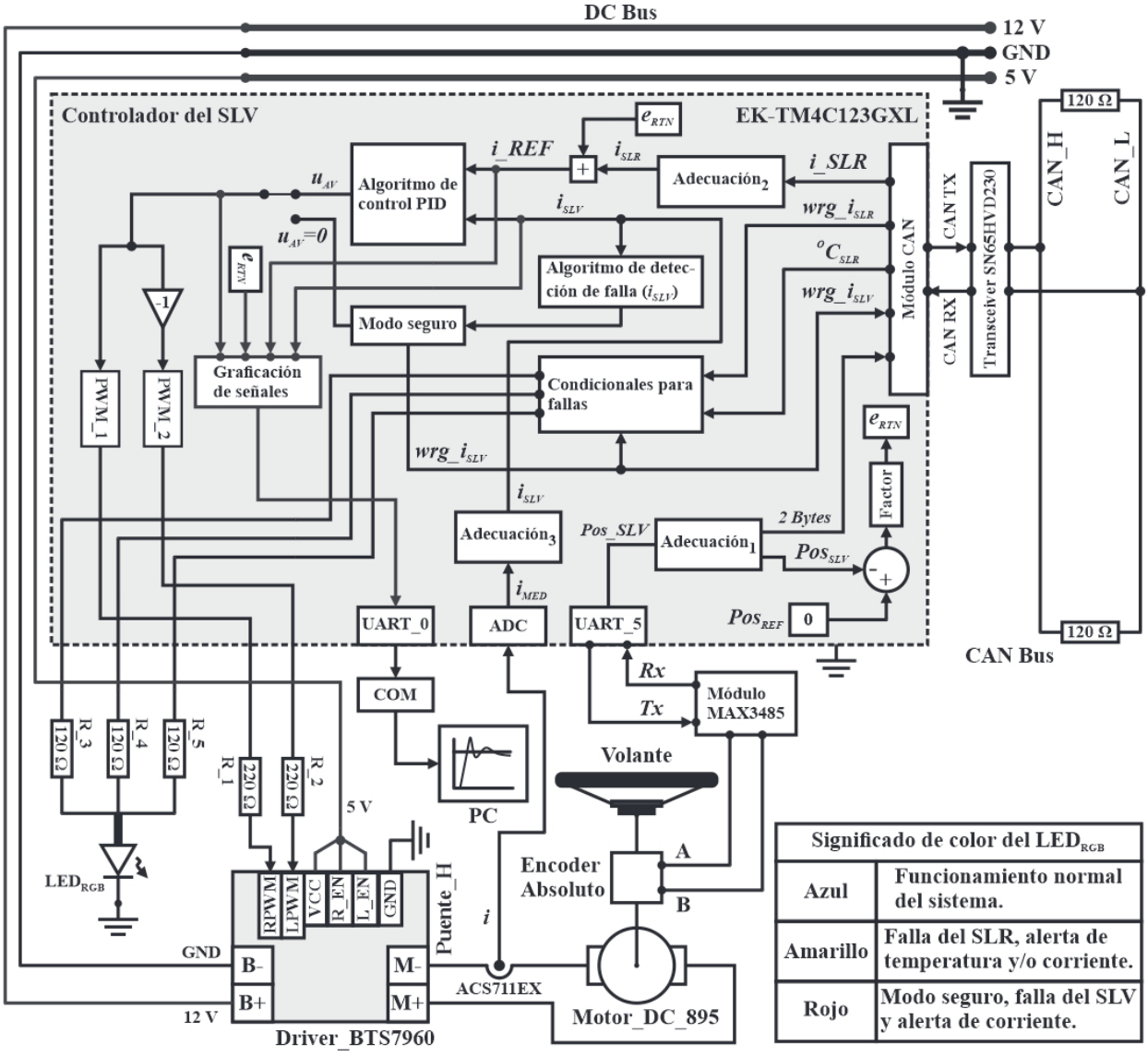

Figura 5.4: Circuito electrónico y de control para el subsistema lado volante.

El parámetro *i_SLR* contiene la corriente eléctrica consumida por el motor del subsistema lado

rueda. Los datos son recibidos en el módulo CAN como 2 bytes. En la Adecuación$_2$, los valores recibidos de corriente son armados y escalados. La señal resultante i_{SLR}, se utiliza en el esquema de control PID. En el mismo control se incluye la función de retorno a cero, donde se requiere el valor constante Pos_{REF} y la señal Pos_{SLV}. Su diferencia es multiplicada por un factor, para obtener el error de retorno (e_{RTN}). Esta compensación es añadida a i_{SLR} para formar la señal de referencia general i_REF.

La señal analógica i es proporcional a la corriente eléctrica consumida por el motor del SLV. La información de la señal es interpretada mediante un módulo ADC de 12 bits (0 a 4095). El sensor de corriente ACS711EX es bipolar, es decir, el valor 0 representa a -15.5 A y 4095 a +15.5 A. En la Adecuación$_3$ se cambia el intervalo de valores de la corriente medida (i_{MED}), para originar la señal de entrada i_{SLV} del control PID. El rango de esta señal es de -2047 (-15.5 A) a 2048 (+15.5 A).

La señal de salida u_{AV} del control de corriente, representa la realimentación de par en un formato de ciclo de trabajo ($duty\ \ cycle$). Al existir errores negativos, u_{AV} también es negativa. Para este caso, la señal de salida se debe multiplicar por -1. Así el controlador suministra una conmutación PWM unipolar al puente H en sus terminales RPWM y LPWM.

El algoritmo de detección de fallas analiza las lecturas de corriente del SLV (i_{SLV}). Este mecanismo determina la superación de umbrales de funcionamiento, excluyendo los picos de corriente eléctrica. En caso de que se superen (14 A $< i_{SLV} <$ -14 A), el modo seguro se activa. Donde la realimentación de par y la función de retorno se deshabilitan, llevando a cero la salida del control ($u_{AV} = 0$). A su vez, el modo seguro genera una alerta de consumo de corriente (wrg_i_{SLV}) que se transmite por CAN Bus al SLR. En el módulo CAN se reciben otros parámetros: la alerta de consumo de corriente del SLR (wrg_i_{SLR}) y la temperatura de su actuador ($^{\circ}C_{SLR}$). Ambos datos son procesados en el bloque de condicionales junto con la bandera de alerta del SLV (wrg_i_{SLV}). Este bloque consta de un conjunto de condiciones para determinar los alertas visuales de fallas. El Led_{RGB} muestra diferentes colores de acuerdo al estado de funcionamiento del sistema. El significado de cada color se presenta en la tabla inferior derecha de la Figura 5.4.

El módulo UART_0 se utiliza para mostrar las señales importantes en el análisis del subsistema. La información viaja por el puerto serie, hasta una interfaz gráfica desarrollada con un script de MatLab.

5.2.2. Controlador del SLR

El diagrama electrónico y de control del subsistema lado rueda, se observa en la Figura 5.5. La importancia del seguimiento de posición en este subsistema, ocasiona que el controlador inicie con la recepción de mensajes CAN. En estos mensajes se obtiene la posición angular del volante

(*Pos_SLV*) y la alerta de corriente del SLV (wrg_i_{SLV}). La información de la posición angular se recibe en 2 bytes. Por lo tanto, la Adecuación$_1$ se encarga de armar los datos y de escalarlos. El objetivo del escalamiento es modificar la sensibilidad del encoder absoluto y tener un sentido de giro. La señal resultante Pos_{SLV} es la referencia para el esquema de control del SLR. Su rango de valores es de $\pm$ 5000.

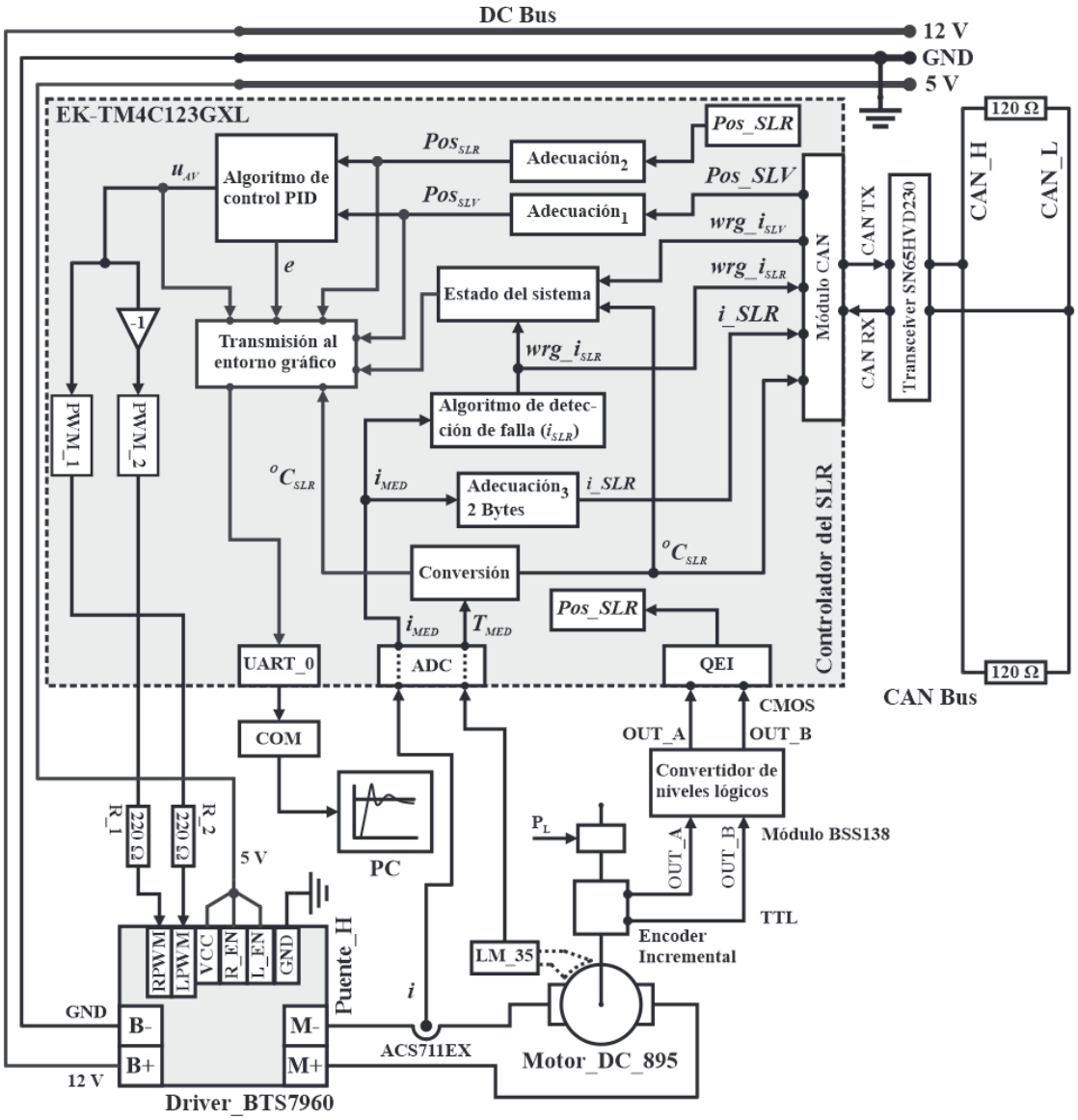

Figura 5.5: Circuito electrónico y de control para el subsistema lado rueda.

Por otro lado, la señal de entrada Pos_{SLR} del control se obtiene del encoder incremental. Este sensor de posición se encarga de enviar dos señales (OUT_A y OUT_B) al módulo QEI del controlador. El módulo hace una lectura en cuadratura de las fases A y B, para obtener la posición angular local (*Pos_SLR*). Las dos señales del sensor no ingresan directamente a la tarjeta de

desarrollo, se necesita una adecuación de niveles de voltaje (módulo BSS138). En el bloque de Adecuación$_2$, se modifica la resolución del encoder y si es necesario, se invierte el signo del valor de posición. El sentido de giro del mecanismo del SLR debe coincidir con el sentido de giro del volante.

Con las señales de referencia (Pos_{SLV}) y de posición medida (Pos_{SLR}), el controlador realiza el seguimiento de posición angular. La señal de salida u_{AV} del control, es manejada como un formato de ciclo de trabajo. Esta señal es multiplicada por -1, si sus valores son negativos. En seguida cada señal PWM (PWM_1 y PWM_2) es suministrada al puente H, dependiendo del sentido de giro requerido en el actuador.

El motor del SLR está sometido a cargas externas (P_L) para simular la realimentación de par. Entre mayor sea la carga aplicada, mayor es la corriente consumida por el actuador. La señal de corriente i del sensor ACS711EX, se ingresa en el módulo ADC. De este módulo se obtiene una señal de 12 bits (i_{MED}), que se debe procesar en el bloque de Adecuación$_3$ y en el Algoritmo de detección de fallas. En el primer bloque, los datos de la señal se dividen en 2 bytes (i_SLR) para poder transmitirse en el CAN Bus. Mientras tanto, en la detección de fallas la corriente medida se compara con dos umbrales establecidos. El algoritmo al determinar la superación de los umbrales (14 A $< i_{MED} <$ -14 A), genera una alerta de consumo de corriente (wrg_i_{SLR}). Esta bandera se integra en los mensajes CAN que son enviados al subsistema lado volante.

La temperatura del actuador es monitoreada con un sensor LM35. Su señal de salida se ingresa en el módulo ADC. El bloque de conversión utiliza la sensibilidad del sensor (10 mV/°C) para convertir los valores de la señal T_{MED} a grados Celsius ($^\circ C_{SLR}$). La temperatura es transmitida por el módulo CAN e ingresada al bloque de Estado del sistema. Este bloque genera las alertas mostradas en la interfaz gráfica. El entorno es desarrollado en App Designer de MatLab y recibe los datos de forma serial.

5.3. Instrumentación

Las Figuras 5.4 y 5.5 presentan la interconexión de los subsistemas mediante CAN Bus. Las tarjetas de desarrollo al trabajar con niveles CMOS, necesitan de un transceiver para sus módulos CAN. Este dispositivo convierte los niveles de voltaje del microcontrolador a niveles del estándar ISO 11898-2, en la capa física del protocolo. El transceiver utilizado es el SN65HVD230, compatible con microcontroladores de tecnología CMOS.

La lectura del encoder absoluto se realiza mediante una conversión de comunicación RS-485 y serial. El dispositivo convertidor es un módulo MAX3485. El modo de comunicación que se establece entre el controlador y el sensor es half-duplex. En el diagrama electrónico de la Figura

5.6, se observa que el driver se conecta a dos salidas digitales. El estado HIGH en las terminales RE y DE, permite la transmisión de tramas del controlador. Mientras tanto, el estado LOW determina la recepción de tramas en el módulo UART, provenientes del sensor de posición.

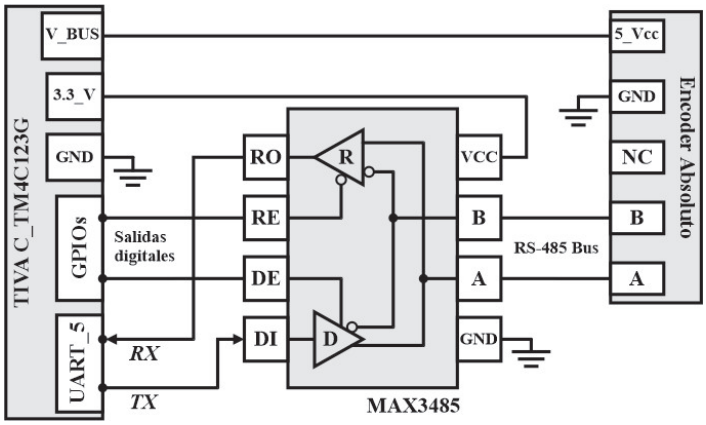

Figura 5.6: Circuito electrónico para la lectura del encoder absoluto.

Por otro lado, la lectura del encoder incremental requiere de un convertidor de niveles lógicos. Este dispositivo cambia las señales TTL del sensor a niveles CMOS. El módulo que se utiliza es el BSS138, integrado por canales bidireccionales de alta velocidad.

5.4. Integración de los módulos de electrónica

Los módulos de electrónica descritos anteriormente, son integrados formando una unidad central para cada subsistema. Estas unidades cuentan con terminales de conexión para potencia y sensores. Sus placas de circuito impreso son diseñadas con el software de EAGLE. Los elementos electrónicos son ubicados en las PCBs, de forma que se facilite su conexión. Especialmente, las tarjetas de desarrollo que se someten a constantes maniobras. A continuación, se presentan los circuitos implementados en las placas y los resultados de su fabricación.

5.4.1. Unidad central del SLV

El diagrama electrónico para la placa de circuito impreso del SLV, se observa en la Figura 5.7. Mientras tanto, en la Figura 5.8 se presenta la materialización de la placa e integración de los elementos electrónicos del subsistema. Las dimensiones de la PCB resultante son: 91 *mm* de largo y 84.5 *mm* de ancho.

Las borneras y terminales forman la periferia de la unidad central. Estos conectores son para las señales de entrada, salida y alimentación. Las conexiones mostradas en la Figura 5.8 corresponden

a: comunicación con el subsistema lado rueda (CAN Bus), alimentación externa (fuente conmutada), medición de corriente eléctrica y de posición angular del volante, control del driver de puente H, activación de la alerta visual del sistema y alimentación del actuador de enfriamiento.

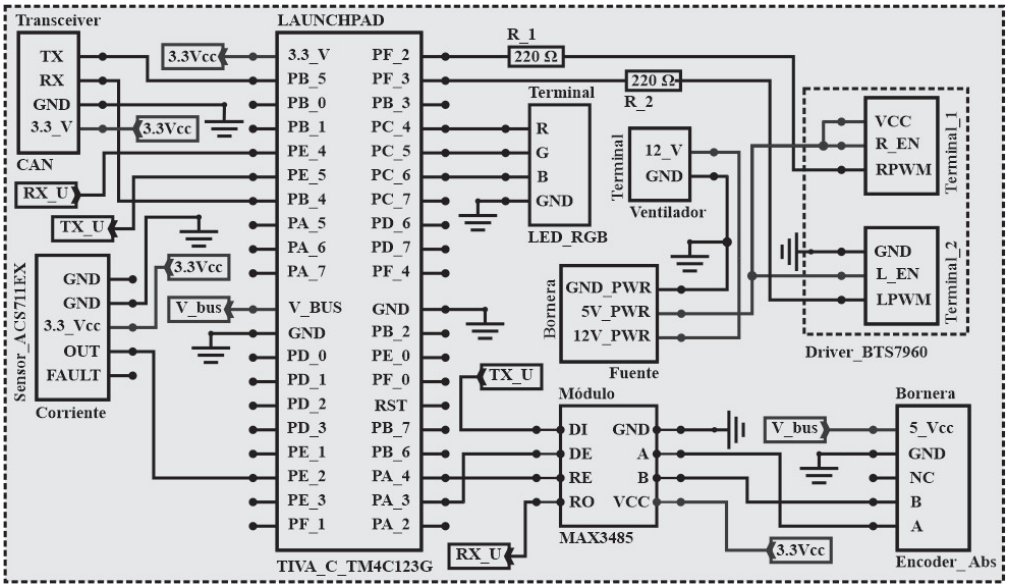

Figura 5.7: Circuito electrónico para la unidad central del SLV.

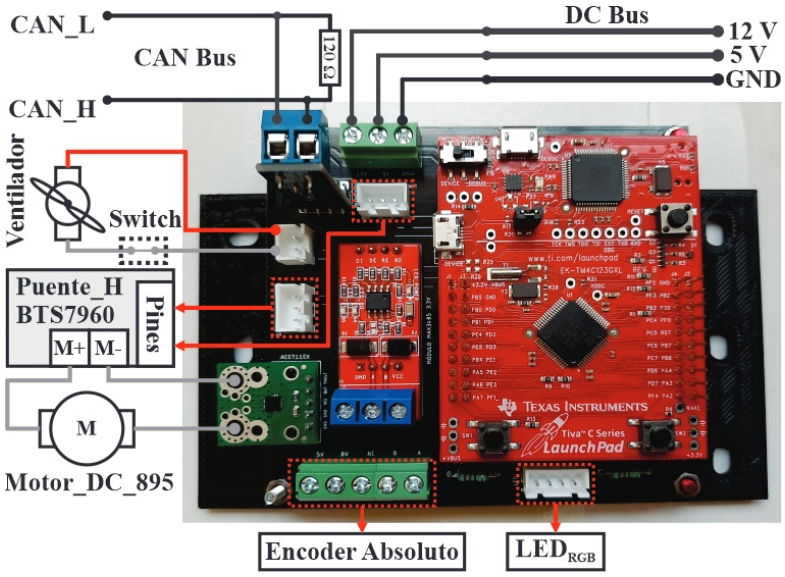

Figura 5.8: Periféricos y conexiones de la unidad central del SLV.

5.4.2. Unidad central del SLR

El diagrama electrónico de la placa de circuito impreso del SLR, se observa en la Figura 5.9. En la Figura 5.10 se presenta la PCB desarrollada con los elementos electrónicos del subistema

lado rueda. Esta unidad central es similar a la anterior, excepto por el sensor de temperatura y el convertidor de niveles lógicos. También en la Figura 5.10 se indican los periféricos y las conexiones correspondientes. La PCB se coloca en una base fabricada con impresión 3D. Este soporte es diseñado de acuerdo con las dimensiones de la placa de circuito impreso (85 *mm* x 88.5 *mm*).

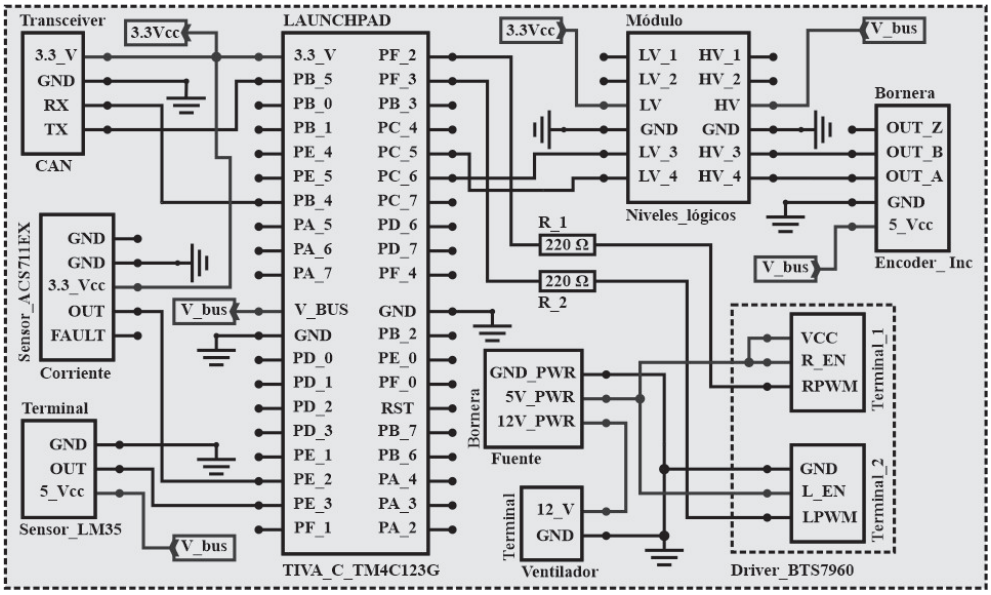

Figura 5.9: Circuito electrónico para la unidad central del SLR.

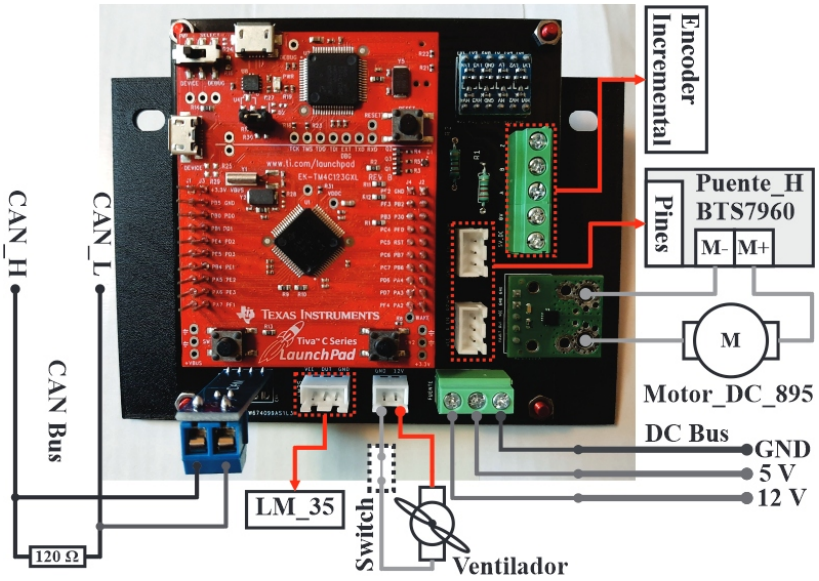

Figura 5.10: Periféricos y conexiones de la unidad central del SLR.

Capítulo 6

Desarrollo de lógica de control

En este capítulo se describe el desarrollo de los bloques lógicos de la dirección steer-by-wire. Se presenta la simulación del sistema, con el software Simulink de MatLab. También se incluye la descripción de la lógica de control, implementada en los códigos del subsistema lado volante (SLV) y del subsistema lado rueda (SLR). Estos códigos son desarrollados en lenguaje de programación C, en Code Composer Studio.

6.1. Simulación del sistema steer-by-wire

La dirección steer-by-wire es un sistema interconectado de dos motores DC. El motor acoplado al volante se controla para tener una realimentación de par. La realimentación está en función de la corriente consumida por el otro actuador. Mientras tanto, el motor del SLR se controla para igualar la posición angular del volante. Es posible simular este sistema con el modelo matemático de motores DC [45]:

$$L\frac{\mathrm{d}i_a}{\mathrm{d}t} = -K_e\omega - R_a i_a + V_t \tag{6.1}$$

$$J\frac{\mathrm{d}\omega}{\mathrm{d}t} = K_m i_a - B\omega - T_L \tag{6.2}$$

donde la variable i_a es la corriente de armadura, R_a la resistencia de armadura, L la inductancia, J la inercia, T_L la carga aplicada, B el coeficiente de fricción viscosa, K_e y K_m son constantes de la máquina (eléctrica y mecánica), y V_t el voltaje aplicado. El término $K_m i_a$ es el par generado por el motor y el término $K_e\omega$ es la fuerza contraelectromotriz. La ec. 6.1 representa el modelado eléctrico y la ec. 6.2 el modelado mecánico. También se tiene que:

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = \omega \tag{6.3}$$

donde ω es la velocidad angular y θ es la posición angular del motor DC. Estas ecuaciones son pro-

gramadas en el entorno de Simulink. A continuación se describen los bloques lógicos desarrollados en la simulación del sistema steer-by-wire.

6.1.1. Programación del subsistema lado volante

El SLV está integrado por dos bloques lógicos que son: control PID de corriente eléctrica y modelo matemático del motor DC (véase Figura 6.1). En el bloque de control se requiere una corriente de referencia (Ref_ia) y la corriente eléctrica consumida (ia_SLV). Estas señales se usan para obtener el voltaje (Uav_SLV) que se suministra al modelo de la máquina. La alimentación del motor se limita a 12 V. En este modelo también se ingresa una carga (TL), que representa el par que aplica el conductor al volante.

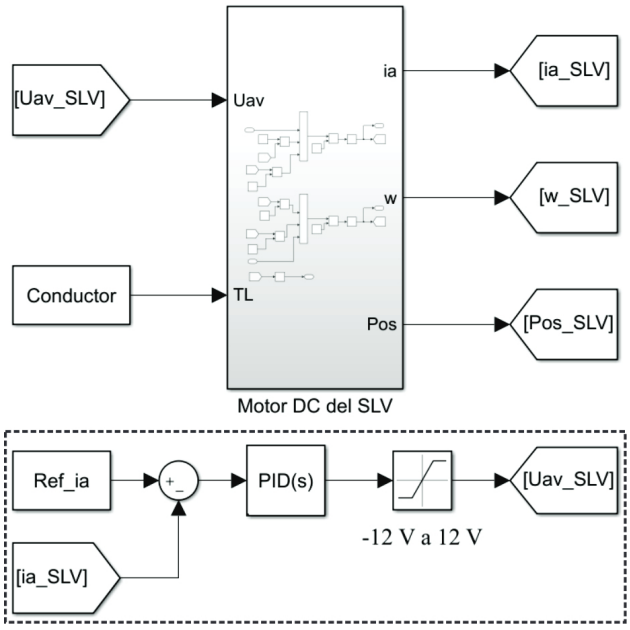

Figura 6.1: Bloques lógicos del subsistema lado volante.

En el bloque lógico del motor se encuentra la ecuación eléctrica y la ecuación mecánica. Al ser resueltas se obtiene la corriente consumida (ia_SLV) y la velocidad angular (w_SLV). Además, integrando esta velocidad se obtiene la posición angular del actuador (Pos_SLV).

6.1.2. Programación del subsistema lado rueda

En la Figura 6.2 se observa la interconexión de los bloques lógicos del SLR. Esta simulación es una réplica del subsistema anterior. Sin embargo, el control PID realiza un seguimiento de posición angular. La señal que se suministra en el control, es la diferencia entre posición de referencia

(Ref_Pos) y la posición medida en el motor (Pos_SLR). La señal de salida para el voltaje de alimentación (Uav_SLR), también se limita a 12 V. El par que se ingresa al modelo representa una perturbación del camino. Su activación es en determinado tiempo de la simulación.

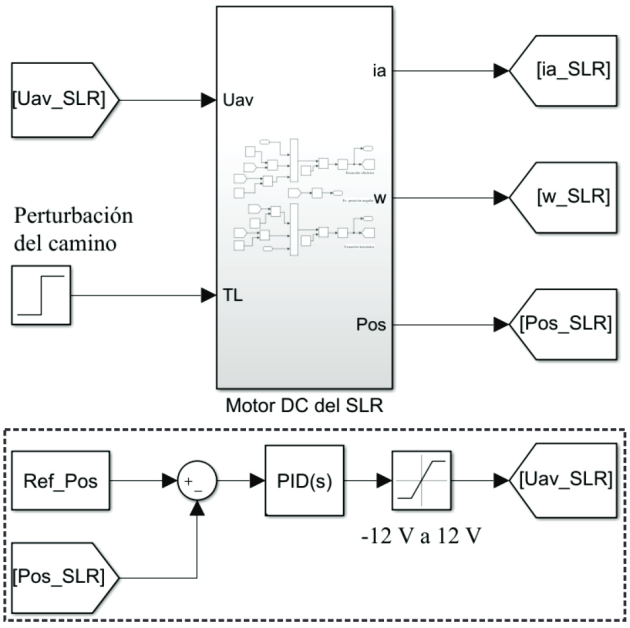

Figura 6.2: Bloques lógicos del subsistema lado rueda.

6.1.3. Simulación de los subsistemas interconectados

La interconexión de los subsistemas se observa en la Figura 6.3. La posición de referencia para el SLR, es la obtenida en el modelo del motor del lado volante (Pos_SLV). Mientras tanto, en el SLV se usa como referencia, la corriente consumida por el actuador del lado rueda (ia_SLR). Esta señal de corriente eléctrica, no se ingresa directamente al control del lado volante. Existe un ajuste de par que regula la sensación de conducción.

La simulación del sistema steer-by-wire, también incluye la función de retorno a cero del volante. En esta función se requiere la diferencia entre la posición 0 y la posición medida en el subsistema (Pos_SLV). El error es multiplicado por una ganancia, que determina la rapidez en que el volante regresa a la posición central. En el seguimiento de corriente se tiene una referencia general (ia_REF), que resulta al añadir la compensación del retorno a 0.

Los bloques del modelo de motor DC incluyen una entrada para la posición inicial del volante y de las ruedas. Los valores iniciales que se ingresan, se muestran en la Figura 6.3. La configuración presentada es para simular el control de posición, el control de corriente y la función de retorno. El par que aplica el conductor, puede ser representado con una señal sinusoidal. Además, cuando se realiza el retorno a 0, el conductor no aplica par al volante y se anula la perturbación del camino.

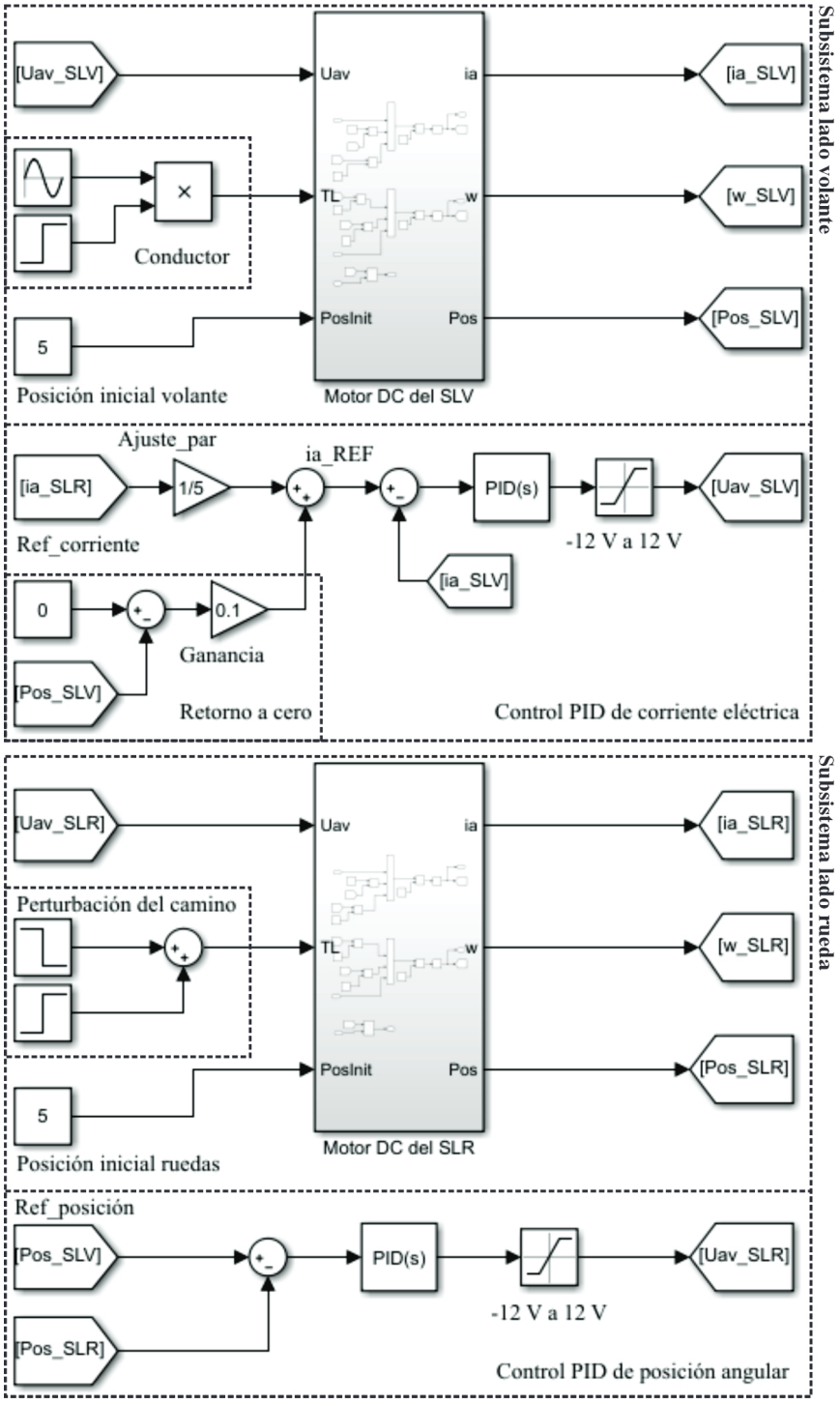

Figura 6.3: Simulación del sistema steer-by-wire en Simulink de MatLab.

Los resultados de la simulación, se observan en las Figuras 6.4 y 6.5. En las gráficas se presentan tres casos que son: retorno a posición cero, giro libre y perturbación del camino, respectivamente. En la Figura 6.4 a) y b) se observa cómo la posición de referencia (Pos_SLV) pasa de 5 a 0 radianes y es seguida adecuadamente por la posición medida del motor lado rueda. En seguida, se habilita

el movimiento en un sentido y otro del volante con una función sinusoidal. En este caso también la posición medida en el motor del lado rueda (Pos_SLR) sigue correctamente la señal de referencia (Pos_SLV). Incluso con la perturbación del camino, que sucede entre los 12 y 14 segundos de la simulación (véase Figura 6.4 b)). La perturbación reduce el desplazamiento angular del sistema.

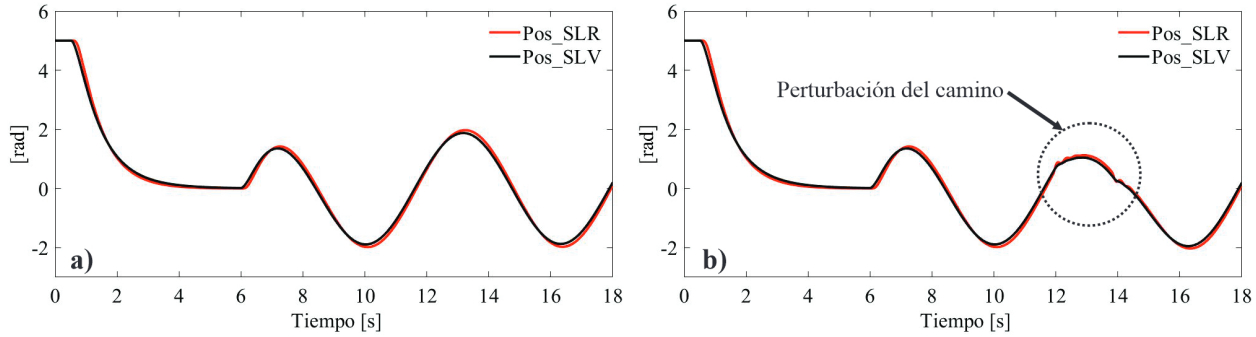

Figura 6.4: Comparación de resultados del control de posición angular en la simulación a) sin cargas aplicadas y b) con perturbación del camino.

En el control PID de corriente también se logran resultados satisfactorios. En la Figura 6.5 a) se observa que la corriente consumida por el motor del SLV (ia_SLV) iguala a la señal de referencia general (ia_REF). La perturbación del camino, se refleja como un aumento en la corriente del motor del lado rueda (ia_SLR). Este aumento influye en el resto de las señales de corriente. Por otro lado, en la Figura 6.5 b) se visualizan las señales de control que reflejan las maniobras de compensación, tanto de corriente (proporcional al par aplicado) y de posición angular.

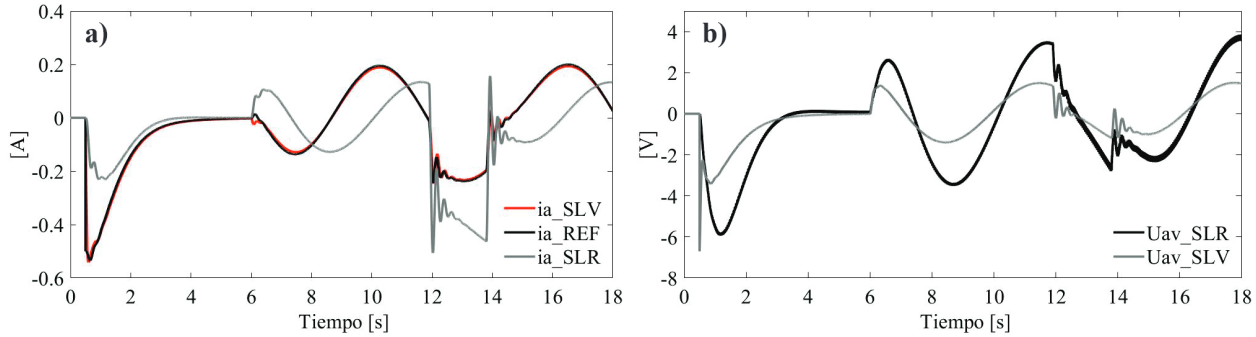

Figura 6.5: Resultados de la simulación con perturbación aplicada a) control de corriente eléctrica y b) señales de salida de los bloques de control PID.

6.2. Programación del sistema steer-by-wire

Después de verificar el desempeño del sistema de control propuesto y corroborar la validez de los resultados. Se procede a implementar en el microcontrolador los bloques lógicos arriba descri-

tos. En el software del sistema se utiliza el kit de desarrollo de TivaWare. Este complemento es para el manejo de los periféricos y de los recursos internos de la tarjeta TM4C123G. Los códigos desarrollados para ambos subsistemas se encuentran en el Apéndice A.

Cabe señalar que las condiciones establecidas para la detección de posibles fallas en el sistema, se establecen de manera arbitraria con el objetivo de probar el algoritmo de detección. Por lo tanto, se establece una corriente máxima de consumo en los motores de 14 A a pesar de que los motores pueden consumir hasta 30 A a velocidad nominal. Por otro lado se establece una temperatura de falla de 40 $^{\circ}C$ a pesar de que la temperatura máxima nominal en los motores es de 80 $^{\circ}C$. A continuación, se describe la lógica de los programas, la configuración de módulos, los controles PID, las mediciones de sensores, así como las lecturas y escrituras del CAN Bus.

6.2.1. Lógica de control del SLV

La estructura y la lógica del programa para el subsistema lado volante, se observa en la Figura 6.6. En la parte inicial se incluyen las librerías (TivaWare) para periféricos, timers e interrupciones. Además, se declaran las variables y los prototipos de funciones. Mientras tanto, en el $setup()$ se establece la velocidad de procesamiento del microcontrolador (50 MHz) y la activación de sus módulos CAN, UART, GPIO, ADC y PWM.

La medición del tiempo es necesario en el control PID y en el algoritmo de detección de falla (i_{SLV}). Para medirlo se utiliza un timer y una interrupción periódica. Cada milisegundo se activa una rutina de interrupción, donde una variable denominada $millis$ aumenta en 1. También se configura un timer como disparador para las lecturas del módulo ADC. En cada ciclo de muestreo ocurre una interrupción para obtener las muestras de este módulo. La frecuencia de muestreo establecida es de 40 KHz.

En la activación del módulo PWM, se establece un bloque generador y dos señales de salida. La frecuencia que se utiliza es de 10 KHz, originando un contador de 0 a 5000. El modo de conteo que se configura para ajustar el ancho de pulso es ascendente-descendente.

Por otro lado, la velocidad de transmisión del módulo CAN se configura a 1 Mbps. En el $setup()$ se establecen 2 objetos de mensaje, uno para transmisión y el otro para recepción. Los parámetros que se configuran en los objetos de mensaje son: identificador de mensaje, máscara del identificador, indicadores de estado (transmisión, recepción y filtrado), cantidad de bytes de datos y el puntero a los datos del mensaje. Este último parámetro solo es válido para el objeto de transmisión.

En la Figura 6.7 se observa la implementación de interrupciones en el módulo CAN. Estas interrupciones son activadas por el envío y recepción de mensajes. También debido a errores en el canal de comunicación. En la rutina se determina la causa y se actualizan las banderas de recepción

de mensajes ($rxFlag_{SLV}$) y de error del CAN Bus ($errFlag_{SLV}$).

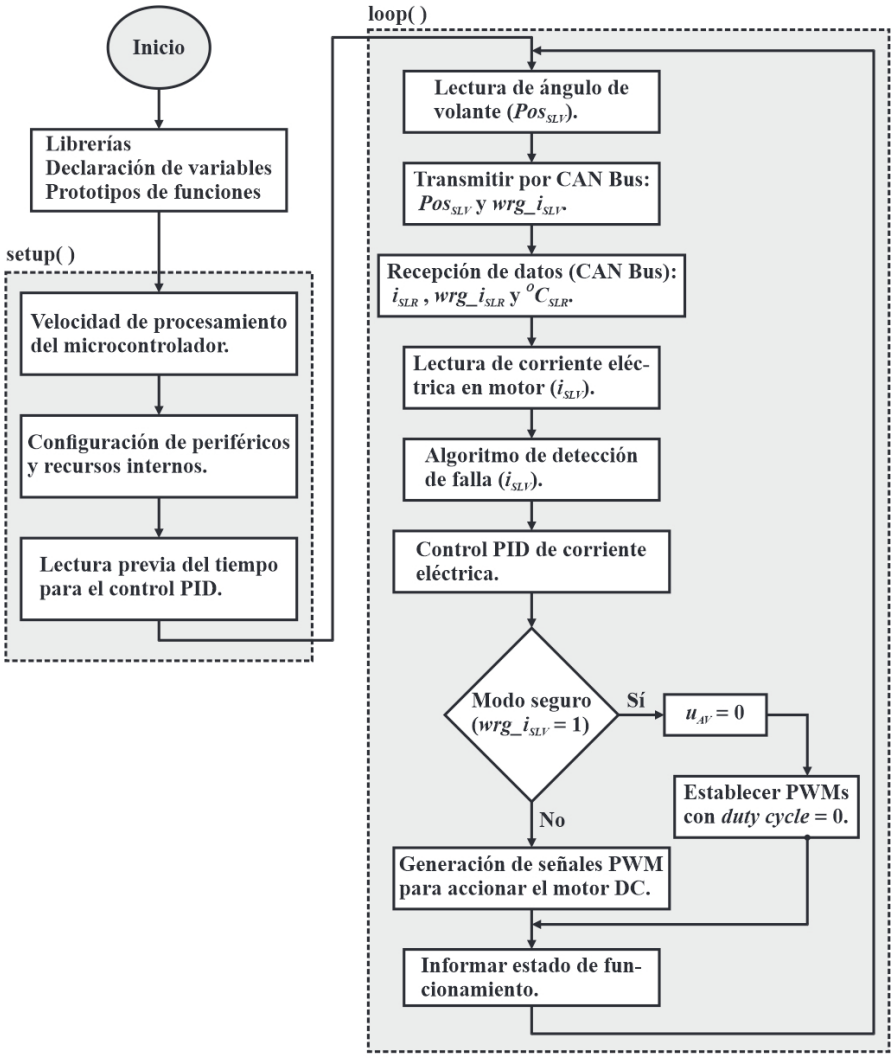

Figura 6.6: Estructura y lógica del código para el subsistema lado volante.

En seguida se configura el UART_0, este periférico se utiliza para enviar datos a la PC y así, visualizar las señales importantes del SLV. Se configura a 115200 bps, utilizando el oscilador interno de 16 MHz como fuente de reloj.

Por otro lado, en el módulo GPIO se configuran 5 salidas digitales, 3 salidas se utilizan para la activación del indicador luminoso (LED_{RGB}). Mientras tanto, las 2 salidas restantes habilitan los pines DE y RE del driver MAX3485. Los pines del driver se colocan en estado ALTO, cuando se solicita la posición angular al encoder absoluto. El estado en BAJO permite recibir las tramas de respuesta del sensor. Estas tramas son enviadas y recibidas mediante el módulo UART_5 del controlador. Este módulo UART se configura de acuerdo a los parámetros de comunicación del encoder. Se consideran 8 bits de datos, un bit de parada y ningún bit de paridad. Además, la tasa de

baudios se establece en 9600 bps.

Para solicitar la posición angular del volante, la trama que se envía al encoder absoluto es: `0x0001 0x0003 0x0000 0x0000 0x0000 0x0001 0x0084 0x000A`. La respuesta que se obtiene consiste en una trama de 7 bytes [0-6], donde la posición es almacenada en el byte [3] y [4]. Los datos son armados a través de un corrimiento de bits (desplazamiento a la izquierda). Considerando al byte [3] como el más significativo.

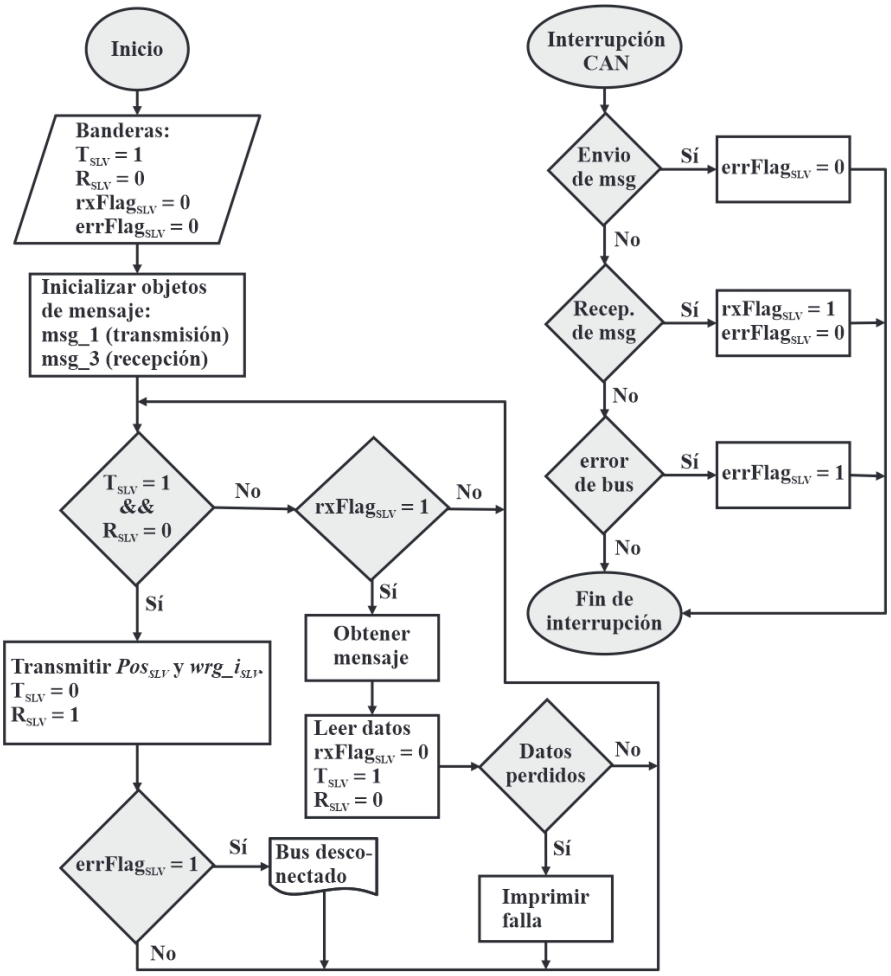

Figura 6.7: Diagrama de flujo para la comunicación CAN en el SLV.

La manera en que se envían y se reciben los datos en el CAN Bus, se observa en la Figura 6.7. Dada la importancia del control de posición, primero se realiza la escritura en el CAN Bus. Se transmite la posición angular del volante (Pos_{SLV}) y la bandera de alerta del SLV (wrg_i_{SLV}). En seguida, se actualizan las banderas de transmisión y recepción (T_{SLV} y R_{SLV}). Preparando la lectura del canal de comunicación. Cuando la bandera de recepción de mensajes está activada ($rxFlag_{SLV}$ = 1), se procede a obtener los datos del SLR. Estos datos son: corriente eléctrica consumida por el motor (i_{SLR}), alerta de consumo de corriente (wrg_i_{SLR}) y temperatura del actuador ($^{\circ}C_{SLR}$).

En el módulo ADC se obtiene la corriente consumida por el motor del subsistema local. Este parámetro (i_{SLV}) se utiliza en el control PID y en el algoritmo de detección de falla. El diagrama de flujo para el algoritmo, se observa en la Figura 6.8. Cuando se supera el umbral de funcionamiento establecido ($\pm$ 14 A), se determina el tiempo parcial (t_P) en que ocurrió el evento. Mientras la corriente supere el umbral, el tiempo transcurrido (t_T) se va actualizando. Si la diferencia de los tiempos supera los 2 segundos, la bandera de alerta wrg_i_{SLV} se coloca en estado ALTO y el modo seguro es activado. El tiempo umbral (t_U) se utiliza como mecanismo para descartar los picos de corriente eléctrica.

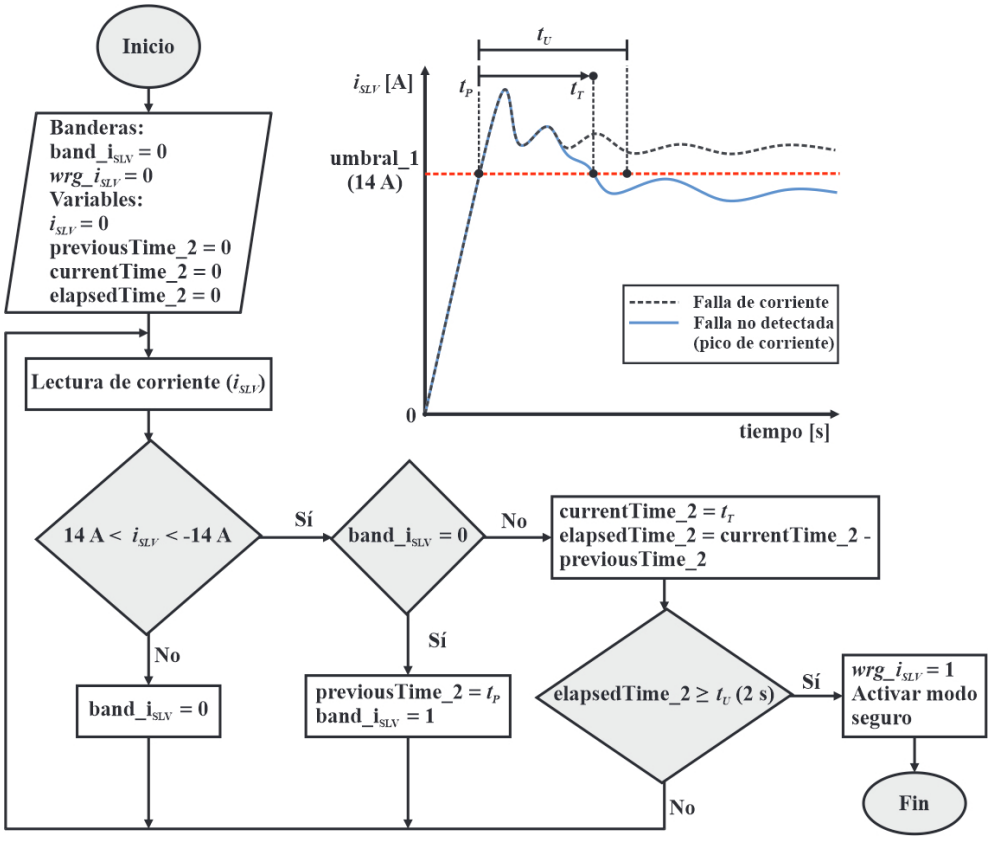

Figura 6.8: Diagrama de flujo para el algoritmo de detección de falla en el SLV.

El esquema de control para el SLV, se observa en la Figura 6.9. La función de retorno a cero proporciona un error de compensación (e_{RTN}), que añadido a la corriente i_{SLR} origina la señal general de referencia (i_REF). El error entre i_REF e i_{SLV}, es utilizado por el control PID. Las ganancias del controlador son K_p (proporcional), K_i (integral) y K_d (derivativa). Además, se ajusta a una frecuencia de 200 Hz.

En el esquema de control también se incluye un saturador. Sus límites están en función del periodo de salida para el generador PWM. Después del saturador, la señal u_{AV} es sometida a la condición del modo seguro. Si el modo seguro está habilitado, el ciclo de trabajo para las señales

PWM es cero. Caso contrario, el motor es accionado según el sentido de giro requerido. Cuando u_{AV} consta de valores negativos, se multiplica por -1 para considerarla como ciclo de trabajo.

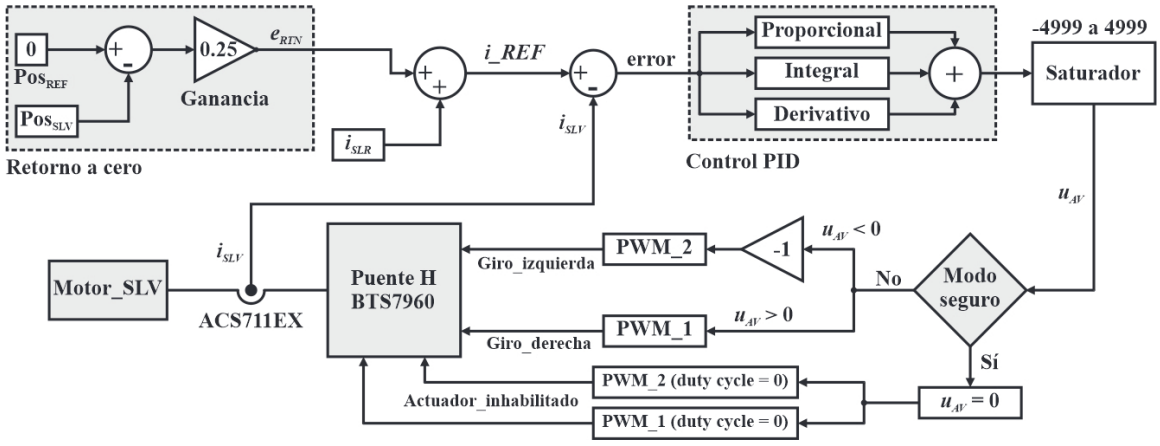

Figura 6.9: Esquema de control PID de corriente para el SLV.

La siguiente tarea en el ciclo *loop*(), consiste en informar el estado de funcionamiento del sistema. En esta rutina se activa el led indicador, el color que presenta depende de un conjunto de condicionales (véase Figura 6.10).

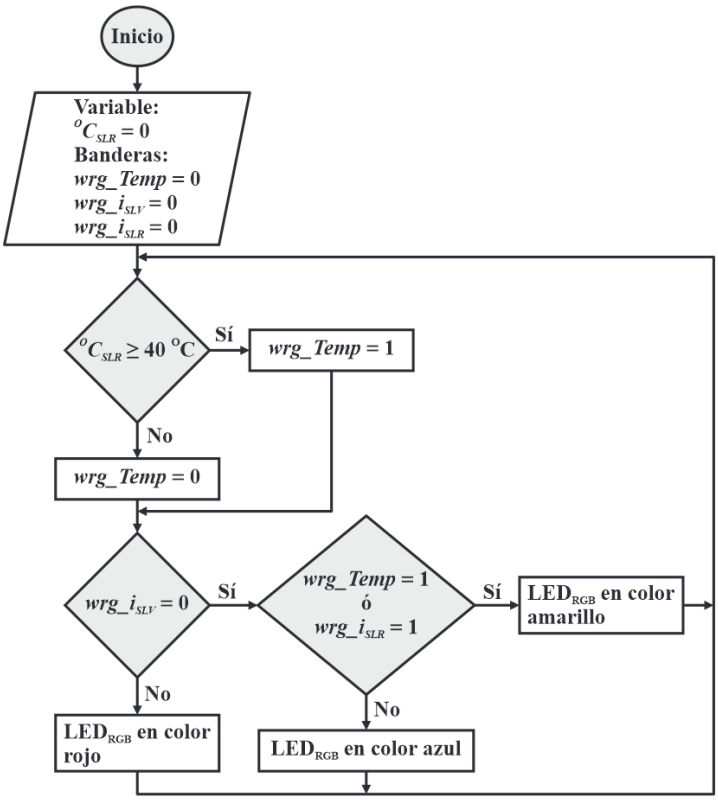

Figura 6.10: Diagrama de flujo para informar el estado de funcionamiento del sistema.

Mientras ninguna bandera de alerta esté en ALTO, el LED_{RGB} indica funcionamiento normal (color azul). En cambio, si la alerta de corriente del SLR (wrg_i_{SLR}) ó la de temperatura (wrg_Temp) se activa, el LED_{RGB} cambia a color amarillo (falla del SLR). El modo seguro se indica con el led en rojo y se debe a la alerta de corriente del SLV (wrg_i_{SLV}). La bandera wrg_Temp se actualiza, si la temperatura del motor del SLR ($^{\circ}C_{SLR}$) supera el umbral establecido (40 $^{\circ}C$).

6.2.2. Lógica de control del SLR

La estructura y la lógica del programa para el subsistema lado rueda, se observa en la Figura 6.11. De manera similar al SLV, en la parte inicial se incluyen las librerías, variables y prototipos de funciones. En el $setup()$ se configura la velocidad de procesamiento del microcontrolador (50 MHz) y se activan los mismos módulos del subsistema anterior. Excepto por los periféricos UART_5 y GPIO. Además, se configura el módulo QEI para la lectura del encoder incremental.

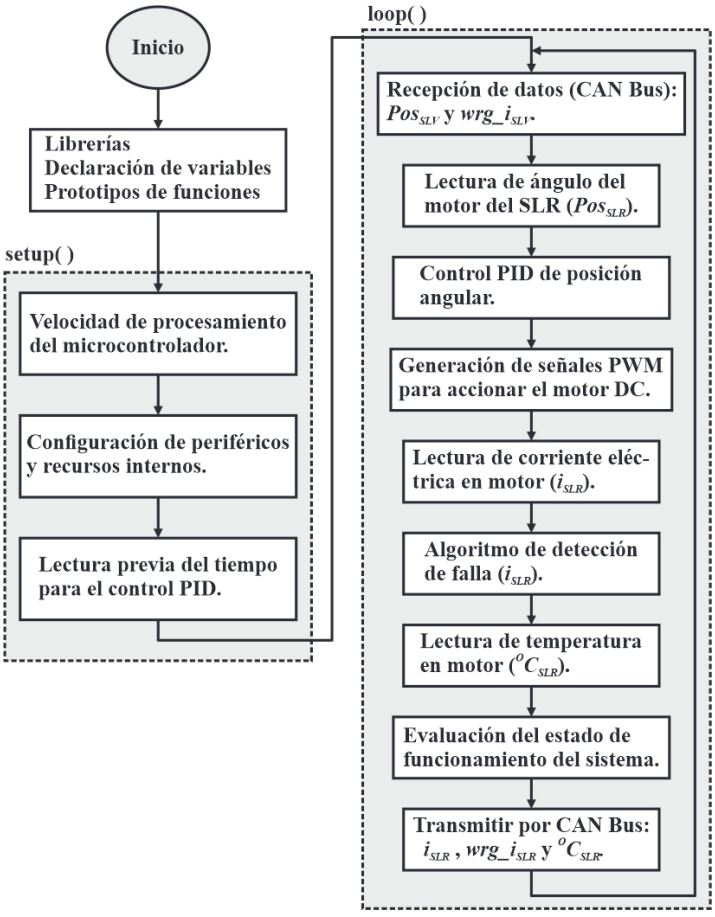

Figura 6.11: Estructura y lógica del código para el subsistema lado rueda.

Para medir el tiempo se sigue empleando un timer y una interrupción. Estos recursos internos

y los módulos: CAN, PWM, UART_0 y ADC, son configurados con los parámetros establecidos en el SLV (frecuencias y velocidades de transmisión). En el módulo PWM se considera el mismo bloque generador y las dos señales de salida. Por otro lado, en el módulo ADC se establecen 2 canales para la toma de muestras. Un canal se utiliza para el sensor de corriente (ACS711EX) y el otro para el sensor de temperatura (LM35).

En la configuración del módulo QEI se activan dos pines de entrada. Estos pines se encargan de detectar los pulsos de la fase A y B del encoder. En este módulo se establece el uso de los dos canales y su lectura en cuadratura.

En la Figura 6.12 se observa la comunicación CAN del subsistema lado rueda. En este subsistema también se establecen 2 objetos de mensaje, uno para recepción y el otro para transmisión. La bandera de error del CAN Bus ($errFlag_{SLR}$) y la de recepción de mensajes ($rxFlag_{SLR}$) se actualizan con el mecanismo de interrupciones CAN.

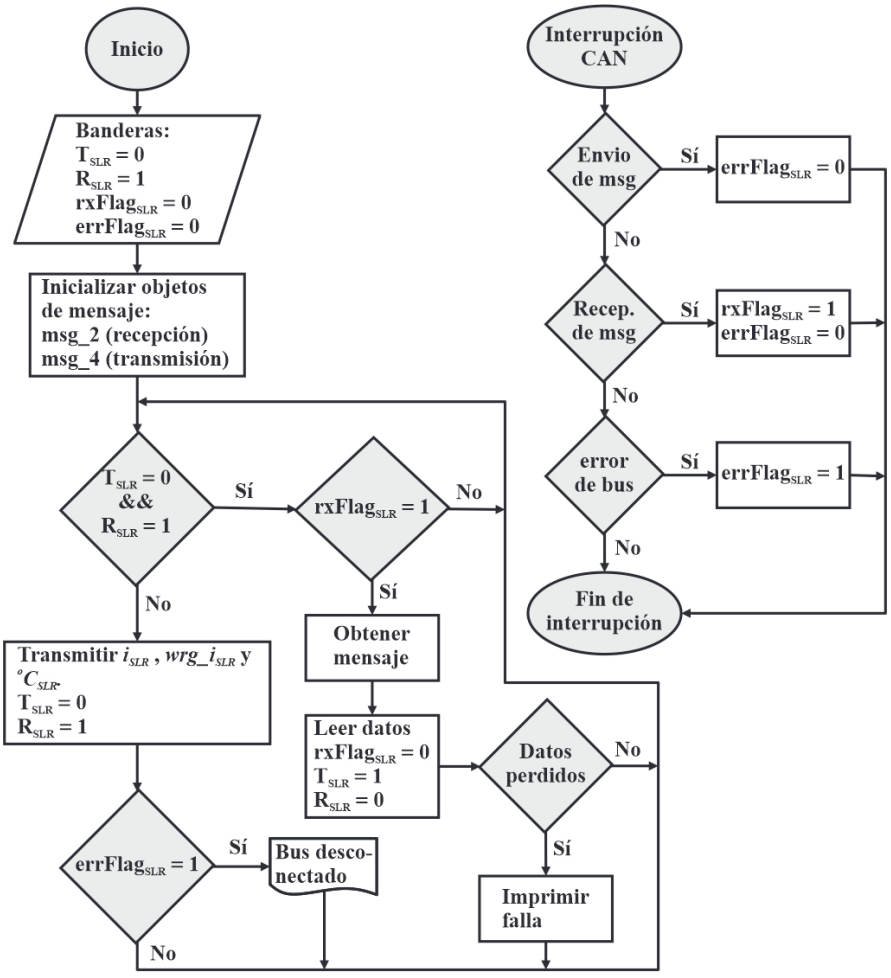

Figura 6.12: Diagrama de flujo para la comunicación CAN en el SLR.

Dada la prioridad del control de posición angular, se establece primero la lectura del canal de

comunicación. Con la bandera de recepción en ALTO, se procede a obtener el mensaje. Los datos recibidos son: el ángulo del volante (Pos_{SLV}) y la bandera de alerta del SLV (wrg_i_{SLV}). Después de esta lectura, se actualizan las banderas de recepción (R_{SLR}) y de transmisión (T_{SLR}). Preparando al módulo CAN para escribir en el bus. El envio de datos, no se realiza de forma inmediata. De acuerdo con la Figura 6.11, los datos son enviados al final de la rutina $loop()$.

Una vez conocido el ángulo del volante (Pos_{SLV}), se obtiene el ángulo del motor del SLR (Pos_{SLR}). Ambos parámetros son utilizados en el control PID, siendo Pos_{SLV} la señal de referencia (véase Figura 6.13). En el algoritmo de control, se emplea el error para generar una señal de salida u_{AV}. Con los valores de esta señal se obtienen dos salidas PWM, que accionan al motor en el sentido y ángulo requerido.

En seguida se realiza la medición de corriente en el motor (i_{SLR}). Los valores obtenidos, se transmiten por CAN Bus al subsistema lado volante. Además, se ingresan en el algoritmo de detección de falla. La lógica del algoritmo es la misma que la presentada en la Figura 6.8. Cuando la corriente eléctrica supera los umbrales establecidos ($\pm$ 14 A), se activa la bandera de alerta asignada ($wrg\_i_{SLR} = 1$).

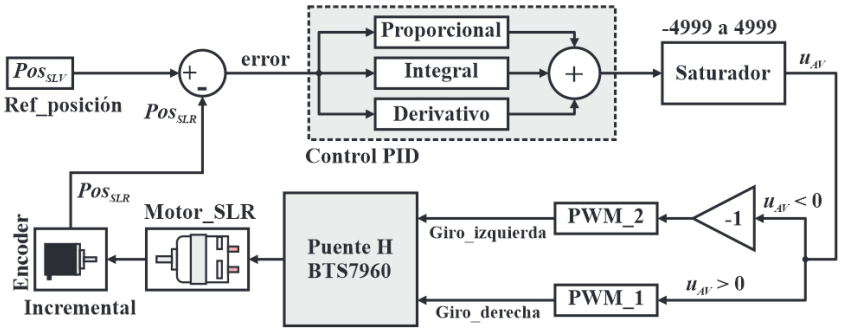

Figura 6.13: Esquema de control PID de posición angular para el SLR.

La otra función del módulo ADC, consiste en la lectura del sensor LM35. Este sensor monitorea la temperatura del motor del SLR ($°C_{SLR}$). Este parámetro se transmite en el CAN Bus, junto con la alerta de corriente wrg_i_{SLR}.

Finalmente, se tiene la evaluación del estado de funcionamiento. Esta rutina consiste en reunir todas las banderas de alerta de la dirección steer-by-wire. Clasificándolas y determinando las condiciones de los subsistemas. La información es enviada por puerto serie a la interfaz gráfica desarrollada en App Designer (del software MatLab).

Capítulo 7

Resultados y conclusiones

En este capítulo se analizan los resultados obtenidos del sistema steer-by-wire (véase Figura 7.1). En la plataforma experimental se realizan algunas pruebas que muestran el desempeño de los dos esquemas de control PID, descentralizados e interconectados mediante CAN Bus. Las pruebas de funcionamiento ocurren en condiciones normales y en ciertos escenarios de falla. En ambos casos se recopilan datos para evaluar el desempeño del prototipo de dirección desarrollado. Los escenarios de prueba establecidos para las condiciones normales son:

- Prueba de seguimiento sin perturbaciones de par.
- Carga aplicada en el subsistema lado rueda.
- Maniobras alrededor de la posición cero.

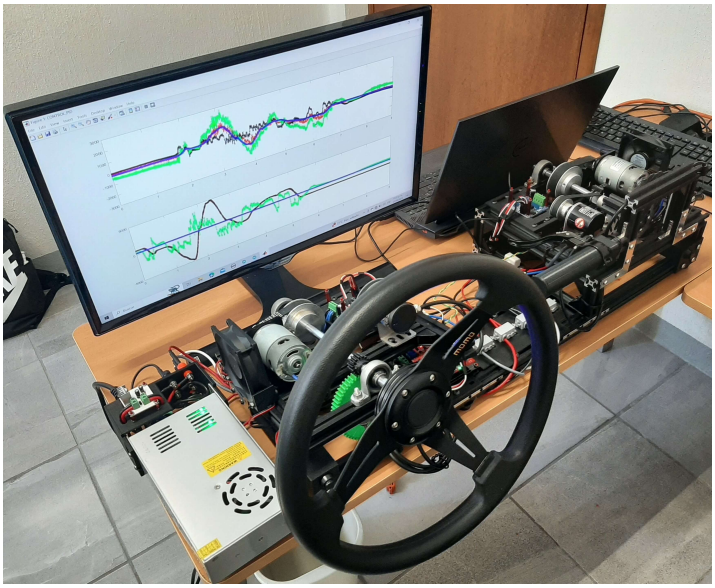

Figura 7.1: Obtención de resultados en el prototipo de dirección steer-by-wire, utilizando la herramienta de software MatLab.

Por otro lado, se tienen dos escenarios de falla: aumento de la temperatura del motor del lado rueda y superación de umbrales de corriente eléctrica en ambos motores. En los escenarios de falla

se debe detectar el alcance de umbrales establecidos y la entrada del modo seguro en el subsistema lado volante (SLV). Algunas características de las pruebas experimentales, son las siguientes:

- El aumento de la temperatura en el actuador del subsistema lado rueda (SLR), se simula acercando una fuente de calor al sensor LM35.
- Los umbrales de corriente se alcanzan rápidamente deteniendo por completo el motor del SLR, y girando el volante de dirección mientras se acciona el freno de disco del lado rueda.
- Se recopilan dos grupos de datos, uno por cada subistema. Al utilizar dos puertos seriales, la información no se recibe al mismo tiempo en la PC. Para presentar estos datos de forma correcta se requiere implementar un mecanismo de sincronización.

A continuación, se describen las pruebas experimentales y se presentan los resultados de forma gráfica. También se muestra la interfaz desarrollada en App Designer de MatLab, para despliegue de información del sistema. Finalmente se exponen las conclusiones y se proporcionan los trabajos futuros.

7.1. Escenarios de prueba en condiciones normales

Los siguientes escenarios de prueba tienen la intención de validar los esquemas de control propuestos en condiciones normales. Cabe mencionar, que el error de posición angular presentado, se mide en porcentaje con respecto a la cantidad de vueltas total del volante (4 vueltas).

7.1.1. Prueba de seguimiento sin perturbaciones de par

En esta prueba no se aplica par al motor del SLR. El volante se mueve en ambas direcciones hasta llegar al tope mecánico. Cuando el volante se encuentra fuera de la posición central y el usuario lo suelta se observa la función de retorno a la posición angular cero (posición central del volante, es decir, ruedas alineadas al frente en la práctica). En la Figura 7.2 a) se presenta la acción de retorno (e_{RTN}). La forma de la señal e_{RTN} coincide con el desplazamiento angular del volante (de manera invertida). Esta compensación se añade a la corriente de referencia del lado rueda (i_{SLR}) para formar la señal de referencia general (i_REF) del controlador. La corriente i_{SLR} alcanza valores entre $\pm$ 5 A en estas condiciones de experimentación.

En la Figura 7.2 b) se muestra la regulación de corriente en el SLV. La señal i_{SLV} es la corriente consumida en el motor del lado volante. La respuesta del control se considera adecuada, i_{SLV} sigue correctamente la señal de referencia general i_REF.

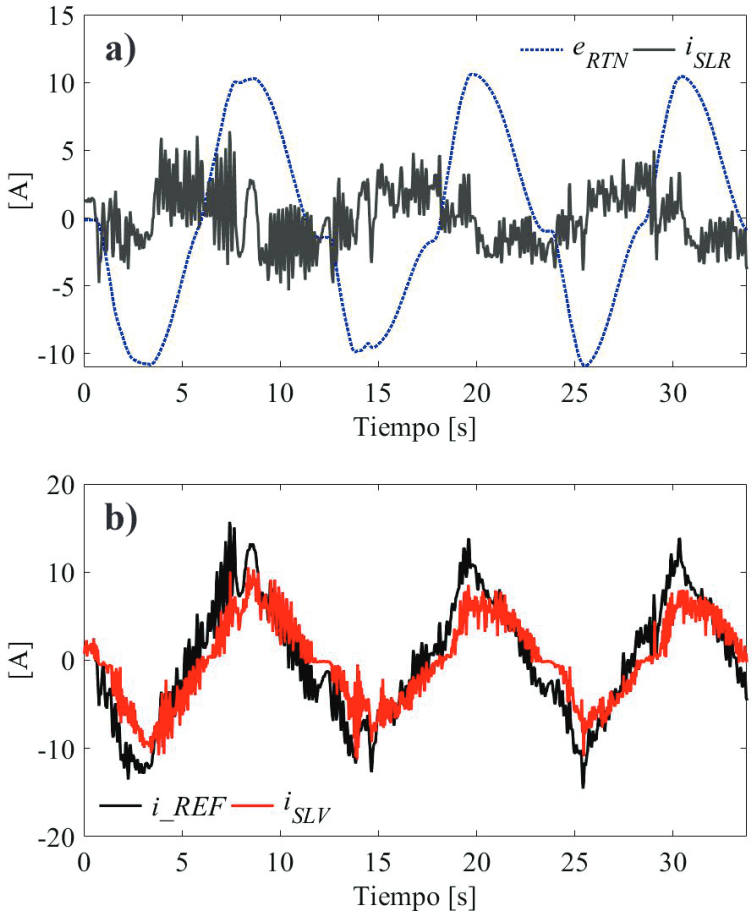

Figura 7.2: Respuesta del control PID sin perturbación a) compensación de retorno a posición cero y corriente medida en el SLR y b) regulación de corriente en el SLV.

La regulación de posición angular en el lado rueda, se observa en la Figura 7.3 a). La posición angular medida en el SLR (Pos_{SLR}) iguala a la posición angular del volante (Pos_{SLV}). Esta acción de control se realiza con un porcentaje de error que ronda en $\pm$ 0.3 % (véase Figura 7.3 b)). En esta prueba el volante tiene un desplazamiento angular de 8π radianes. Por otro lado, en la Figura 7.4 se muestra las señales de control (Uav_SLV y Uav_SLR) para los dos subsistemas.

Estas señales representan el ciclo de trabajo para generar las salidas PWM, que se suministran en los puentes H. Debido a la inexistencia de cargas externas en el SLR, solo se generan señales PWM al 50 % de ciclo de trabajo aproximadamente.

7.1.2. Carga aplicada en el subsistema lado rueda

El objetivo de esta prueba consiste en tener una retroalimentación de par en el volante. Para ello, se aplica carga en el motor del subsistema lado rueda con el freno de disco manual. Los

movimientos realizados en el volante tienen una menor amplitud y mayor rapidez, a diferencia del caso anterior.

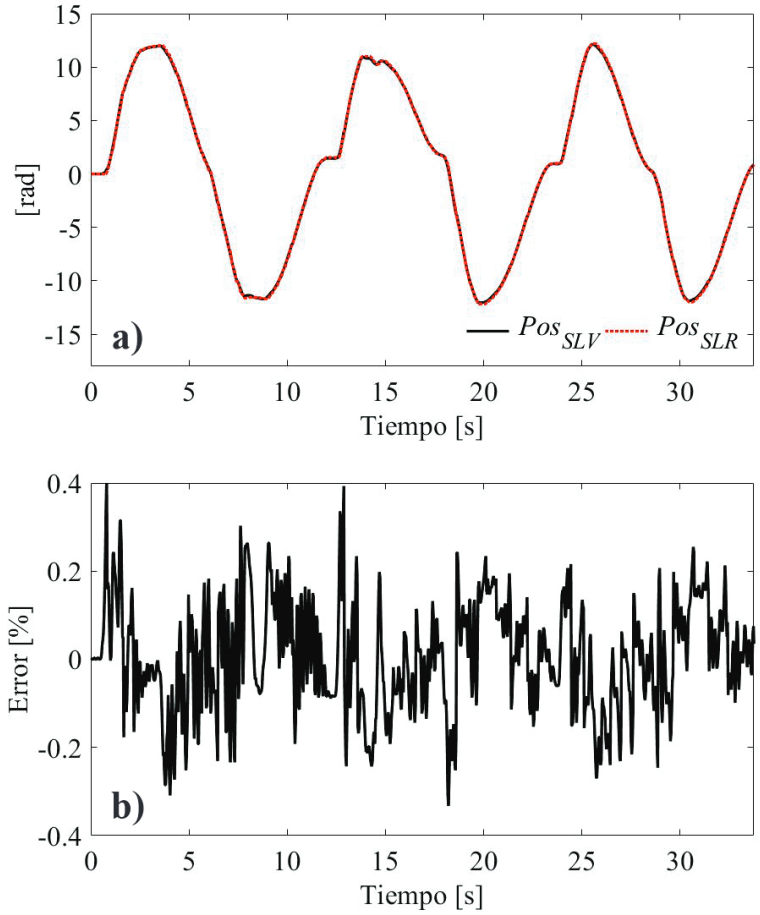

Figura 7.3: Respuesta del control PID sin perturbación a) regulación de posición angular en el SLR y b) porcentaje de error en el seguimiento.

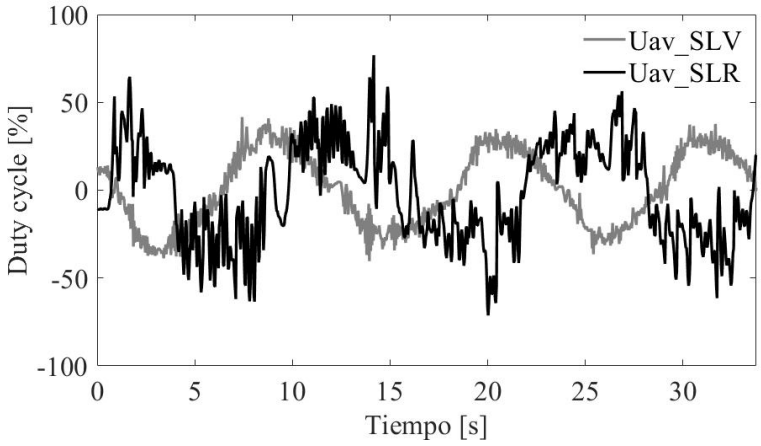

Figura 7.4: Señales de control resultantes en la prueba de seguimiento sin perturbaciones de par.

En la Figura 7.5 a) se observa la corriente de referencia i_{SLR} que representa el par aplicado en el lado rueda. En esta señal se añade la compensación de retorno (e_{RTN}), originando la señal i_REF que se debe sentir en el volante. En la Figura 7.5 b) se muestra que la corriente consumida en el motor del SLV (i_{SLV}) sigue el mismo comportamiento que la corriente de referencia total (i_REF). Logrando así, la sensación de conducción. En seguida se muestran los resultados obtenidos en el control PID de posición angular (véase Figura 7.6).

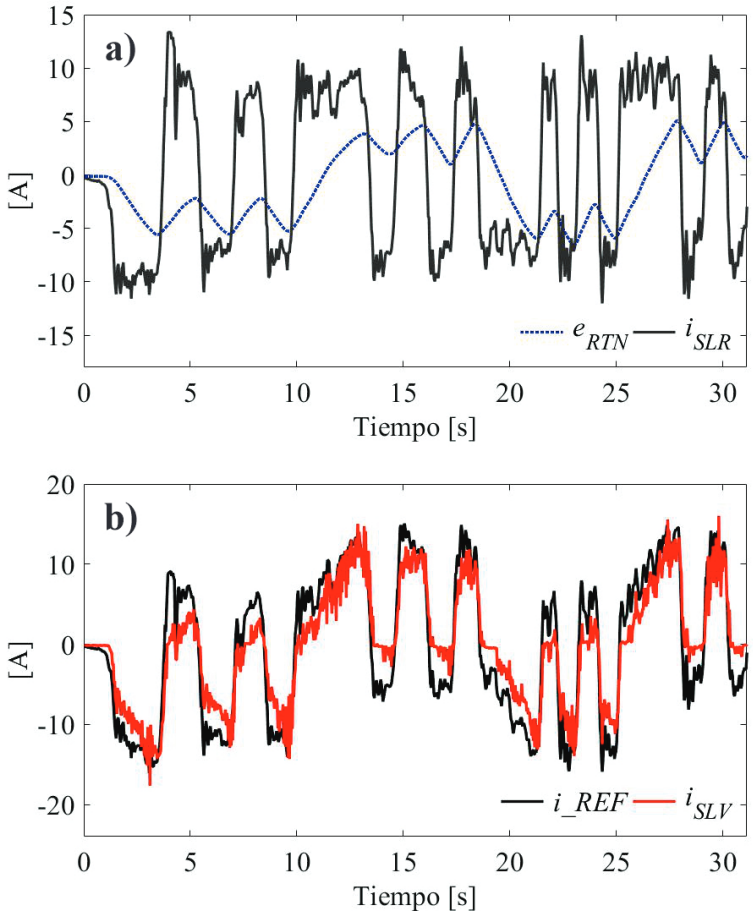

Figura 7.5: Respuesta del control PID con carga aplicada a) compensación de retorno a posición cero y corriente medida en el SLR y b) regulación de corriente en el SLV.

En la Figura 7.6 a) se observa un seguimiento adecuado de posición angular del SLR ante aplicaciones de par en el disco. La intención de esta prueba es analizar el desempeño del control de posición, ante movimientos rápidos y pequeños, mientras se aplica un par arbitrario al SLR. La acción del freno ocasiona que el porcentaje de error aumente (véase Figura 7.6 b)), en comparación con el caso anterior.

Las salidas de control para los dos subsistemas se presentan en la Figura 7.7. Debido a la aplicación de carga se hace evidente el aumento en las señales Uav_SLV y Uav_SLR. Esto significa

un aumento en las maniobras de compensación, tanto de corriente (proporcional al par aplicado) como de posición angular.

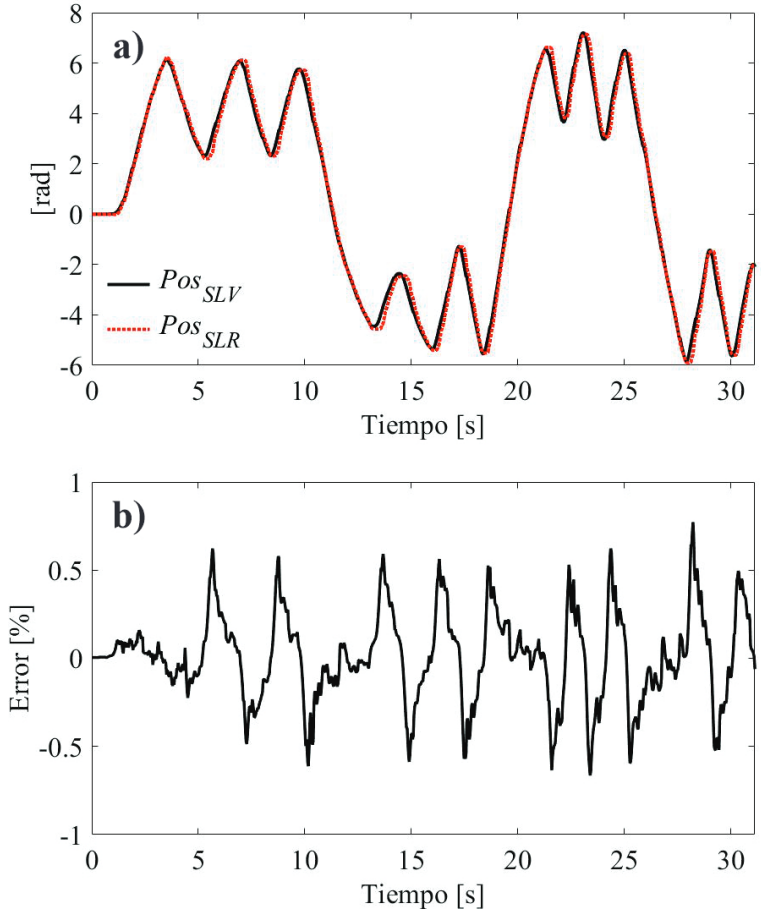

Figura 7.6: Respuesta del control PID con carga aplicada a) regulación de posición angular en el SLR y b) porcentaje de error en el seguimiento.

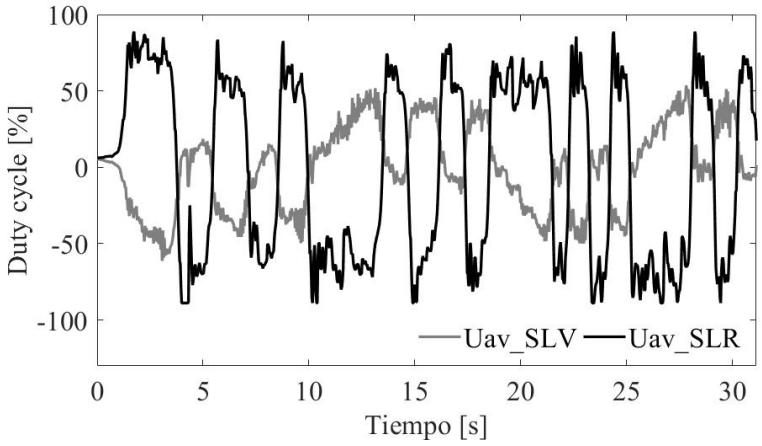

Figura 7.7: Señales de control resultantes en la prueba de carga aplicada en el SLR.

7.1.3. Maniobras alrededor de la posición cero

Esta prueba experimental consiste en mover el volante en ambas direcciones, alrededor de la posición angular cero, mientras se aplica par al motor del SLR. Estas maniobras simulan un movimiento en zig zag como si se evadiera obstáculos. En la Figura 7.8 a) se observa la compensación del retorno a la posición central del volante (e_{RTN}) y la corriente consumida en el actuador del lado rueda (i_{SLR}). Estas dos señales se suman para obtener la señal de referencia total del SLV (i_REF).

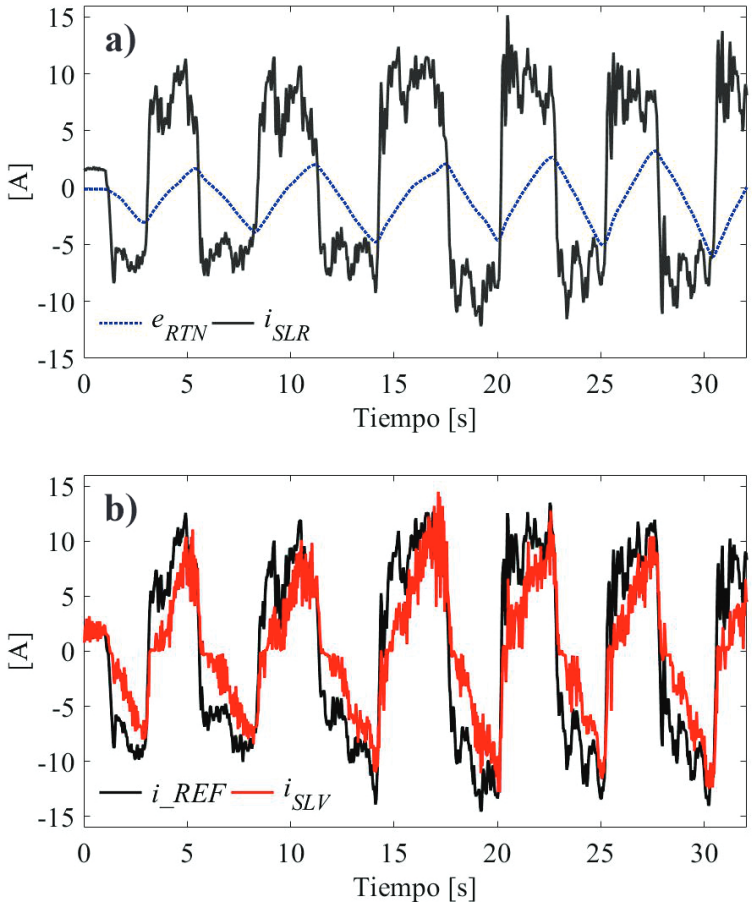

Figura 7.8: Respuesta del control PID con maniobras alrededor de la posición cero a) compensación de retorno del volante y corriente medida en el SLR y b) regulación de corriente en el SLV.

La retroalimentación de par (que es proporcional a la corriente consumida en el motor) se muestra en la Figura 7.8 b). La corriente consumida por el motor del SLV (i_{SLV}) sigue adecuadamente el comportamiento de la señal de referencia i_REF. Por otro lado, en la Figura 7.9 a) se observa la respuesta del control PID de posición angular en el SLR. El giro del volante (Pos_{SLV}) se mantiene alrededor de la posición central (0 rad), con movimientos que se hacen más grandes conforme pasa el tiempo. Bajo este escenario, el seguimiento de posición angular es adecuado, tal como lo muestra

la Figura 7.9 b) en donde se comprueba que el error se mantiene en un rango máximo del 0.65 %. En seguida se presentan las señales de control, que muestran las maniobras de compensación necesarias para mantener el seguimiento de la referencia deseada (véase Figura 7.10).

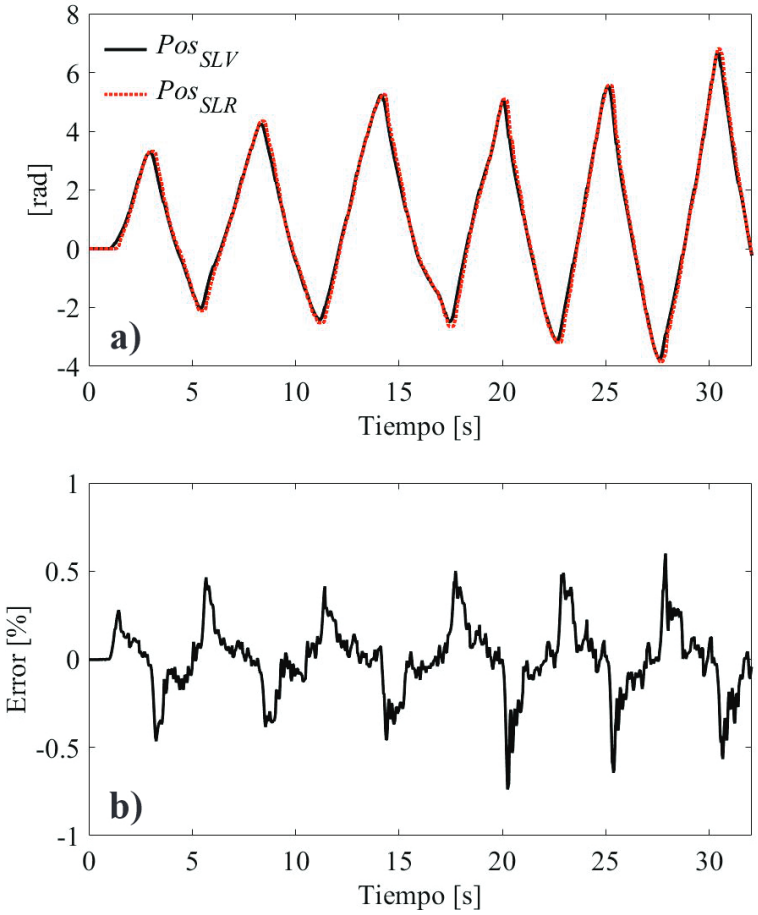

Figura 7.9: Respuesta del control PID con maniobras alrededor de la posición cero a) regulación de posición angular en el SLR y b) porcentaje de error en el seguimiento.

7.2. Pruebas en los escenarios de falla

En el prototipo de dirección steer-by-wire se simulan ciertos escenarios de falla, con el objetivo de evaluar los mecanismos de seguridad diseñados. La interfaz gráfica desarrollada en App Designer es una herramienta auxiliar para mostrar el funcionamiento de estos mecanismos. Además, se utiliza para desplegar en pantalla (PC) los datos de algunos parámetros del sistema. En la Figura 7.11 se observa la integración de la interfaz gráfica en las pruebas de escenarios de falla (véase también Figura 7.12).

En la parte superior de la interfaz se encuentra el botón de *Inicio* y la cantidad de muestras a

capturar. Esta cantidad se establece en 200, con la posibilidad de ser modificada. En la parte inferior se observa una barra que indica el progreso de la captura de muestras. Por otro lado, en la parte central izquierda se presentan los datos de temperatura ($°C$), el giro del volante (medido en grados) y el porcentaje de error en la regulación de posición angular. También se colocan 3 leds para indicar el estado de funcionamiento del sistema. Por lo tanto, el funcionamiento normal y las fallas de los subsistemas se indican en el LED_{RGB} de la plataforma experimental y en la interfaz.

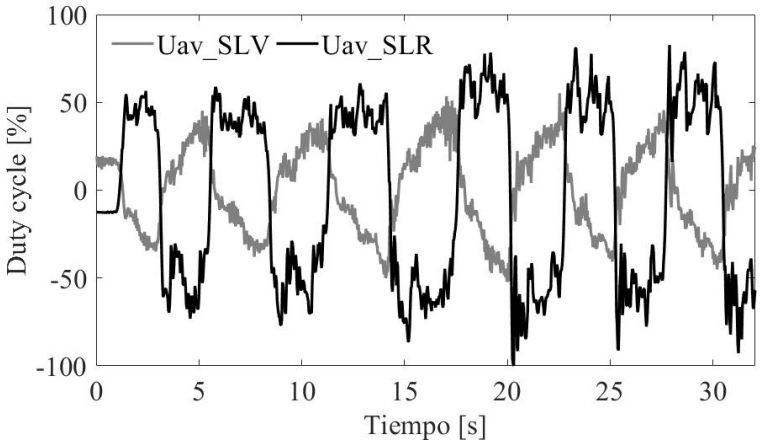

Figura 7.10: Señales de control resultantes en la prueba de maniobras alrededor de la posición cero.

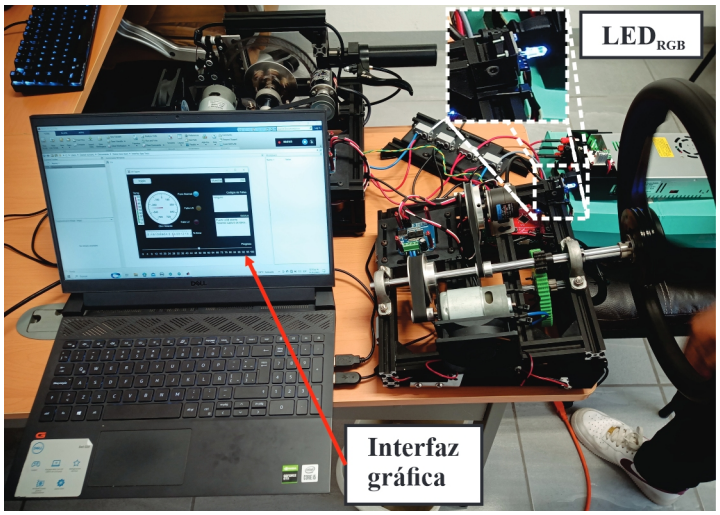

Figura 7.11: Pruebas en el sistema steer-by-wire usando la interfaz gráfica desarrollada.

Finalmente, en la parte central derecha se agrega un recuadro de *Estatus* para mostrar las condiciones del bus serial (abierto ó cerrado) e indicar el proceso de captura de datos (inicio y finalización). Arriba de este recuadro se presentan los códigos de fallas que se registran en el sistema (establecidos siguiendo el protocolo OBD-II pero adaptados a este sistema en particular):

- *C1011*: Falla del SLV, alerta de consumo de corriente y modo seguro activado.
- *C1021*: Falla del SLR, alerta de consumo de corriente.
- *C1022*: Falla del SLR, alerta de temperatura del motor.

A continuación, se presentan los resultados obtenidos en las pruebas de los escenarios de falla.

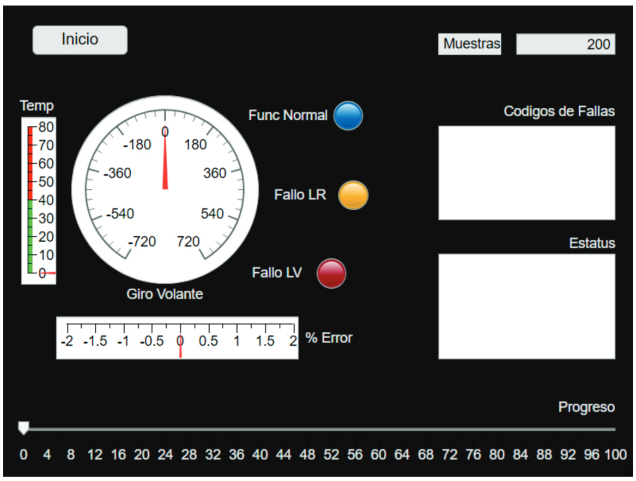

Figura 7.12: Intefaz gráfica de despliegue de información desarrollada en App Designer de MatLab.

7.2.1. Superación de umbrales de corriente eléctrica

En este escenario de falla se pone a prueba el algoritmo de detección de sobre corriente, implementado en cada subistema. Se forza al motor del lado rueda a consumir valores de corriente por encima de los umbrales establecidos ($\pm$ 14 A). Estos umbrales también se utilizan en el algoritmo del SLV, debido a que el motor de retroalimentación alcanza los mismos niveles de corriente que el actuador del SLR.

En la Figura 7.13 se observa la superación de umbrales y su detección en ambos subsistemas. En esta prueba la primera alerta de corriente que se activa es la del SLV (wrg_i_{SLV}), a los 24 segundos aproximadamente (indicado con **E1** en la Figura 7.13 a)). Esto se debe al consumo de corriente en el motor del lado rueda (i_{SLR}) y a la compensación de retorno del volante (e_{RTN}). La suma de estas dos señales origina valores de referencia (i_REF) que sobrepasan el umbral de -14 A.

El control indirecto de par ocasiona que el motor acoplado al volante consuma niveles de corriente cercanos a la señal de referencia. Una vez que i_{SLV} supera el umbral, el algoritmo de detección es aplicado. Cuando el tiempo fuera del umbral establecido alcanza los 2 segundos, se dispara la alerta de corriente (wrg_i_{SLV}) y entra en funcionamiento el modo seguro. En la Figura 7.14 a), se señala el momento (**E1**) en que esta alerta es activada.

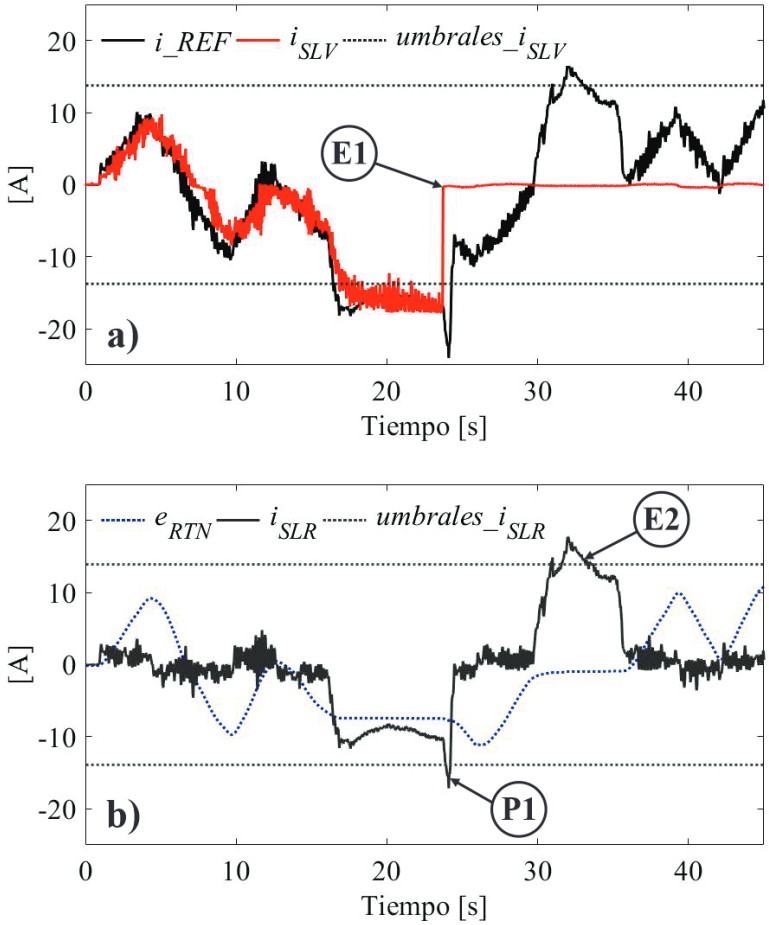

Figura 7.13: Superación de los umbrales de corriente establecidos a) activación del modo seguro en el SLV y b) compensación de retorno del volante y corriente medida en el SLR.

Cuando el modo seguro se habilita, la señal de control del SLV (Uav_SLV) toma el valor de cero. Esto se señala en la Figura 7.14 b) y lleva a la desactivación de la retroalimentación de par en el volante. Además al quedar inhabilitado el motor del SLV, la función de retorno deja de funcionar y el actuador no consume corriente (véase Figura 7.13 a)).

Con respecto a la activación de la alerta de corriente en el SLR, esta ocurre a los 33 segundos (marcado con **E2** en la Figura 7.13 b)). Cuando se aplica nuevamente carga al motor del lado rueda e i_{SLR} supera el umbral de +14 A. Anteriormente, se presenta un pico de corriente que el algoritmo no detecta como falla (**P1**). El algoritmo de detección pone en estado alto a la bandera wrg_i_{SLR}, mientras el tiempo de superación sea igual o mayor a 1 segundo. El momento en que se activa esta bandera de alerta, se indica con la etiqueta **E2** y se observa en la Figura 7.14 a). Por otro lado, la regulación de posición angular en el SLR sigue funcionando a pesar del modo seguro (véase Figura 7.15 a)). En la Figura 7.15 a) se observa que después de generarse la alerta de corriente del SLV, se tiene una perturbación en el seguimiento de posición. Esta perturbación se compensa

rápidamente y es ocasionada por soltar el freno de disco. La respuesta del control PID de posición es adecuada en este escenario de falla. La posición angular medida en el SLR (Pos_{SLR}) iguala a la posición de referencia (Pos_{SLV}). El porcentaje de error se muestra en la Figura 7.15 b). Por último, el incremento de la señal Uav_SLR en la Figura 7.14 b), corresponde a los pares aplicados para generar las alertas de corriente.

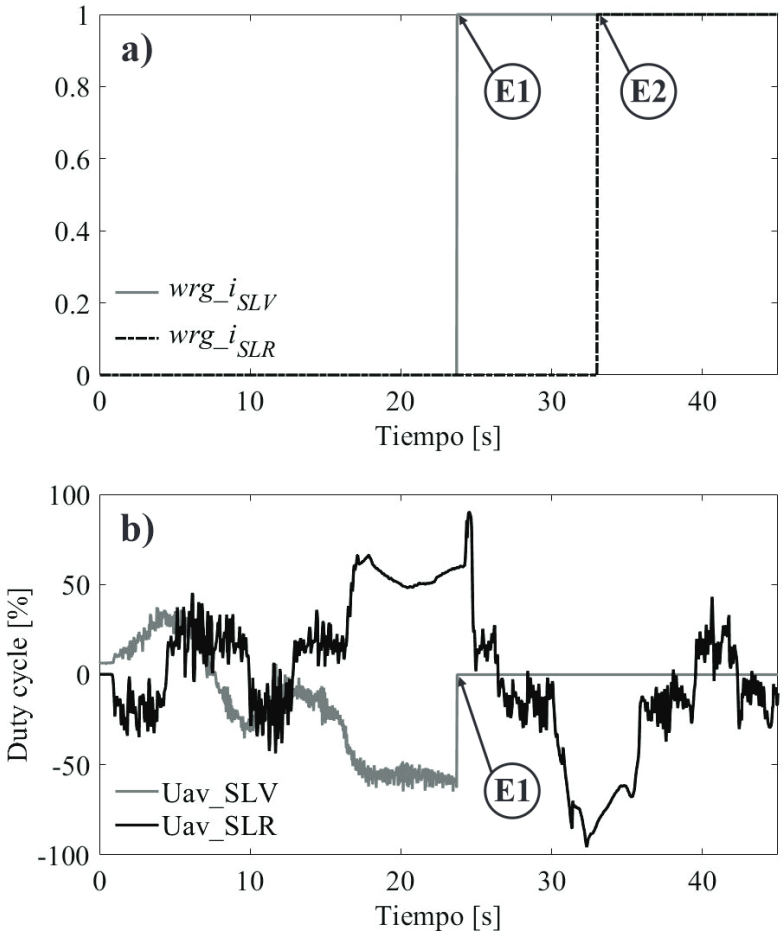

Figura 7.14: Respuesta del algoritmo de detección de falla a) estado de las alertas de corriente y b) señales de control resultantes en el escenario de falla.

7.2.2. Aumento de la temperatura en el motor del SLR

El monitoreo de temperatura en el motor del SLR, se lleva a cabo mediante un sensor LM35 que se observa en la Figura 7.16 a). La aplicación de par en el motor del lado rueda ocasiona el aumento de su temperatura. Sin embargo, una prueba de funcionamiento que genere una temperatura lo suficientemente elevada, toma un tiempo relativamente grande y puede ser destructiva. Para simular una mayor temperatura en el motor del lado rueda, se acerca una fuente de calor (cautín). Los

resultados de la temperatura medida ($°C_{SLR}$) se presentan en la Figura 7.16 b). El punto E_T indica el momento en que se supera el umbral establecido (40 $°C$, aunque según el fabricante soporta hasta 80 $°C$). En ese instante la alerta correspondiente se activa ($wrg\_Temp = 1$) y se visualiza en la interfaz gráfica.

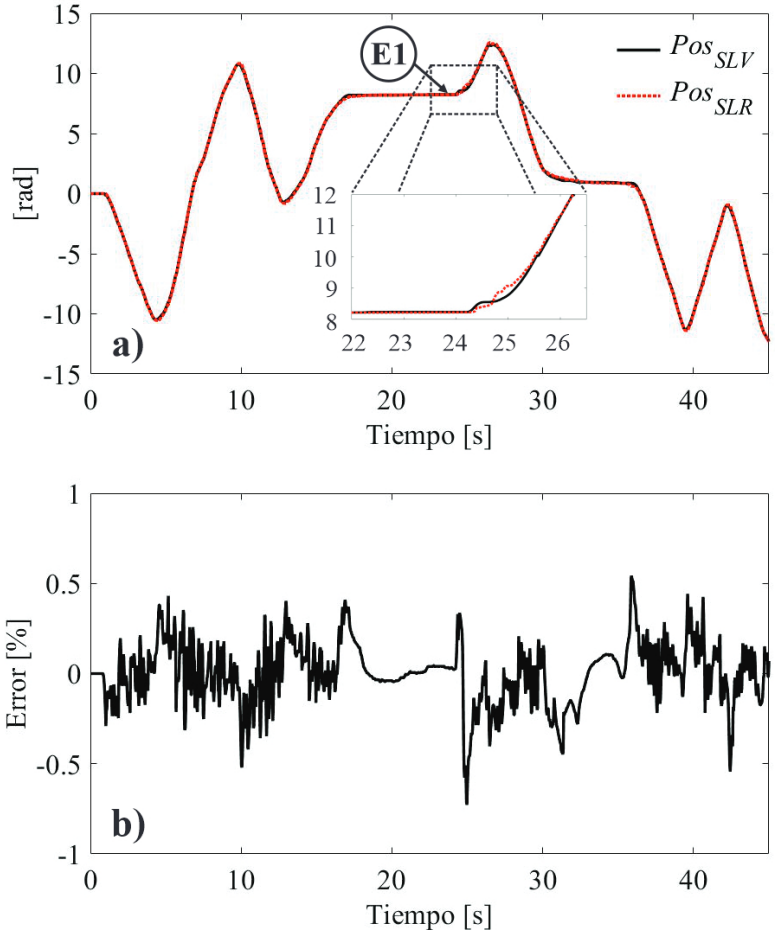

Figura 7.15: Respuesta del control PID ante la superación de umbrales de corriente a) regulación de posición angular en el SLR y b) porcentaje de error en el seguimiento.

7.3. Conclusiones

En este trabajo de tesis, se diseña e implementa un prototipo de dirección steer-by-wire, misma que es puesta a prueba para evaluar su desempeño. La característica principal de este trabajo, es que consta de dos esquemas de control descentralizados e interconectados mediante CAN Bus. Por el canal de comunicación se transmite la posición angular del volante y la corriente consumida por el motor del lado rueda. Además, de información de alertas y datos de temperatura medida. Al realizar las pruebas de funcionamiento se confirma el correcto intercambio de información entre

los dos subsistemas. El tiempo de respuesta y la integridad de los datos enviados por CAN Bus, son adecuados para llevar a cabo las tareas de control.

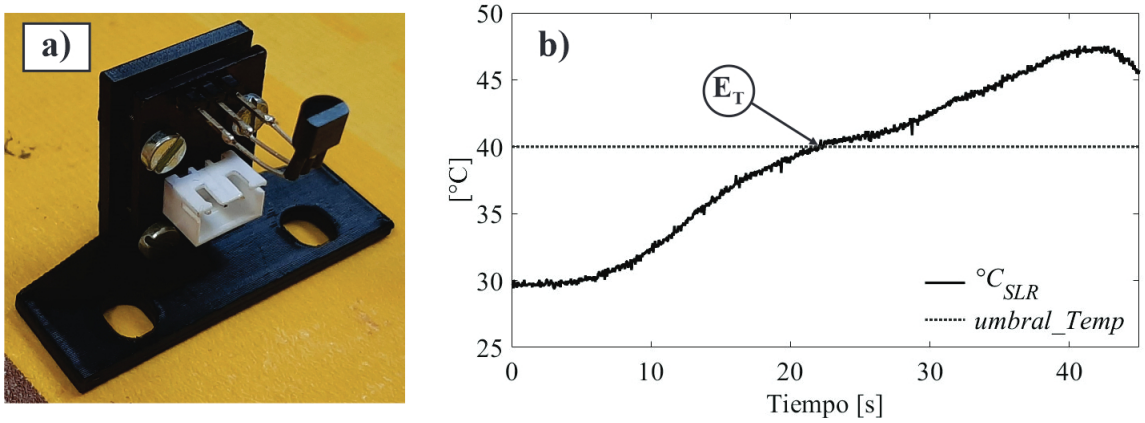

Figura 7.16: Monitoreo de temperatura a) sensor LM35 colocado en su base y b) superación del umbral de temperatura establecido para el motor del SLR.

La simulación desarrollada en el softaware de Simulink de MatLab, es de gran ayuda para entender la lógica de la función de retorno a la posición cero del volante. Con esta simulación se demostró que es posible desarrollar y llevar esta función al código del microcontrolador. Además, el desempeño del retorno del volante se considera adecuado. Por otro lado, con los resultados obtenidos en los escenarios de prueba (condiciones normales) se demuestra que los controles PID implementados son lo suficientemente robustos y estables. Sin embargo, se requiere mejorar ciertos aspectos. Principalmente en la retroalimentación de par, para que el volante oponga mayor resistencia. Estas mejoras necesitan ser consideradas desde el aspecto mecánico y de diseño (transmisión de potencia), e incluso optar por un actuador con mayor potencia.

Los escenarios de falla se pensaron de tal manera que el sistema se mantenga con compeljidad y costos relativamente moderados. Tanto en un sistema real como en el prototipo desarrollado, es necesario cuidar la integridad de los actuadores. Los motores que se utilizan tienen un consumo de corriente nominal y si el nivel se supera estos se dañarán. Por ello, se propone el algoritmo de detección de falla de corriente, que determina la superación de umbrales establecidos. Es un mecanismo de seguridad para alertar al usuario, la falla en el consumo de corriente de uno o ambos motores. En el subistema lado volante se tiene la posibilidad de implementar un modo seguro, donde el actuador se inhabilita sin afectar el seguimiento de posición. Los resultados de esta prueba se consideran adecuados y dentro de un rango aceptable. Con respecto al monitoreo de temperatura, solo se considera en el motor del SLR. Debido a que el seguimiento de posición angular que realiza este actuador es muy importante y crítico. El usuario debe estar consciente de las condiciones de funcionamiento del motor del lado rueda. Con lo descrito anteriormente, se comprueban de manera

adecuada las hipótesis planteadas en el **Capítulo 1**.

7.4. Trabajos futuros

A continuación, se presenta una lista de trabajos futuros relacionados con mecanismos de seguridad y el rediseño del prototipo de dirección steer-by-wire.

I. Desarrollo de sistemas de redundancia para incrementar la seguridad:

- Redundancia de líneas de comunicación.
- Redundancia de unidades de control con su respectiva fuente de respaldo.
- Redundancia de actuadores en el lado rueda.

II. Desarrollo del sistema mecánico tamaño real del lado rueda, con un actuador de acuerdo a los requerimientos.

III. Diseño y manufactura de una estructura modular en donde se incluya toda la electrónica incluidos los sensores, actuadores y dispositivos de procesamiento. La cual cumpla con requerimientos específicos de volumen, peso, par otorgado y sensores especializados.

IV. Desarrollo e implementación de sistemas de asistencia de dirección, por ejemplo: sistema de mantenimiento de carril, ajuste automático de par aplicado al volante, ángulo de giro variable, conducción autónoma, entre otros.

Referencias

[1] M. Würges, "New Electrical Power Steering Systems," en *Encyclopedia of Automotive Engineering*. John Wiley & Sons, Ltd, 2013, págs. 1-17, ISBN: 9781118354179. DOI: https://doi.org/10.1002/9781118354179.auto008.

[2] A. Gaedke, M. Heger, M. Sprinzl, S. Grüner y A. Vähning, "Electric Power Steering Systems," en *Steering Handbook*, M. Harrer y P. Pfeffer, eds. Cham: Springer International Publishing, 2017, págs. 403-467, ISBN: 978-3-319-05449-0. DOI: 10.1007/978-3-319-05449-0_15.

[3] F. Serrano, B. A. Rodríguez-Gómez y M. A. F. Barahona, "Control del Par en Motores de Corriente Directa por Control de Modelo Interno (IMC)," *3C Tecnología. Glosas de innovación aplicadas a la pyme*, vol. 8, n.º 1, págs. 42-63, 2019. DOI: http://dx.doi.org/10.17993/3ctecno/2019.v8n1e29/42-63.

[4] A. Beltrán, J. Rumbo, H. Azcaray, K. Santiago, M. Calixto y E. Sarmiento, "Simulación y control de la velocidad y par electromagnético de un motor de inducción trifásico: Un enfoque a vehículos eléctricos," *Revista Iberoamericana de Automática e Informática industrial*, vol. 16, n.º 3, págs. 308-320, jun. de 2019. DOI: 10.4995/riai.2019.10452.

[5] D. Hart, *Electrónica de potencia* (Fuera de colección Out of series). Pearson Educación, 2001, ISBN: 9788420531793.

[6] M. Rashid, *Electrónica de potencia: circuitos, dispositivos y aplicaciones* (Área: Ingeniería). Pearson Educación, 2004, ISBN: 9789702605324.

[7] A. Shrinath y A. Emadi, "Electronic control units for automotive electrical power systems: Communication and networks," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 218, n.º 11, págs. 1217-1230, 2004. DOI: 10.1243/0954407042579996.

[8] D. Andrango y M. Luis, "Estudio del protocolo CAN (Controller Area Network) y su aplicación en redes de control," Tesis de licenciatura, Escuela Politécnica Nacional, Repositorio Institucional, 2007. dirección: http://bibdigital.epn.edu.ec/handle/15000/403.

[9] J. Beltrán Zambrano, "Desarrollo de un simulador electrónico de una ECU y su diagnóstico sobre CAN y OBD-II," Tesis de licenciatura, Universidad de Sevilla, Repositorio Institucional, 2015.

[10] C. Huang, F. Naghdy, H. Du y H. Huang, "Fault tolerant steer-by-wire systems: An overview," *Annual Reviews in Control*, vol. 47, págs. 98-111, 2019, ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2019.04.001.

[11] M. R. Crawford, "A Steer-by-Wire Test Bed for Position Control and Force Feedback," Bachelor thesis, University of Wollongong, 2017.

[12] J. Pimentel, "An Architecture for a Safety-Critical Steer-by-Wire System," *SAE Technical Papers*, mar. de 2004. DOI: 10.4271/2004-01-0714.

[13] L. Eckstein, L. Hesse y M. Klein, "Steer-by-Wire, Potential, and Challenges," en *Encyclopedia of Automotive Engineering*. John Wiley & Sons, Ltd, 2014, ISBN: 9781118354179. DOI: https://doi.org/10.1002/9781118354179.auto010.

[14] D. Semmel, "Hydraulic Power Supply," en *Steering Handbook*, M. Harrer y P. Pfeffer, eds. Cham: Springer International Publishing, 2017, págs. 357-379, ISBN: 978-3-319-05449-0. DOI: 10.1007/978-3-319-05449-0_13.

[15] J. Gessat, A. Seewald y D. Zimmermann, "Electrically Powered Hydraulic Steering," en *Steering Handbook*, M. Harrer y P. Pfeffer, eds. Cham: Springer International Publishing, 2017, págs. 381-401, ISBN: 978-3-319-05449-0. DOI: 10.1007/978-3-319-05449-0_14.

[16] M. Reuter y A. Saal, "Superimposed Steering System," en *Steering Handbook*, M. Harrer y P. Pfeffer, eds. Springer International Publishing, 2017, págs. 469-492, ISBN: 978-3-319-05449-0. DOI: 10.1007/978-3-319-05449-0_16.

[17] P.-S. Huang y A. Pruckner, "Steer by Wire," en *Steering Handbook*, M. Harrer y P. Pfeffer, eds. Cham: Springer International Publishing, 2017, págs. 513-526, ISBN: 978-3-319-05449-0. DOI: 10.1007/978-3-319-05449-0_18.

[18] R. Hayama, M. Higashi, S. Kawahara, S. Nakano y H. Kumamoto, "Fault-Tolerant Architecture of Yaw Moment Management with Steer-by-Wire, Active Braking and Driving-Torque Distribution Integrated Control," en *SAE World Congress & Exhibition*, SAE International, 2008. DOI: https://doi.org/10.4271/2008-01-0110.

[19] "Fahrwerksysteme," en *Handbuch Kraftfahrzeugelektronik: Grundlagen, Komponenten, Systeme, Anwendungen*, H. Wallentowitz y K. Reif, eds. Vieweg, 2006, págs. 113-177, ISBN: 978-3-8348-9121-1. DOI: 10.1007/978-3-8348-9121-1_3.

[20] Y. Onoda, Y. Onuma, T. Goto y T. Sugitani, "Design Concept and Advantages of Steer-by-Wire System," en *SAE World Congress & Exhibition*, SAE International, 2008. DOI: https://doi.org/10.4271/2008-01-0493.

[21] B. Zheng y S. Anwar, "Fault-Tolerant Control of the Road Wheel Subsystem in a Steer-By-Wire System," *International Journal of Vehicular Technology*, vol. 2008, ene. de 2008. DOI: 10.1155/2008/859571.

[22] C. Zong, H. Xiang, L. He y F. Sha, "Study on control method of dual-motor for steer-by-wire system," en *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, 2012, págs. 2890-2893. DOI: 10.1109/CECNet.2012.6201884.

[23] M. Z. Mohd Tumari, M. S. Saealal, W. N. Abd Rashid, S. Saat y M. A. Mohd Nasir, "The Vehicle Steer by Wire Control System by Implementing PID Controller," *JTEC*, vol. 9, n.º 3-2, págs. 43-47, oct. de 2017.

[24] S. M. Sahboun y A. A. A. Emhemed, "Controller Design for Steer-by-Wire System," *Journal of Mechatronics and Robotics*, vol. 6, págs. 1-6, ene. de 2022. DOI: 10.3844/jmrsp.2022.1.6.

[25] Y. Nianjiong y L. Qifeng, "Design and Simulation for Steer-by-Wire System Based on Fuzzy-PID," en *2015 7th International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 1, 2015, págs. 291-294. DOI: 10.1109/IHMSC.2015.43.

[26] H. Chen, L. Zhu, X. Liu, S. Yu y D. Zhao, "Study on Steering by Wire Controller Based on Improved H∞ Algorithm," *International Journal of Online Engineering (iJOE)*, vol. 9, n.º S2, págs. 35-40, mar. de 2013. DOI: 10.3991/ijoe.v9iS2.2569.

[27] Z. Sun, J. Zheng, Z. Man y H. Wang, "Robust Control of a Vehicle Steer-by-Wire System Using Adaptive Sliding Mode," *IEEE Transactions on Industrial Electronics*, vol. 63, n.º 4, págs. 2251-2262, 2016. DOI: 10.1109/TIE.2015.2499246.

[28] A. E. Cetin, M. Arif Adli, D. E. Barkana y H. Kucuk, "Adaptive on-line parameter identification of a steer-by-wire system," *Mechatronics*, vol. 22, págs. 152-166, 2012, ISSN: 0957-4158. DOI: https://doi.org/10.1016/j.mechatronics.2012.01.002.

[29] J. Álvarez. "Saviez-vous que cette Mercedes se conduisait avec des joysticks?" (2020), dirección: https://fr.motor1.com/news/452717/mercedes-benz-concept-joysticks/ (visitado 15-10-2023).

[30] Supercars.net. "GM Hy-Wire Concept." (2021), dirección: https://bit.ly/GMH-Wire (visitado 15-10-2023).

[31] J. Costas. "¿Podemos fiarnos de la dirección asistida sin conexión mecánica (by-wire)?" (2016), dirección: https://bit.ly/motorp-q50 (visitado 15-10-2023).

[32] Lexus-Europe-Newsroom. "World Premiere of the All-New Lexus RZ." (2022), dirección: https://newsroom.lexus.eu/world-premiere-of-the-all-new-lexus-rz/ (visitado 15-10-2023).

[33] Toyota-Newsroom. "Details of All-New bZ4X BEV Announced." (2021), dirección: https://global.toyota/en/newsroom/toyota/36254760.html (visitado 15-10-2023).

[34] M. A. Rodríguez Lendoiro, "Riesgos inherentes a los intervinientes en siniestros con vehículos," Tesis de licenciatura, Universidad de las Palmas de Gran Canaria, Repositorio Institucional, 2019.

[35] A. Perez, O. Berreteaga, A. Ruiz, A. Urkidi y J. Perez-Cerrolaza, "Una metodología para el desarrollo de hardware y software embebidos en sistemas críticos de seguridad," *Sistemas, cibernética e informática*, vol. 3, n.º 2, págs. 70-75, 2006. dirección: https://www.iiisci.org/journal/PDV/risci/pdfs/C863GM.pdf.

[36] L. Hernández, C. Isaza, F. R. Trejo-Macotela, K. Anaya y J. P. Zavala De Paz, "Comparación cualitativa de protocolos telemáticos intravehiculares," en *Ciencias de la Tecnología e Innovación, Handbook T-I*, 1.ª ed. ECORFAN-México, S.C., 2016, vol. 1, págs. 67-77.

[37] O. Avatefipour, A. Hafeez y H. Malik, "Linking Received Packet to the Transmitter Through Physical-Fingerprinting of Controller Area Network," ene. de 2018. DOI: 10.1109/WIFS.2017.8267643.

[38] D. Encinas, P. Meilan, J. A. Bava y M. Naiouf, "Protocolo de comunicaciones CAN aplicado a sistemas satelitales y vehículos lanzadores," en *XV Congreso Argentino de Ciencias de la Computación*, 2009.

[39] S. F. Lokman, A. Othman y M. Husaini, "Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, n.º 184, jul. de 2019. DOI: 10.1186/s13638-019-1484-3.

[40] J. D. Rairán-Antolines, C. E. Guerrero-Cifuentes y J. A. Mateus-Pineda, "Diseño de controladores de tipo proporcional integral derivativo (PID) y difuso para la posición de un motor de corriente continua (DC)," *Ingeniería y Universidad*, vol. 14, n.º 1, págs. 137-160, 2010. dirección: http://www.redalyc.org/articulo.oa?id=47715438007.

[41] K. J. Åström y T. Hägglund, *Control PID avanzado*. Pearson, Madrid, 2009.

[42] K. Ogata, *Ingeniería de control moderna*, 5ª ed. Madrid: Pearson, 2010, ISBN: 978-84-8322-660-5.

[43] G. C. Goodwin, S. F. Graebe y M. E. Salgado, *Control System Design*, 1ª ed. USA: Prentice Hall PTR, 2000, ISBN: 0139586539.

[44] H. A. Sánchez Ortiz. "Control PID. Discretización e implementación en un microcontrolador." (2022), dirección: https://www.youtube.com/watch?v=OiqDurT_XSw (visitado 27-03-2024).

[45] M. S. Alvarez Alvarado, "Modelo matemático de un motor de corriente continua separadamente excitado: Control de velocidad por corriente de armadura," *Lat. Am. J. Phys. Educ.*, vol. 6, n.º 1, págs. 155-161, 2012.

APÉNDICE

Apéndice A

Programación del sistema steer-by-wire

A.1. Código del subsistema lado volante

```
//********** Bibliotecas **********//
#include <stdint.h>
#include <float.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_can.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/adc.h"
#include "driverlib/pwm.h"
#include "driverlib/qei.h"
#include "driverlib/udma.h"
#include "driverlib/systick.h"
#include "driverlib/can.h"
#include "utils/uartstdio.h"

//***************** Variables declaradas *****************//
//********** Variables del bus CAN bidireccional *********//
//******************** Para transmitir *******************//
// Contador de interrupciones, igual a cantidad de mensajes transmitidos.
volatile uint32_t MsgCount_1 = 0;
// Bandera de error del canal CAN (bus desconectado).
volatile bool errFlag_SLV = 0;
// Variable que almacena las posiciones del volante, es dividida y
// transmitida por el bus CAN como 2 bytes.
unsigned short int lectura_1 = 0;
```

```
// Variable que almacena el byte menos significativo de lectura_1.
unsigned char dataTX0;
// Variable que almacena el byte mas significativo de lectura_1.
unsigned char dataTX1;
// Bandera para indicar que el CAN debe o no transmitir datos.
// Inicialmente transmite los datos del subsistema lado volante.
volatile bool T_SLV = 1;

//******************** Para recibir ******************//
// Contador de interrupciones, igual a cantidad de mensajes recibidos.
volatile uint32_t MsgCount_3 = 0;
// Bandera para que el manejador de interrupciones indique si un
// mensaje es recibido.
volatile bool rxFlag_SLV = 0;
// Variable encargada de almacenar el valor rearmado de la corriente
// del subsistema lado rueda, recibido en 2 bytes.
unsigned short int rearmado_2 = 2033;
// Variable que almacena el byte menos significativo del valor de
// corriente.
unsigned char dataRX0;
// Variable que almacena el byte mas significativo del valor de corriente.
unsigned char dataRX1;
// Variable del nuevo rango de valor para la corriente.
int16_t Ref_Current;
// Variable que almacena la alerta de corriente del lado rueda.
unsigned char dataRX2;
// Variable que procesa la alerta de corriente recibida.
int16_t wrg_iSLR = 0;
// Variable que almacena los datos de temperatura recibidos en el bus CAN.
unsigned char dataRX3;
// Variable que toma los datos de temperatura para su procesamiento.
int16_t Temp_SLR = 0;
// Alerta de temperatura elevada en el motor del subsistema lado rueda.
int16_t wrg_Temp = 0;
// Bandera para indicar que el CAN debe o no recibir datos. Inicialmente
// se coloca en bajo para dar prioridad al envio de datos.
volatile bool R_SLV = 0;

//******** Variables para la etapa de potencia (driver BTS7960) ********//
// Frecuencia para el periodo de PWM de 10 KHz.
#define F_Hz 10000
// Variable del periodo PWM.
uint32_t T_ciclos;
// Variables del ciclo de trabajo.
double C_Duty;
double C_Duty2;

//****** Variables para la lectura ADC del sensor ACS711EX ******//
// Frecuencia de muestreo de 40 KHz.
#define FS 40000
// Variable que obtiene las muestras del ADC.
uint32_t s[8];
// Variable que almacena la corriente promedio del lado volante.
int i_prom;
```

```
// Variable para el nuevo rango de la corriente elec. medida.
int32_t current = 0;

//********** Variables para la lectura del encoder absoluto **********//
// Trama para solicitar la pos. angular del volante.
char A = 0x0001;
char B = 0x0003;
char C = 0x0000;
char D = 0x0000;
char E = 0x0000;
char F = 0x0001;
char G = 0x0084;
char H = 0x000A;
// Variable de 7 bytes para almacenar la respuesta del encoder absoluto.
uint8_t result[7];
// Variable que almacena el valor de la pos. del volante.
unsigned short int pos_abs;
// Variable para cambiar el rango de valores devueltos por el encoder.
int16_t volante;
// Pos. angular del subsistema lado volante
double Pos_SLV;

//********** Variable para medir el tiempo **********//
volatile uint32_t millis = 0;

//********** Constantes del controlador **********//
// Ganancia proporcional.
double Kp = 4;
// Ganancia integral.
double Ki = 33;
// Ganancia derivativa.
double Kd = 0;

//********** Variables externas del controlador **********//
double i_SLV; // Entrada del controlador
double Output;
int Out;
double i_SLR; // Referencia parcial del controlador
double out_put;

//********** Variables internas del controlador **********//
double error;
double e_RTN;
int error0;
double lastError;
double cumError;
double rateError;
unsigned long currentTime;
unsigned long previousTime;
double elapsedTime;

//********* Variables para la alerta de corriente ********//
unsigned long currentTime_2;
unsigned long previousTime_2;
```

```
unsigned long elapsedTime_2;
volatile bool band1_iSLV = 0;
volatile bool band2_iSLV = 0;
// Bandera de alerta (subsistema lado volante)
int wrg_iSLV = 0;

//***** La rutina de error que se llama si la biblioteca de
// controladores encuentra un error *****//
#ifdef DEBUG
void
_error_(char *pcFilename, uint32_t ui32Line)
{
}
#endif

//*************** Prototipos de funciones ***************//
// Prototipo para configurar el UART.
void ConfigureUART(void);
// Prototipo para configurar el bus CAN.
void CANIntHandler(void);
// Prototipo para el registro de interrupciones ADC.
void ObtenerMuestra(void);
// Prototipo para configurar el reloj PWM.
void PWM(void);
// Prototipo para configurar el PWM.
void PWMSignalConfigure(void);
// Prototipo que configura el temporizador de tick del sistema para
// interrumpir cada 1 milisegundo.
void SysTickbegin();
// Prototipo para el controlador de interrupciones que incrementa la
// variable millis en 1 cada vez que se llama.
void SycTickInt();
// Prototipo de retardo.
void Wait(uint32_t time);
// Prototipo con el algoritmo PID.
double compute_PID(double input);

int main(void)
{
    // Establece el objeto de mensaje CAN.
    tCANMsgObject msg_1; //-----> TX
    tCANMsgObject msg_3; //-----> RX

    //********** Para transmitir **********//
    // Los datos del mensaje tienen 4 bytes de longitud que se
    // pueden asignar como un int32.
    unsigned int msgData_1;
    // Puntero a msgData para acceder a bytes individuales.
    unsigned char *msgDataPtr_1 = (unsigned char *)&msgData_1;
    //************************************//

    // Arreglo de 8 bytes para los datos recibidos en el bus CAN.
    uint8_t msgData_3[8]; //-----> RX
```

```
//********** Reloj configurado a 50 MHz **********//
SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);
// Habilitando el apilamiento diferido para los controladores de
// interrupciones.
FPULazyStackingEnable();
//** Llamado del prototipo que configura el UART (puerto serial) **//
ConfigureUART();

//******** Configurando el puerto CAN ********//
// Habilitando el puerto B.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
// Configurando los pines GPIO para CAN.
GPIOPinConfigure(GPIO_PB4_CAN0RX);
GPIOPinConfigure(GPIO_PB5_CAN0TX);
// Habilitando los pines CAN: 4 --> RX y 5 --> TX.
GPIOPinTypeCAN(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5);
// Habilitando el CAN 0.
SysCtlPeripheralEnable(SYSCTL_PERIPH_CAN0);
// Inicializar el controlador CAN.
CANInit(CAN0_BASE);
// Configura la tasa de bits para el bus CAN.
CANBitRateSet(CAN0_BASE, SysCtlClockGet(), 1000000);
// Registro para las interrupciones de CAN.
CANIntRegister(CAN0_BASE, CANIntHandler);
// Habilitando las interrupiones del controlador CAN individual.
CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR |
CAN_INT_STATUS);
// Habilitando las interrupciones de CAN 0.
IntEnable(INT_CAN0);
// Habilitando el controlador CAN 0.
CANEnable(CAN0_BASE);

//********** Para transmitir **********//
// Inicializando el objeto de mensaje usado para enviar mensajes CAN.
msg_1.ui32MsgID = 0;
msg_1.ui32MsgIDMask = 0;
msg_1.ui32Flags = MSG_OBJ_TX_INT_ENABLE;
msg_1.ui32MsgLen = sizeof(msgDataPtr_1);
msg_1.pui8MsgData = msgDataPtr_1;
//************************************//

//************ Para recibir **********//
// Inicializando el objeto de mensaje usado para recibir mensajes CAN.
msg_3.ui32MsgID = 1;
msg_3.ui32MsgIDMask = 0;
msg_3.ui32Flags = MSG_OBJ_RX_INT_ENABLE | MSG_OBJ_USE_ID_FILTER;
msg_3.ui32MsgLen = 8; // Permite hasta 8 bytes.
// Configura el objeto de mensaje: Direc. base del controlador CAN,
// objeto de mensaje, puntero al objeto de mensaje, indica el tipo
// de mensaje.
CANMessageSet(CAN0_BASE, 2, &msg_3, MSG_OBJ_TYPE_RX);
//************************************//
```

```
//******** Configurando el temporizador ********//
// Habilitando el temporizador 0.
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
// Configurando el tipo de temporizador.
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
// Estableciendo la cantidad de ciclos de muestreo (frec. osc. / frec.
// de muestreo).
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()/FS);
// Habilitando el timer como un disparador.
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
// Habilitando el temporizador.
TimerEnable(TIMER0_BASE, TIMER_A);

//************ Configurando el ADC ************//
// Habilitando el ADC 0.
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
// Habilitando el puerto E.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
// Habilitando el pin ADC 2.
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);
// Configurando el pin de entrada--> corriente de manejo de 2 mA y una
// resistencia de PULL_DOWN.
GPIOPadConfigSet(GPIO_PORTE_BASE, GPIO_PIN_2, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPD);
// Configurando la secuencia de muestreo--> secuenciador 0 toma ocho
// muestras, con la mayor prioridad (0).
ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_TIMER, 0);
// Configurando cada paso de la secuencia--> secuenciador 0 iniciando
// desde 0 hasta 7 con el canal 1.
ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 5, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 6, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 7, ADC_CTL_CH1|ADC_CTL_IE|
ADC_CTL_END);
// Habilitando la secuencia.
ADCSequenceEnable(ADC0_BASE, 0);
// Habilitando la rutina de interrupciones.
ADCIntEnable(ADC0_BASE, 0);
// Registro para las interrupciones de ADC.
ADCIntRegister(ADC0_BASE, 0, ObtenerMuestra);
// Habilitando las interrupciones.
IntEnable(INT_ADC0SS0);
// Habilitando de forma general todas las interrupciones.
IntMasterEnable();

//***** Llamado de prototipos: PWM, tiempo *****//
PWM();
PWMSignalConfigure();
SysTickbegin();
```

```
// Configurando las salidas digitales para el led RGB.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_5|
GPIO_PIN_6);

//********** Configurando el UART para comunicarse con el
// encoder absoluto **********//
// Habilitando el UART5.
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
// Habilitando el puerto utilizado por el UART.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
// Configurando los pines GPIO para el UART.
GPIOPinConfigure(GPIO_PE4_U5RX);
GPIOPinConfigure(GPIO_PE5_U5TX);
GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
// Establece el UART--> direc. base, vel. del reloj suministrado,
// tasa de baudios, formato de datos.
// Formato de datos: cantidad de bits de datos | bits de parada |
// Paridad--> 8 bits de datos por byte | 1 bit de parada | sin bit
// de paridad.
UARTConfigSetExpClk(UART5_BASE, SysCtlClockGet(), 9600,
(UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

// Activando los pines DE y RE del driver MAX3485 para indicar
// el envio de una trama.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3|GPIO_PIN_4);
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3|GPIO_PIN_4, 24);
// Delay (10 ms)
SysCtlDelay(SysCtlClockGet() / 100 / 3);
// Solicitando la pos. del encoder absoluto.
UARTCharPut(UART5_BASE, A);
UARTCharPut(UART5_BASE, B);
UARTCharPut(UART5_BASE, C);
UARTCharPut(UART5_BASE, D);
UARTCharPut(UART5_BASE, E);
UARTCharPut(UART5_BASE, F);
UARTCharPut(UART5_BASE, G);
UARTCharPut(UART5_BASE, H);
// Delay (10 ms);
SysCtlDelay(SysCtlClockGet() / 100 / 3);

// Inicializando el error acumulado, el error anterior y el
// tiempo previo.
cumError = 0;
lastError = 0;
previousTime = (unsigned long)millis;

while(1) {
    // Desactivando los pines DE y RE para recibir la trama del
    // encoder absoluto.
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3|GPIO_PIN_4, 0);
    // Delay (2 ms)
    SysCtlDelay(SysCtlClockGet() / 500 / 3);
```

```
// Determinar si hay caracteres en el FIFO RX.
if(UARTCharsAvail(UART5_BASE)){
    result[0] = UARTCharGet(UART5_BASE);
    result[1] = UARTCharGet(UART5_BASE);
    result[2] = UARTCharGet(UART5_BASE);
    result[3] = UARTCharGet(UART5_BASE);
    result[4] = UARTCharGet(UART5_BASE);
    result[5] = UARTCharGet(UART5_BASE);
    result[6] = UARTCharGet(UART5_BASE);
    // Delay (1 ms)
    SysCtlDelay(SysCtlClockGet() / 1000 / 3);
}

// Obteniendo la pos. del encoder abs. mediante el byte 3 y
// 4 de la trama recibida.
pos_abs = (unsigned short int) (result[3] << 8 | result[4]);
volante = (((pos_abs*5000)/16383)-5000);
Pos_SLV = (double)volante;

//************ Para transmitir ************//
if (T_SLV){
    lectura_1 = pos_abs;
    dataTX0 = (char) 0x00FF & lectura_1;
    dataTX1 = (char) 0X00FF & (lectura_1 >> 8);
    //****** Datos a enviar por CAN ******//
    // Pos. del volante dividida en 2 bytes.
    msgDataPtr_1[0] = dataTX0;
    msgDataPtr_1[1] = dataTX1;
    // Alerta de consumo anormal de corriente en el motor
    // del sub. lado volante, activar modo seguro.
    msgDataPtr_1[2] = (unsigned char)wrng_LV;
    // Configura el objeto de mensaje: Direc. base del controlador
    // CAN, objeto de mensaje, puntero al objeto de mensaje,
    // indica el tipo de mensaje.
    // El mensaje se transmite inmediatamente.
    CANMessageSet(CAN0_BASE, 1, &msg_1, MSG_OBJ_TYPE_TX);
    // Delay (2 ms);
    SysCtlDelay(SysCtlClockGet() / 500 / 3);
    // Verificando si ocurrieron errores.
    if(errFlag_SLV) {
        UARTprintf("CAN Bus error\n");
    }
    T_SLV = 0;
    R_SLV = 1;
}
//****************************************//

//************* Para recibir *************//
if (R_SLV){
    if(rxFlag_SLV) {
        // Configurando el puntero dentro del objeto de mensaje.
        msg_3.pui8MsgData = msgData_3;
        // Obteniendo el mensaje. Se coloca un 0 al final, el
        // indicador que elimina las interrupciones no se
```

```
            // encuentra establecido. Controlador de interrupciones
            // activado.
            CANMessageGet(CAN0_BASE, 2, &msg_3, 0);
            // Restableciendo el indicador de mensaje pendiente.
            rxFlag_SLV = 0;
            // Alerta de mensajes perdidos.
            if(msg_3.ui32Flags & MSG_OBJ_DATA_LOST) {
                UARTprintf("CAN message loss detected\n");
            }
            //***** Obteniendo el mensaje recibido *****//
            // Corriente.
            dataRX0 = msgData_3[0];
            dataRX1 = msgData_3[1];
            // Alerta de corriente del lado rueda.
            dataRX2 = msgData_3[2];
            // Temperatura.
            dataRX3 = msgData_3[3];

            rearmado_2 = (unsigned short int)(dataRX1 << 8 | dataRX0);
            wrg_iSLR = (unsigned short int)dataRX2;
            Temp_SLR = (unsigned short int)dataRX3;
            // Deshabilitando RX y habilitando TX.
            R_SLV = 0;
            T_SLV = 1;
        }
    }
    //*************************************//

    Ref_Current = rearmado_2 - 2033;
    i_SLR = (double)Ref_Current;
    //***** Obteniendo la corriente elec. promedio del motor
    // (subsistema lado volante) *****//.
    i_prom = (s[0] + s[1] + s[2] + s[3] + s[4] + s[5] +
    s[6] + s[7])/8;
    current = i_prom - 2033;
    i_SLV = (double)current;

    // Algoritmo para detectar fallas en el consumo de corriente.
    // Voltaje = i_prom * (3.3 / 4095) ---> I = (Voltaje - 1.65) /
    // Sensibilidad ---> Sensibilidad = 90 mV/A.
    // En este caso i_prom = 484 = -14 A e i_prom = 3611 = 14 A.
    if(current < -1549 || current > 1578){
        if(!band2_iSLV){
            previousTime_2 = (unsigned long)millis;
            band2_iSLV = 1;
        }
        if(band2_iSLV){
            currentTime_2 = (unsigned long)millis;
            elapsedTime_2 = currentTime_2 - previousTime_2;
            if(elapsedTime_2 >= 2000){
                band1_iSLV = 1;
            }
        }
    }
```

```
else{
    band2_iSLV = 0;
}
// Obtener el tiempo actual.
currentTime = (unsigned long)millis;
// Calcular el tiempo transcurrido.
elapsedTime = (double)currentTime-(double)previousTime;

// Aplicando el control PID.
if(elapsedTime >= 5){
    Output = compute_PID(i_SLV);
    Out = (int)Output;
    //error0 = (int)e_RTN;
    // Imprimir la corriente del motor del SLV.
    UARTprintf("%d\n", current);
    // Imprimir la corriente del motor del SLR (ref. parcial).
    UARTprintf("%d\n", Ref_Current);
    // Imprimir la salida del control PID.
    UARTprintf("%d\n", Out);
    // Imprimir la compensacion de retorno a cero.
    //UARTprintf("%d\n", error0);
    // Almacenar el tiempo anterior.
    previousTime = currentTime;
}

if(!band1_iSLV){
    if(Output > 0){
        // Para que el motor gire a la derecha (cuando se tiene
        // un sistema de poleas y banda).
        C_Duty = 0;
        C_Duty2 = Output;
        // Estableciendo el ancho de pulso mediante el ciclo
        // de trabajo.
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, (uint32_t)C_Duty);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, (uint32_t)C_Duty2);
    }
    else if(Output == 0){
        // Para que el motor no gire.
        C_Duty = 0;
        C_Duty2 = 0;
        // Estableciendo el ancho de pulso mediante el ciclo
        // de trabajo.
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, (uint32_t)C_Duty);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, (uint32_t)C_Duty2);
    }
    else{
        // Para que el motor gire a la izquierda (cuando se tiene
        // un sistema de poleas y banda).
        C_Duty = ((-1)*Output);
        C_Duty2 = 0;
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, (uint32_t)C_Duty);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, (uint32_t)C_Duty2);
    }
}
```

```
        else{
            // Para que el motor no gire.
            C_Duty = 0;
            C_Duty2 = 0;
            // Estableciendo el ancho de pulso mediante el ciclo de
            // trabajo.
            PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, (uint32_t)C_Duty);
            PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, (uint32_t)C_Duty2);

            wrg_iSLV = 1;
        }

        // Alerta de temperatura del subsistema lado rueda.
        if(Temp_SLR >= 40){
            wrg_Temp = 1;
        }
        // Condicionales para activar las alertas de falla en el led.
        if(wrg_iSLV == 0){
            if(wrg_Temp == 1 || wrg_iSLR == 1){
                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_5|
                GPIO_PIN_6, 48);
            }
            else{
                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_5|
                GPIO_PIN_6, 64);
            }
        }
        else{
            GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4|GPIO_PIN_5|
            GPIO_PIN_6, 16);
        }

        // Activando los pines DE y RE del driver MAX3485 para
        // indicar el envio de una trama.
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3|GPIO_PIN_4, 24);
        // Delay (1 ms).
        SysCtlDelay(SysCtlClockGet() / 1000 / 3);
        // Solicitando la pos. del encoder absoluto.
        UARTCharPut(UART5_BASE, A);
        UARTCharPut(UART5_BASE, B);
        UARTCharPut(UART5_BASE, C);
        UARTCharPut(UART5_BASE, D);
        UARTCharPut(UART5_BASE, E);
        UARTCharPut(UART5_BASE, F);
        UARTCharPut(UART5_BASE, G);
        UARTCharPut(UART5_BASE, H);
        // Delay (5 ms).
        SysCtlDelay(SysCtlClockGet() / 200 / 3);
    }
}

//*****************************************************
// Configurando el UART y sus pines. Esto debe llamarse
// antes de UARTprintf().
```

```
//*****************************************************
void ConfigureUART(void)
{
    // Habilitando el puerto GPIO utilizado por el UART.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // Habilitando el UART0.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    // Configurando los pines para el UART.
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    // Usando el oscilador interno de 16 MHz como fuente de reloj UART.
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    // Estableciendo el UART a 115200 baudios y 16 MHz.
    UARTStdioConfig(0, 115200, 16000000);
}

//*****************************************************************
// Configurando el bus CAN. Comprueba la causa de las interrupciones
// y mantiene un recuento de todos los mensajes que han sido
// transmitidos y recibidos.
//*****************************************************************
void CANIntHandler(void){
    // Encontrando la causa de las interrupciones.
    uint32_t status;
    status = CANIntStatus(CAN0_BASE, CAN_INT_STS_CAUSE);
    // Detectando errores en el bus CAN.
    if(status == CAN_INT_INTID_STATUS){
       // Lee el estado del controlador. Este acto borra las interrup-
       // ciones. Si el nodo CAN no se encuentra conectado al bus con
       // otros dispositivos presentes, se producen errores y se indica
       // en el estado del controlador.
       status = CANStatusGet(CAN0_BASE, CAN_STS_CONTROL);
       // Bandera en ALTO para indicar error.
       errFlag_SLV = 1;
    }

    //********** Para transmitir **********//
    // Comprobar si la causa es el objeto de mensaje 1.
    else if(status == 1) {
        // Borrando las interrupciones del objeto de mensaje.
        CANIntClear(CAN0_BASE, 1);
        // Incrementando contador de mensajes transmitidos.
        MsgCount_1++;
        // Borrando cualquier indicador de error (mensaje enviado).
        errFlag_SLV = 0;
    }
    //************************************//

    //*********** Para recibir ***********//
    // Comprobar si la causa es el objeto de mensaje 2.
    else if(status == 2) {
        // Borrando las interrupciones del objeto de mensaje.
        CANIntClear(CAN0_BASE, 2);
```

```
        // Incrementando contador de mensajes recibidos.
        MsgCount_3++;
        // Bandera de mensaje recibido pendiente.
        rxFlag_SLV = 1;
        // Borrando cualquier indicador de error (mensaje recibido).
        errFlag_SLV = 0;
    }
    //*************************************//
    // Causa inesperada.
    else {
        UARTprintf("Unexpected CAN Bus interrupt\n");
    }
}

void ObtenerMuestra(void){
    //***** Limpiando las interrupciones del ADC *****//
    ADCIntClear(ADC0_BASE, 0);
    //***** Obteniendo la muestra en un puntero *****//
    ADCSequenceDataGet(ADC0_BASE, 0, s);
}

void PWM(void){
    // Configurando el reloj PWM a 50 MHz, se aplica un divisor al
    // reloj establecido en el procesador principal.
    SysCtlPWMClockSet(SYSCTL_PWMDIV_1);
    // Habilitando el PWM 1.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    // Habilitando los pines del puerto F.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    // Habilitando el pin 3 (PF3) como salida PWM, led verde.
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
    // Habilitando el pin 2 (PF2) como salida PWM, led azul.
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
    // Habilitando el pin 3 (PF3) como un pin que se conecta a
    // la salida 7 del PWM.
    GPIOPinConfigure(GPIO_PF3_M1PWM7);
    // Habilitando el pin 2 (PF2) como un pin que se conecta a
    // la salida 6 del PWM.
    GPIOPinConfigure(GPIO_PF2_M1PWM6);
}

void PWMSignalConfigure(void){
    //***** Estableciendo el periodo --> T_ciclos.
    // T_ciclos=(50,000,000/1)/10,000=5,000 *****//
    T_ciclos = (SysCtlClockGet()/1)/F_Hz;
    // Configurando el contador PWM.
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_UP_DOWN |
    PWM_GEN_MODE_NO_SYNC);
    // Configurando el PWM con el periodo.
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, T_ciclos);
    // Habilitando el generador 3 del PWM con el contador.
    PWMGenEnable(PWM1_BASE, PWM_GEN_3);
    // Habilitando las salidas de PWM.
    PWMOutputState(PWM1_BASE, PWM_OUT_7_BIT | PWM_OUT_6_BIT, true);
```

```
}
// El reloj del CPU es de 50 MHz. Por lo tanto, el valor que se incluye
// en SysTickPeriodSet es: CPU_CLOCK / (1 / time_unit). Donde time_unit
// es el tiempo en segundos. En este caso 1 ms = 0.001 s.
// 50 MHz / (1 / 0.001 s) = 50,000.
void SysTickbegin(){
    // Estableciendo el periodo del contador SysTick.
    SysTickPeriodSet(50000);
    // Registro del controlador de interrupciones de SysTick.
    // SycTickInt es llamada en las interrupciones.
    SysTickIntRegister(SycTickInt);
    // Habilita las interrupciones.
    SysTickIntEnable();
    // Habilita el contador SysTick.
    SysTickEnable();
}

void SycTickInt(){
    millis++;
}

// Crea retardos en milisegundos.
void Wait(uint32_t time){
   uint32_t temp = millis;
   while( (millis-temp) < time ){

   }
}

// Algoritmo del controlador PID.
double compute_PID(double input){
    // Compensacion de retorno a cero.
    e_RTN = (0-Pos_SLV)/4;
    // Error global (i_REF-i_SLV).
    error = (i_SLR+e_RTN)-i_SLV;
    // Frecuencia de control: 0.005 s.
    // Calculando la integral del error.
    cumError = cumError+(error*0.005);
    // Calculando la derivada del error.
    rateError = (error-lastError)/0.005;
    // Calculando la salida del control PID.
    out_put = ((Kp*error)+(Ki*cumError)+(Kd*rateError));
    // Normalizando Uav (motor alimentado con 12 V).
    out_put = out_put/3; // out_put = (out_put*4)/12;

    //********** Saturador **********//
    if(out_put>4999){
        out_put = 4999;
    }
    if(out_put<-4999){
        out_put = -4999;
    }
    //******************************//
    // Almacenando el error anterior.
```

```
    lastError = error;
    // Devolviendo la salida del control PID.
    return(out_put);
}
```

A.2. Código del subsistema lado rueda

```
//********** Bibliotecas **********//
#include <stdint.h>
#include <float.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_can.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/adc.h"
#include "driverlib/pwm.h"
#include "driverlib/qei.h"
#include "driverlib/udma.h"
#include "driverlib/systick.h"
#include "driverlib/can.h"
#include "utils/uartstdio.h"

//***************** Variables declaradas *****************//
//********** Variables del bus CAN bidireccional *********//
//********************** Para recibir ********************//
// Contador de interrupciones, igual a cantidad de mensajes recibidos.
volatile uint32_t MsgCount_2 = 0;
// Bandera para que el manejador de interrupciones indique si un
// mensaje es recibido.
volatile bool rxFlag_SLR = 0;
// Variable encargada de almacenar el valor rearmado de la pos. angular
// del subsistema lado volante, recibido en 2 bytes.
unsigned short int rearmado_1 = 16383;
// Variable que almacena el byte menos significativo del valor de
// pos. angular.
```

```
unsigned char dataRX0;
// Variable que almacena el byte mas significativo del valor de
// pos. angular.
unsigned char dataRX1;
// Variable para el nuevo rango de la pos. medida con el encoder absoluto.
int16_t volante;
// Variable que almacena la alerta de corriente del lado volante.
unsigned char dataRX2;
// Variable que procesa la alerta de corriente recibida.
int16_t wrg_iSLV = 0;
// Bandera para indicar que el CAN debe o no recibir datos. Inicialmente
// se coloca en alto para obtener los datos provenientes del subsistema
// lado volante.
volatile bool R_SLR = 1;

//******************** Para transmitir ******************//
// Contador de interrupciones, igual a cantidad de mensajes transmitidos.
volatile uint32_t MsgCount_4 = 0;
// Bandera de error del canal CAN (bus desconectado).
volatile bool errFlag_SLR = 0;
// Variable que almacena el valor de la corriente elec. consumida por el
// motor del subsistema lado rueda, es dividido y transmitido por el bus
// CAN como 2 bytes.
unsigned short int lectura_4 = 0;
// Variable que almacena el byte menos significativo de lectura_4.
unsigned char dataTX0;
// Variable que almacena el byte mas significativo de lectura_4.
unsigned char dataTX1;
// Bandera para indicar que el CAN debe o no transmitir datos.
// Inicialmente no se envia los datos del subsistema lado rueda.
volatile bool T_SLR = 0;

//******** Variables para la etapa de potencia (driver BTS7960) ********//
// Frecuencia para el periodo de PWM de 10 KHz.
#define F_Hz 10000
// Variable del periodo PWM.
uint32_t T_ciclos;
// Variables del ciclo de trabajo.
double C_Duty;
double C_Duty2;

//****** Variables para la lectura ADC de los sensores: ACS711EX y LM35
// Frecuencia de muestreo de 40 KHz.
#define FS 40000
// Variable que obtiene las muestras del ADC.
uint32_t s[8];
// Variable que almacena la corriente del motor del subsistema lado rueda.
int i_SLR;
// Variable que almacena el promedio de las muestras tomadas del sensor
// de temperatura LM35.
int temp;
// Temperatura en grados Celsius.
int Temp_SLR;
// Variable de alerta de temperatura en el motor del subs. lado rueda.
```

```
int16_t wrg_Temp = 0;

//****** Variables para la lectura del encoder incremental ******//
// Valor anterior del encoder incremental.
int32_t posicion = 0;
// Variable considerada como entrada del control PID.
int32_t position = 0;
// Cantidad de pulsos seleccionada para el encoder. Superando este
// valor se tiene un reinicio.
uint32_t NumPulsos = 20000;
// Valor actual del encoder incremental.
int32_t Lec = 0;
// Diferencia entre el valor actual y el valor anterior.
int32_t dif = 0;

//********** Variable para medir el tiempo **********//
volatile uint32_t millis = 0;

//********** Constantes del controlador **********//
// Ganancia proporcional.
double Kp = 250;
// Ganancia integral.
double Ki = 400;
// Ganancia derivativa.
double Kd = 1;

//********** Variables externas del controlador **********//
double Pos_SLR; // Pos. medida en el subsistema lado rueda
double Output;
int Out;
double Pos_SLV; // Pos. angular de referencia (volante)
double out_put;
volatile uint32_t Count_2 = 0;

//********** Variables internas del controlador **********//
double error;
int e_imp;
double lastError;
double cumError;
double rateError;
unsigned long currentTime;
unsigned long previousTime;
double elapsedTime;

//********* Variables para la alerta de corriente ********//
unsigned long currentTime_3;
unsigned long previousTime_3;
unsigned long elapsedTime_3;
volatile bool band_iSLR = 0;
// Bandera de alerta (subsistema lado rueda)
int wrg_iSLR = 0;

//********* Banderas adicionales para las fallas ********//
// Bandera para indicar el estado de funcionamiento del sistema.
```

```
int16_t Func_norm = 1;
// Bandera para indicar si existen fallas en el subsistema lado rueda.
int16_t Falla_SLR = 0;

//***** La rutina de error que se llama si la biblioteca de
// controladores encuentra un error *****//
#ifdef DEBUG
void
_error_(char *pcFilename, uint32_t ui32Line)
{
}
#endif

//*************** Prototipos de funciones ***************//
// Prototipo para configurar el UART.
void ConfigureUART(void);
// Prototipo para configurar el bus CAN.
void CANIntHandler(void);
// Prototipo para configurar el reloj PWM.
void PWM(void);
// Prototipo para configurar el PWM.
void PWMSignalConfigure(void);
// Prototipo para configurar la lectura del encoder incremental.
void configurarEncoder(void);
// Prototipo para obtener los datos del encoder incremental.
void datosE();
// Prototipo que configura el temporizador de tick del sistema para
// interrumpir cada 1 milisegundo.
void SysTickbegin();
// Prototipo para el controlador de interrupciones que incrementa la
// variable millis en 1 cada vez que se llama.
void SycTickInt();
// Prototipo de retardo.
void Wait(uint32_t time);
// Prototipo con el algoritmo PID.
double compute_PID(double input);
// Prototipo para el registro de interrupciones ADC.
void ObtenerMuestra(void);

int main(void)
{
    // Establece el objeto de mensaje CAN.
    tCANMsgObject msg_2; //-----> RX
    tCANMsgObject msg_4; //-----> TX
    // Arreglo de 8 bytes para los datos recibidos en el bus CAN.
    uint8_t msgData_2[8]; //-----> RX

    //********** Para transmitir **********//
    // Los datos del mensaje tienen 4 bytes de longitud que se
    // pueden asignar como un int32.
    unsigned int msgData_4;
    // Puntero a msgData para acceder a bytes individuales.
    unsigned char *msgDataPtr_4 = (unsigned char *)&msgData_4;
    //*********************************//
```

```
//********** Reloj configurado a 50 MHz **********//
SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);
// Habilitando el apilamiento diferido para los controladores de
// interrupciones.
FPULazyStackingEnable();
//** Llamado del prototipo que configura el UART (puerto serial) **//
ConfigureUART();

//******** Configurando el puerto CAN ********//
// Habilitando el puerto B.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
// Configurando los pines GPIO para CAN.
GPIOPinConfigure(GPIO_PB4_CAN0RX);
GPIOPinConfigure(GPIO_PB5_CAN0TX);
// Habilitando los pines CAN: 4 --> RX y 5 --> TX.
GPIOPinTypeCAN(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5);
// Habilitando el CAN 0.
SysCtlPeripheralEnable(SYSCTL_PERIPH_CAN0);
// Inicializar el controlador CAN.
CANInit(CAN0_BASE);
// Configura la tasa de bits para el bus CAN.
CANBitRateSet(CAN0_BASE, SysCtlClockGet(), 1000000);
// Registro para las interrupciones de CAN.
CANIntRegister(CAN0_BASE, CANIntHandler);
// Habilitando las interrupiones del controlador CAN individual.
CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR |
CAN_INT_STATUS);
// Habilitando las interrupciones de CAN 0.
IntEnable(INT_CAN0);
// Habilitando el controlador CAN 0.
CANEnable(CAN0_BASE);

//************ Para recibir ***********//
// Inicializando el objeto de mensaje usado para recibir mensajes CAN.
msg_2.ui32MsgID = 0;
msg_2.ui32MsgIDMask = 0;
msg_2.ui32Flags = MSG_OBJ_RX_INT_ENABLE | MSG_OBJ_USE_ID_FILTER;
msg_2.ui32MsgLen = 8; // Permite hasta 8 bytes.
// Configura el objeto de mensaje: Direc. base del controlador CAN,
// objeto de mensaje, puntero al objeto de mensaje, indica el tipo
// de mensaje.
CANMessageSet(CAN0_BASE, 1, &msg_2, MSG_OBJ_TYPE_RX);
//************************************//
//********** Para transmitir **********//
// Inicializando el objeto de mensaje usado para enviar mensajes CAN.
msg_4.ui32MsgID = 1;
msg_4.ui32MsgIDMask = 0;
msg_4.ui32Flags = MSG_OBJ_TX_INT_ENABLE;
msg_4.ui32MsgLen = sizeof(msgDataPtr_4);
msg_4.pui8MsgData = msgDataPtr_4;
//************************************//
//***** Llamado de prototipos para: PWM, encoder inc., tiempo *****//
PWM();
```

```
PWMSignalConfigure();
configurarEncoder();
SysTickbegin();

//******** Configurando el temporizador ********//
// Habilitando el temporizador 0.
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
// Configurando el tipo de temporizador.
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
// Estableciendo la cantidad de ciclos de muestreo (frec. osc. / frec.
// de muestreo).
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()/FS);
// Habilitando el timer como un disparador.
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
// Habilitando el temporizador.
TimerEnable(TIMER0_BASE, TIMER_A);

//************ Configurando el ADC ************//
// Habilitando el ADC 0.
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
// Habilitando el puerto E.
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
// Habilitando el pin ADC 2 y ADC 3.
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3);
// Configurando los pines de entrada--> corriente de manejo de 2 mA
// y una resistencia de PULL_DOWN.
GPIOPadConfigSet(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3,
GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
// Configurando la secuencia de muestreo--> secuenciador 0 toma ocho
// muestras, con la mayor prioridad (0).
ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_TIMER, 0);
// Configurando cada paso de la secuencia--> secuenciador 0 iniciando
// desde 0 hasta 3 con el canal 1 (pin PE2) y de 4 hasta 7 con el
// canal 0 (pin PE3).
ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH0);
ADCSequenceStepConfigure(ADC0_BASE, 0, 5, ADC_CTL_CH0);
ADCSequenceStepConfigure(ADC0_BASE, 0, 6, ADC_CTL_CH0);
ADCSequenceStepConfigure(ADC0_BASE, 0, 7, ADC_CTL_CH0|ADC_CTL_IE|
ADC_CTL_END);
// Habilitando la secuencia.
ADCSequenceEnable(ADC0_BASE, 0);
// Habilitando la rutina de interrupciones.
ADCIntEnable(ADC0_BASE, 0);
// Registro para las interrupciones de ADC.
ADCIntRegister(ADC0_BASE, 0, ObtenerMuestra);
// Habilitando las interrupciones.
IntEnable(INT_ADC0SS0);
// Habilitando de forma general todas las interrupciones.
IntMasterEnable();
```

```
// Inicializando el error acumulado, el error anterior y el
// tiempo previo.
cumError = 0;
lastError = 0;
previousTime = (unsigned long)millis;

while(1) {
    //************* Para recibir *************//
    if (R_SLR){
        if(rxFlag_SLR) {
            // Configurando el puntero dentro del objeto de mensaje.
            msg_2.pui8MsgData = msgData_2;
            // Obteniendo el mensaje. Se coloca un 0 al final, el
            // indicador que elimina las interrupciones no se
            // encuentra establecido. Controlador de interrupciones
            // activado.
            CANMessageGet(CAN0_BASE, 1, &msg_2, 0);
            // Restableciendo el indicador de mensaje pendiente.
            rxFlag_SLR = 0;
            // Alerta de mensajes perdidos.
            if(msg_2.ui32Flags & MSG_OBJ_DATA_LOST) {
                UARTprintf("CAN message loss detected\n");
            }
            //***** Obteniendo el mensaje recibido *****//
            // Pos. del volante.
            dataRX0 = msgData_2[0];
            dataRX1 = msgData_2[1];
            // Alerta de corriente del subsistema lado volante.
            dataRX2 = msgData_2[2];

            rearmado_1 = (unsigned short int)(dataRX1 << 8 | dataRX0);
            wrg_iSLV = (unsigned short int)dataRX2;
            // Deshabilitando RX y habilitando TX.
            R_SLR = 0;
            T_SLR = 1;
        }
    }
    //*************************************//

    volante = (((rearmado_1*5000)/16383)-5000);
    Pos_SLV = (double)volante;
    // Obteniendo la pos. angular medida con el encoder incremental.
    datosE();
    // Cambiando el sentido de conteo del encoder.
    position = ((-1)*posicion/2);
    Pos_SLR = (double)position;

    // Obtener el tiempo actual.
    currentTime = (unsigned long)millis;
    // Calcular el tiempo transcurrido.
    elapsedTime = (double)currentTime-(double)previousTime;

    // Aplicando el control PID.
    if(elapsedTime >= 5){
```

```
        Output = compute_PID(Pos_SLR);
        Out = (int)Output;
        e_imp = (int)error;
        Count_2++;
        if(Count_2 >= 8){
            Count_2 = 0;
            // Imprimir pos. angular medida en el SLR.
            //UARTprintf("%d\n", position);
            // Imprimir error.
            UARTprintf("%d\n", e_imp);
            // Imprimir pos. angular del volante (referencia).
            UARTprintf("%d\n", volante);
            // Imprimir la salida del control PID.
            //UARTprintf("%d\n", Out);
            // Imprimir la temperatura medida.
            UARTprintf("%d\n", Temp_SLR);
            // Indicar el estado de funcionamiento del sistema.
            UARTprintf("%d\n", Func_norm);
            // Indicar el estado de func. del subsistema lado rueda.
            UARTprintf("%d\n", Falla_SLR);
            // Imprimir el estado de func. del subs. lado volante.
            UARTprintf("%d\n", wrg_iSLV);
            // Alerta de corriente elec. del lado rueda.
            UARTprintf("%d\n", wrg_iSLR);
            // Alerta de temperatura.
            UARTprintf("%d\n", wrg_Temp);
        }
        // Almacenar el tiempo anterior.
        previousTime = currentTime;
    }

    if(Output > 0){
        // Para que el motor gire a la derecha (cuando se tiene
        // un sistema de poleas y banda).
        C_Duty = Output;
        C_Duty2 = 0;
        // Estableciendo el ancho de pulso mediante el ciclo
        // de trabajo.
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, (uint32_t)C_Duty);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, (uint32_t)C_Duty2);
    }
    else if(Output == 0){
        // Para que el motor no gire.
        C_Duty = 0;
        C_Duty2 = 0;
        // Estableciendo el ancho de pulso mediante el ciclo
        // de trabajo.
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, (uint32_t)C_Duty);
        PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, (uint32_t)C_Duty2);
    }
    else{
        // Para que el motor gire a la izquierda (cuando se tiene
        // un sistema de poleas y banda).
        C_Duty = 0;
```

```
    C_Duty2 = ((-1)*Output);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, (uint32_t)C_Duty);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, (uint32_t)C_Duty2);
}

//***** Obteniendo la corriente elec. promedio del motor
// (subsistema lado rueda) *****//.
i_SLR = (s[0] + s[1] + s[2] + s[3])/4;
// Algoritmo para detectar fallas en el consumo de corriente.
// Voltaje = i_SLR * (3.3 / 4095) ---> I = (Voltaje - 1.65) /
// Sensibilidad ---> Sensibilidad = 90 mV/A.
// En este caso i_SLR = 484 = -14 A e i_SLR = 3611 = 14 A.
if(i_SLR < 484 || i_SLR > 3611){
    if(!band_iSLR){
        previousTime_3 = (unsigned long)millis;
        band_iSLR = 1;
    }
    if(band_iSLR){
        currentTime_3 = (unsigned long)millis;
        elapsedTime_3 = currentTime_3 - previousTime_3;
        if(elapsedTime_3 >= 1000){
            wrg_iSLR = 1;
        }
    }
}
else{
    band_iSLR = 0;
}

// Obteniendo el valor promedio de la temperatura.
temp = (s[4] + s[5] + s[6] + s[7])/4;
// Convirtiendo a grados Celsius.
Temp_SLR = ((temp*330)/4095);
// Determinando las alertas en el lado rueda y el
// funcionamiento del sistema.
if(Temp_SLR >= 40){
    wrg_Temp = 1;
}
if(wrg_iSLR == 1 || wrg_Temp == 1){
    Falla_SLR = 1;
}
if(wrg_iSLV == 1 || Falla_SLR == 1){
    Func_norm = 0;
}
//************ Para transmitir ************//
if (T_SLR){
    lectura_4 = i_SLR;
    dataTX0 = (char) 0x00FF & lectura_4;
    dataTX1 = (char) 0X00FF & (lectura_4 >> 8);
    //****** Datos a enviar por CAN ******//
    // Corriente del motor, dividida en 2 bytes.
    msgDataPtr_4[0] = dataTX0;
    msgDataPtr_4[1] = dataTX1;
    // Alerta de consumo excesivo de corriente en el motor
```

```
            // del sub. lado rueda.
            msgDataPtr_4[2] = (unsigned char)wrg_iSLR;
            // Temperatura del motor.
            msgDataPtr_4[3] = (unsigned char)Temp_SLR;
            // Configura el objeto de mensaje: Direc. base del controlador
            // CAN, objeto de mensaje, puntero al objeto de mensaje,
            // indica el tipo de mensaje.
            // El mensaje se transmite inmediatamente.
            CANMessageSet(CAN0_BASE, 2, &msg_4, MSG_OBJ_TYPE_TX);
            // Delay (2 ms);
            SysCtlDelay(SysCtlClockGet() / 500 / 3);
            // Verificando si ocurrieron errores.
            if(errFlag_SLR) {
                UARTprintf("CAN Bus error\n");
            }
            T_SLR = 0;
            R_SLR = 1;
        }
        //*************************************//
    }
}

//*****************************************************
// Configurando el UART y sus pines. Esto debe llamarse
// antes de UARTprintf().
//*****************************************************
void ConfigureUART(void)
{
    // Habilitando el puerto GPIO utilizado por el UART.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // Habilitando el UART0.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    // Configurando los pines para el UART.
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    // Usando el oscilador interno de 16 MHz como fuente de reloj UART.
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
    // Estableciendo el UART a 115200 baudios y 16 MHz.
    UARTStdioConfig(0, 115200, 16000000);
}

//*************************************************************
// Configurando el bus CAN. Comprueba la causa de las interrupciones
// y mantiene un recuento de todos los mensajes que han sido
// transmitidos y recibidos.
//*************************************************************
void CANIntHandler(void){
    // Encontrando la causa de las interrupciones.
    uint32_t status;
    status = CANIntStatus(CAN0_BASE, CAN_INT_STS_CAUSE);
    // Detectando errores en el bus CAN.
    if(status == CAN_INT_INTID_STATUS){
       // Lee el estado del controlador. Este acto borra las interrup-
```

```
        // ciones. Si el nodo CAN no se encuentra conectado al bus con
        // otrs dispositivos presentes, se producen errores y se indica
        // en el estado del controlador.
        status = CANStatusGet(CAN0_BASE, CAN_STS_CONTROL);
        // Bandera en ALTO para indicar error.
        errFlag_SLR = 1;
    }

    //*********** Para recibir **********//
    // Comprobar si la causa es el objeto de mensaje 1.
    else if(status == 1) {
        // Borrando las interrupciones del objeto de mensaje.
        CANIntClear(CAN0_BASE, 1);
        // Incrementando contador de mensajes recibidos.
        MsgCount_2++;
        // Bandera de mensaje recibido pendiente.
        rxFlag_SLR = 1;
        // Borrando cualquier indicador de error (mensaje recibido).
        errFlag_SLR = 0;
    }
    //************************************//

    //********** Para transmitir **********//
    // Comprobar si la causa es el objeto de mensaje 2.
    else if(status == 2) {
        // Borrando las interrupciones del objeto de mensaje.
        CANIntClear(CAN0_BASE, 2);
        // Incrementando contador de mensajes transmitidos.
        MsgCount_4++;
        // Borrando cualquier indicador de error (mensaje enviado).
        errFlag_SLR = 0;
    }
    //************************************//

    // Causa inesperada.
    else {
        UARTprintf("Unexpected CAN Bus interrupt\n");
    }
}

void PWM(void){
    // Configurando el reloj PWM a 50 MHz, se aplica un divisor al
    // reloj establecido en el procesador principal.
    SysCtlPWMClockSet(SYSCTL_PWMDIV_1);
    // Habilitando el PWM 1.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    // Habilitando los pines del puerto F.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    // Habilitando el pin 3 (PF3) como salida PWM, led verde.
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
    // Habilitando el pin 2 (PF2) como salida PWM, led azul.
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
    // Habilitando el pin 3 (PF3) como un pin que se conecta a
    // la salida 7 del PWM.
```

```
    GPIOPinConfigure(GPIO_PF3_M1PWM7);
    // Habilitando el pin 2 (PF2) como un pin que se conecta a
    // la salida 6 del PWM.
    GPIOPinConfigure(GPIO_PF2_M1PWM6);
}

void PWMSignalConfigure(void){
    //***** Estableciendo el periodo --> T_ciclos.
    // T_ciclos=(50,000,000/1)/10,000=5,000 *****//
    T_ciclos = (SysCtlClockGet()/1)/F_Hz;
    // Configurando el contador PWM.
    PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_UP_DOWN |
    PWM_GEN_MODE_NO_SYNC);
    // Configurando el PWM con el periodo.
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, T_ciclos);
    // Habilitando el generador 3 del PWM con el contador.
    PWMGenEnable(PWM1_BASE, PWM_GEN_3);
    // Habilitando las salidas de PWM.
    PWMOutputState(PWM1_BASE, PWM_OUT_7_BIT | PWM_OUT_6_BIT, true);
}

void configurarEncoder(){
    // Habilitando los pines del puerto C.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    // Habilitando el QEI 1.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI1);
    // Configurando el pin 4 (PC4), pin 5 (PC5) y pin 6 (PC6) como
    // pines de lectura del encoder incremental.
    GPIOPinTypeQEI(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6);
    // Habilitando el pin PC4 como el encargado de detectar el pulso
    // de referencia del encoder.
    GPIOPinConfigure(GPIO_PC4_IDX1);
    // Habilitando el pin PC5 como el encargado de detectar el pulso
    // de la fase A del encoder.
    GPIOPinConfigure(GPIO_PC5_PHA1);
    // Habilitando el pin PC6 como el encargado de detectar el pulso
    // de la fase B del encoder.
    GPIOPinConfigure(GPIO_PC6_PHB1);
    // Esperando a que QEI 1 se encuentre listo.
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_QEI1)){

    }
    // Configurando el codificador con la lectura de los dos canales
    // (A y B), sin reinicio, en cuadratura y sin intercambiarlos.
    QEIConfigure(QEI1_BASE, (QEI_CONFIG_CAPTURE_A_B |
    QEI_CONFIG_NO_RESET | QEI_CONFIG_QUADRATURE |
    QEI_CONFIG_NO_SWAP), (NumPulsos-1));
    // Configurando el filtro de entrada.
    QEIFilterConfigure(QEI1_BASE, QEI_FILTCNT_2);
    // Habilitando el filtro de entrada.
    QEIFilterEnable(QEI1_BASE);
    // Habilitando el codificador de cuadratura.
    QEIEnable(QEI1_BASE);
    // Configurando la captura de velocidad. Periodo = 10 ms, cantidad
```

```
    // de ciclos = Periodo / (1/50 MHz) = 500,000.
    QEIVelocityConfigure(QEI1_BASE, QEI_VELDIV_1, 500000);
    // Habilitando la captura de velocidad.
    QEIVelocityEnable(QEI1_BASE);
}

void datosE(){
    // Obteniendo la pos. angular del eje.
    Lec = QEIPositionGet(QEI1_BASE);
    dif = Lec - posicion;
    if(dif > 19000){
        posicion = Lec - 20000;
    }
    else{
        posicion = Lec;
    }
}

// El reloj del CPU es de 50 MHz. Por lo tanto, el valor que se incluye
// en SysTickPeriodSet es: CPU_CLOCK / (1 / time_unit). Donde time_unit
// es el tiempo en segundos. En este caso 1 ms = 0.001 s.
// 50 MHz / (1 / 0.001 s) = 50,000.
void SysTickbegin(){
    // Estableciendo el periodo del contador SysTick.
    SysTickPeriodSet(50000);
    // Registro del controlador de interrupciones de SysTick.
    // SycTickInt es llamada en las interrupciones.
    SysTickIntRegister(SycTickInt);
    // Habilita las interrupciones.
    SysTickIntEnable();
    // Habilita el contador SysTick.
    SysTickEnable();
}

void SycTickInt(){
    millis++;
}
// Crea retardos en milisegundos.
void Wait(uint32_t time){
   uint32_t temp = millis;
   while( (millis-temp) < time ){

   }
}

// Algoritmo del controlador PID.
double compute_PID(double input){
    // Error entre el valor de referencia y el valor medido con el
    // encoder incremental.
    error = Pos_SLV-Pos_SLR;
    // Frecuencia de control: 0.005 s.
    // Calculando la integral del error.
    cumError = cumError+(error*0.005);
    // Calculando la derivada del error.
```

```
    rateError = (error-lastError)/0.005;
    // Calculando la salida del control PID.
    out_put = ((Kp*error)+(Ki*cumError)+(Kd*rateError));
    // Normalizando Uav (motor alimentado con 12 V).
    out_put = out_put/12;

    //********** Saturador **********//
    if(out_put>4999){
        out_put = 4999;
    }
    if(out_put<-4999){
        out_put = -4999;
    }
    //******************************//

    // Almacenando el error anterior.
    lastError = error;
    // Devolviendo la salida del control PID.
    return(out_put);
}

void ObtenerMuestra(void){
    //***** Limpiando las interrupciones del ADC *****//
    ADCIntClear(ADC0_BASE, 0);
    //***** Obteniendo la muestra en un puntero *****//
    ADCSequenceDataGet(ADC0_BASE, 0, s);
}
```