ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Grado

DISEÑO E IMPLEMENTACIÓN DE SISTEMA DE CONTROL DE VELOCIDAD PARA VEHÍCULO ELÉCTRICO EXPERIMENTAL

(Design and Implementation of Speed Control System for Experimental Electric Vehicle)

Para acceder al Título de

GRADUADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Paulino San Miguel Herrera

09 - 2017

(1) El presente trabajo se enmarca dentro de un proyecto de investigación por lo que la organización del mismo puede diferir de la de un proyecto industrial.

(2) Redactado con el procesador de textos Lyx v.2.2.3.

Índice general

1 INTRODUCCIÓN. 1

1.1 VEHÍCULOS ELÉCTRICOS. 2
1.1.1 Breve historia. 2
1.1.2 Automóviles eléctricos. 5
1.1.3 Partes principales. 7

1.2 VEHÍCULOS AUTÓNOMOS. 11

2 VISIÓN GLOBAL. 12

2.1 OBJETIVO DEL DEPARTAMENTO. 13

2.2 OBJETIVO DEL TRABAJO. 14

2.3 REQUISITOS. 15

2.4 PARTE FÍSICA. 15

2.5 PARTE LÓGICA. 17

2.6 COMUNICACIÓN. 17
2.6.1 Sistema de control inteligente. 17
2.6.2 Protocolos de comunicación. 18

2.7 MECANISMO DE CONTROL. 19

3 SUBSISTEMA DE ACELERACIÓN. 27

3.1 ELEMENTOS PRINCIPALES. 27
3.1.1 Motor. 27
3.1.2 Alimentación. 28
3.1.3 Controladora. 29

3.2 AJUSTES INICIALES. 30

3.2.1 Posibilidad de alternar entre modo de conducción autónomo y manual. 30
3.2.2 Actuar sobre los pines de aceleración de la controladora. 32
3.2.3 Obtener la velocidad del vehículo experimental. 33
3.2.4 Parámetros de la controladora. 35

3.3 IMPLEMENTACIÓN. 39
3.3.1 Montaje del circuito. 40
3.3.2 Parámetros del controlador. 41

4 SUBSISTEMA DE FRENADO. 43

4.1 INTRODUCCIÓN. 44

4.2 MÉTODOS DE FRENADO. 46
4.2.1 Freno regenerativo. 46
4.2.2 Frenos en conducción manual. 47
4.2.3 Frenos en conducción autónoma. 48

4.3 CENTRAL HIDRÁULICA COMPACTA. 49
4.3.1 Partes principales. 49
4.3.2 Esquema de conexión. 51

4.4 AJUSTES INICIALES. 52
4.4.1 Cableado. 52
4.4.2 Alimentación. 53
4.4.3 Comprobación de funcionamiento a tensión nominal. 54
4.4.4 Variación de presión en el circuito de freno. 55

4.5 IMPLEMENTACIÓN. 60

5 SISTEMA DE CONTROL DE VELOCIDAD FINAL. 61

5.1 HARDWARE. 61

5.2 SOFTWARE 62

6 CONTINUACIÓN DE LA LÍNEA DE INVESTIGACIÓN. 63

6.1 MEJORA DEL SISTEMA DE CONTROL DE VELOCIDAD. 63

6.2 TABLERO DE ABORDO. 64
6.2.1 Software empleado. 65

6.2.2 Creación de objetos tridimensionales con BLENDER. 66
6.2.3 Tablero en processing. 69
6.2.4 Establecer comunicación con Arduino. 70
6.2.5 Probando el código. 75
6.2.6 Mejoras del panel. 78

ANEXO I: CÁLCULO DE FILTROS. 79

FILTRADO DE LA SALIDA PWM DE ARDUINO. 82

FILTRADO SEÑAL SENSOR HALL. 84

ANEXO II: COMUNICACIÓN ARDUINO-MATLAB. 87

PRUEBA COMUNICACIÓN SERIE. 87
Código MATLAB. 88
Código ARDUINO. 90
Resultados. 92

PRUEBA COMUNICACIÓN ETHERNET. 92
Código MATLAB. 92
Código ARDUINO. 95
Resultados. 97

ANEXO III: CÓDIGOS PARCIALES DEL CONTROL DE VELOCIDAD. 98

CÓDIGO DEL PID. 98

CÓDIGO DE CONTEO DE ACTIVACIONES DEL SENSOR HALL. 100

CÓDIGO DEL SUBSISTEMA DE ACELERACIÓN. 103

CÓDIGO DEL SUBSISTEMA DE FRENADO. 106

ANEXO IV: CÓDIGO DEL CONTROL DE VELOCIDAD FINAL. 108

ANEXO V: HOJAS DE CARACTERÍSTICAS. 112

ANEXO VI: COSTE DEL SISTEMA. 113

BIBLIOGRAFÍA 114

Índice de figuras

1.1 Vehículo de Verbiest 2
1.2 Vehículo de Cugnot 2
1.3 La Jamais Contente 3
1.4 Propulsión diesel-eléctrica en un submarino 4
1.5 Tesla Roadster. 5
1.6 Rimac Concept One 6
1.7 Tesla Modelo S100D 6
1.8 Nissan LEAF 6
1.9 Freno de tambor 9
1.10 Freno de disco 10
1.11 Freno dinámico 10
1.12 Waymo 11

2.1 Cuadriciclo experimental 12
2.2 Actuador del volante 13
2.3 LIDAR 13
2.4 Cámara frontal 14
2.5 Esbozo inicial 15
2.6 Esbozo inicial modificado 16
2.7 Sistema inteligente de control (hardware) 17
2.8 Sistema de control de lazo cerrado 20
2.9 Controlador PID 21
2.10 Integración trapezoidal 23
2.11 Derivative-kick 25
2.12 Wind-up 25

3.1 Motor del vehículo experimental. 27
3.2 Baterías del vehículo experimental. 28
3.3 Rectificador. 28
3.4 Convertidor. 28
3.5 Controladora 29

3.6 Cableado de la controladora 29
3.7 Pines para la señal de aceleración de la controladora 30
3.8 Clemas para alternar tipo de conducción 30
3.9 Alternar entre modo manual y autónomo 31
3.10 Interruptores de selección de modo 31
3.11 Señal PWM 32
3.12 Filtro activo 33
3.13 Pines de la controladora para el sensor Hall 33
3.14 Prueba para medir los pulsos del sensor Hall del motor 34
3.15 Respuesta al acelerar 36
3.16 Aceleración máxima 37
3.17 Aceleración máxima con anotaciones 38
3.18 Forma de la respuesta ante un aumento de tensión constante 39
3.19 Esquema eléctrico subsistema de aceleración 40
3.20 Circuito físico 40
3.21 Respuesta con los parámetros del PID escogidos. 41
3.22 Respuesta con los parámetros del PID escogidos con anotaciones. . . . 42

4.1 Impacto frontal de un vehículo 45
4.2 Accionamiento de los dispositivos de frenado. 46
4.3 Freno en conducción manual 47
4.4 Central hidráulica compacta HAWE NPC 48
4.5 Central hidráulica compacta del vehículo experimental 48
4.6 Componentes principales de la central hidráulica compacta 49
4.7 Bomba de engranajes 50
4.8 Electroválvula 50
4.9 Esquema de conexión central compacta 51
4.10 Conexiones de la electroválvula 52
4.11 Circuitos aislados 53
4.12 Circuitos con masa común 53
4.13 MOSFET como interruptor 56
4.14 Circuito de control para freno en conducción automática 57
4.15 IRLZ44n 59

5.1 Esquema eléctrico final 1/2 61
5.2 Esquema eléctrico final 2/2 62

6.1 Boceto inicial del tablero de abordo 64

6.2 Programas para el tablero de abordo 65
6.3 Panel de fondo 66
6.4 Velocimetro 66
6.5 Perfil del vehículo 67
6.6 Modelo del vehiculo extruido 67
6.7 UV del modelo del vehiculo 67
6.8 Modelo del vehiculo con UV 68
6.9 Modelo final 68
6.10 Sensores al mínimo (Arduino) 75
6.11 Sensores al mínimo (Processing) 75
6.12 Sensores al máximo (Arduino) 76
6.13 Sensores al máximo (Processing) 76
6.14 Sensores con valor aleatorio (Arduino) 77
6.15 Sensores con valor aleatorio (Processing) 77
6.16 Filtro paso bajo 79
6.17 Filtro paso alto 79
6.18 Filtro pasa banda 79
6.19 Filtro rechaza banda 80
6.20 Parámetros filtro paso bajo 80
6.21 Filtro Butterworth con distintos ordenes 81
6.22 Implementación filtro activo 82
6.23 Salida sin filtrar 84
6.24 Tensiones entradas digitales ATmega328p 84
6.25 Filtro matlab 85
6.26 Filtro 85
6.27 Señal filtrada 86
6.28 Prueba comunicación serie 92
6.29 Prueba comunicación ethernet 97

Capítulo 1

INTRODUCCIÓN.

Se va a desarrollar un **control de velocidad** con objeto de implementarlo en un **vehículo eléctrico** experimental del departamento TEISA de la Universidad de Cantabria.
Dicho control de velocidad supone una de las bases sobre la que se sustentará la búsqueda de la **autonomía del vehículo** (objetivo final del departamento).

Antes de comenzar con el desarrollo del control de velocidad se procede a hablar brevemente sobre los dos conceptos que enmarcan el trabajo:

- Vehículos eléctricos.
- Vehículos autónomos.

Cabe destacar que, pese a aparecer relacionados en múltitud de ocasiones, son conceptos independientes (la desapareción de uno no impediría la evolución del otro).

1.1 VEHÍCULOS ELÉCTRICOS.

1.1.1 Breve historia.

Los primeros planos de un vehículo autopropulsado de los que se tiene constancia datan de 1672 y provienen del jesuita Ferdinand Verbiest quien, como regalo al emperador chino, construyó un juguete autopropulsado a vapor. En la figura 1.1 [1] se observa una representación del mismo.

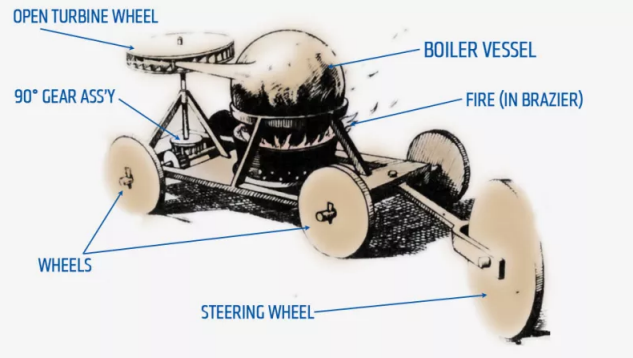

Figura 1.1: Vehículo de Verbiest

El título de ser el primero en poner un vehículo funcional autopropulsado en las calles se lo suele llevar, sin embargo, el inventor francés Nicolas-Joseph Cugnot quien, en 1769, consiguió hacer funcionar una de las versiones de su ”coche a vapor” con el propósito de transportar cargas pesadas. En la figura 1.2 [2] se muestra una réplica de dicho vehículo.

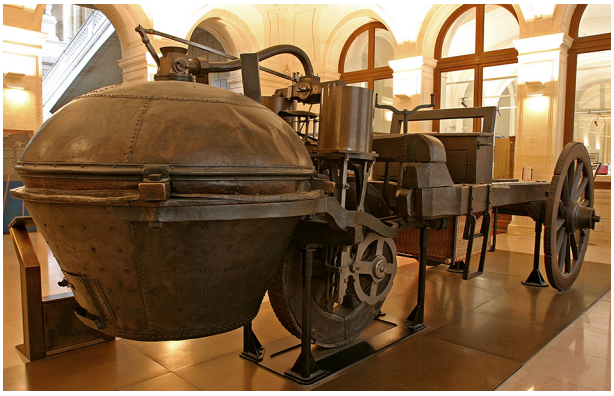

Figura 1.2: Vehículo de Cugnot

Ambos vehículos usaban el vapor como principio de funcionamiento, manteniéndose éste como único sistema para tal fin hasta la aparición del vehículo eléctrico.

Los primeros vehículos autopropulsados eléctricamente capaces de hacerse un hueco en las calles tuvieron que esperar a la aparición de las primeras baterías recargables (plomo-ácido) en 1860, a partir de este punto el vehículo eléctrico sufriría grandes avances que lo llevaron a ocupar un lugar predominante en el transporte.

Merece la pena destacar que el primer vehículo en romper la barrera de los 100 km/h fue eléctrico. Se denominaba ”la Jamais Contente” y lo consiguió en 1899. En la figura 1.3 [3] se muestra dicho vehículo.

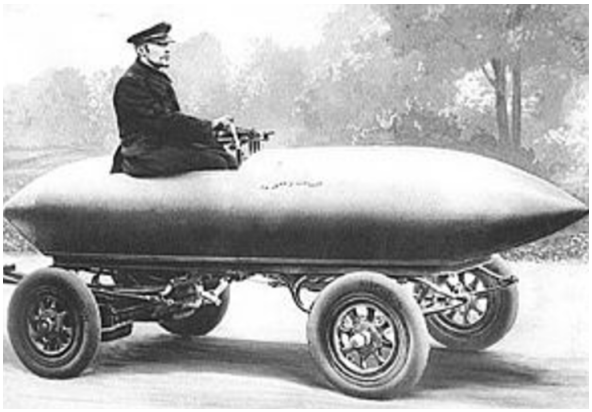

Figura 1.3: La Jamais Contente

La aparición de las baterías de niquel-hierro supuso una considerable mejora en términos de velocidad y autonomía respecto a su predecesora, pese a todo, su edad de oro tocó fin a manos de los vehículos autopropulsados por derivados del petroleo.

Este tipo de vehículos se impuso principalmente por las siguientes razones:

1. Su mayor autonomía.
2. El bajo precio del petroleo en aquella época.
3. La aparición del arrancador eléctrico.
4. La producción en cadena con la consiguiente disminución de precio.

No obstante, la propulsión eléctrica presentaba ciertas ventajas respecto a su principales rivales: par máximo a velocidades muy bajas, curva de par plana para un gran rango de revoluciones, sin necesidad de cajas de cambios, ausencia de embrague, reducción notable de ruido,... [4].

Los vehículos eléctricos acabaron siendo relegados a aplicaciones donde no se precisaba una gran autonomía (carretillas elevadoras, carritos de golf,...), o donde existiese una fuente de tensión externa para suministrar electricidad sin necesidad de baterías (trenes de alta velocidad, trolebús,...).

Para poder aprovechar el potencial de la propulsión eléctrica en aplicaciones donde el uso exclusivo de baterías era inviable, aparecieron métodos de autopropulsión mixta como la propulsión diesel-eléctrica.
Con este tipo de propulsión la electricidad es la encargada de alimentar los motores, sin embargo, para generar la electricidad, se dispone de ”pequeñas centrales eléctricas” (en este caso mediante generadores diesel).
En la figura 1.4 [5] se muestra un esquema de este tipo de propulsión aplicada a un submarino.

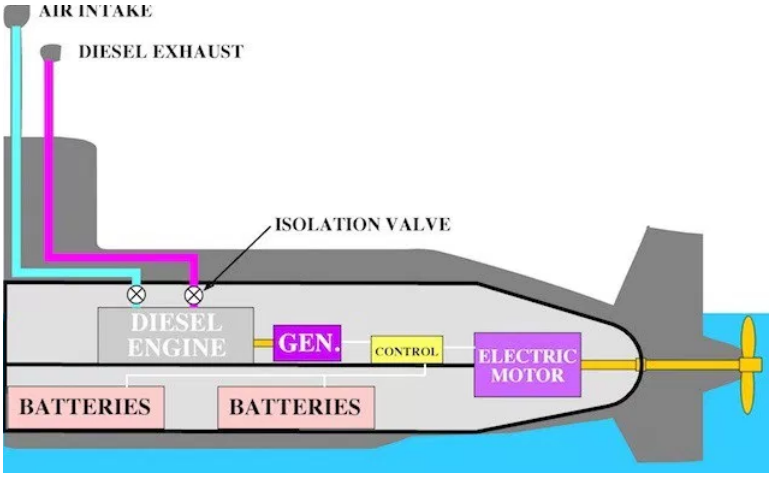

Figura 1.4: Propulsión diesel-eléctrica en un submarino

Actualmente la autopropulsión eléctrica está resurgiendo a pasos agigantados, especialmente en uno de los mercados donde hace unos años prácticamente había desaparecido, el de automóviles de turismo.

Esto se debe, en gran medida, a la concienciación que han empezado a tomar las personas respecto a la contaminación y sus consecuencias, así como a la certeza de que los combustibles fósiles no durarán indefinidamente.
Varios países han anunciado su intención de prohibir la venta de coches diesel y gasolina a lo largo de los próximos años en favor de los coches eléctricos (Francia, Noruega, Reino Unido,...).

Se suele pensar en los vehículos eléctricos como los sustitutos ”limpios” de los vehículos de combustión interna, no obstante, no debemos caer en la trampa de pensar que su nivel de contaminación sea nulo.
Los críticos suelen blandir dos argumentos en contra:

- **«Un vehículo eléctrico será tan limpio como limpia halla sido la generación de la energía que está usando (hasta la fecha las centrales de energía renovable forman un pequeño porcentaje del total)».**
  Esto es totalmente cierto, se desplaza el problema de la contaminación de los vehículos a las centrales, no obstante, acarrearía ciertas ventajas:
  - El aumento de centrales de energías renovables supondría una reducción inmediata en la contaminación asociada a este tipo de vehículos.
  - Sería más sencillo controlar la contaminación asociada a unos cientos de centrales que a unos cientos de millones de vehículos.
  - Se alejarían los focos de contaminación de los núcleos urbanos.
- **«Las baterías usadas en los coches eléctricos suelen llevar ligado algún tipo de contaminación (fabricación, uso y/o fin de vida útil)».**
  Actualmente se pueden reciclar prácticamente en su totalidad, no obstante, sigue siendo uno de los mayores inconvenientes de los coches eléctricos.

1.1.2 Automóviles eléctricos.

La aparición del Tesla Roadster (figura 1.5 [6]), anunciado en 2006 y puesto en el mercado en 2008, supuso un cambio en la percepción de estos vehículos.
Con una aceleración de 0 a 100 km/h en 3.7 segundos y una autonomía de 393 km, la gente se dió cuenta de que este tipo de vehículos se habían convertido en una opción realmente viable.
Desde entonces hasta la fecha multitud de conocidas y nuevas marcas se han adentrado en la producción de este tipo de vehículos y, como consecuencia de ello, a día de hoy tenemos un amplio catálogo a nuestra disposición.

Figura 1.5: Tesla Roadster.

En 2015 entró en producción el que es hasta el momento el coche eléctrico más veloz del mercado, el Rimac Concept One (figura 1.6 [7]), capaz de alcanzar la increible velocidad de 355 km/h (limitados electrónicamente), pasando de 0 a 100 km/h en solo 2.6 segundos.

Figura 1.6: Rimac Concept One

Si lo que buscamos es autonomía Tesla vuelve a aparecer en la lista con su Tesla Modelo S100D (figura 1.7 [8]), que cuenta con una autonomía rondando los 600 km en conducción urbana además de con una aceleración de 0 a 100 en 2.7 segundos.

Figura 1.7: Tesla Modelo S100D

Estos vehículos tienen buenas prestaciones pero el precio es elevado.
No obstante, como se ha comentado, disponemos de multitud de opciones que nos proporcionan prestaciones lo suficientemente buenas para la mayor parte de personas a precios razonables. Uno de los vehículos eléctricos más vendidos es el Nissan LEAF (figura 1.8 [9]), con una aceleración de 0 a 100 km/h en 11.5 segundos, una velocidad punta de 145 km/h y una autonomía de entre 117 y 172 km.

Figura 1.8: Nissan LEAF

1.1.3 Partes principales.

Motor eléctrico.

Son la base de todo vehículo eléctrico, el elemento que posee la capacidad de transformar energía eléctrica en mecánica consiguiendo que nuestro vehículo se ponga en movimiento.

Existen dos tipos claramente diferenciados: motores de corriente alterna y motores de corriente continua.

Los coches con motores de corriente continua son más baratos en su construcción, además, al ser las baterías de corriente continua, se evita la necesidad de usar un inversor.
Por otro lado, los motores de alterna presentan una serie de ventajas, entre las que destacan la seguridad, la sencillez para cambiar de dirección (invertir dos fases), la sencillez en la instalación, la ausencia de escobillas... [10]

Controladora.

La controladora es el cerebro de todo vehículo eléctrico.
Es el elemento encargado de decidir como se comportará nuestro vehículo en cualquier instante.

La función básica que cualquier controladora de un vehículo eléctrico debe poseer es la de controlar la aceleración del vehículo.
Mediante una señal de baja potencia (generalmente un potenciómetro unido al pedal de aceleración), la controladora se encarga de proporcionar al motor la tensión de alimentación correspondiente.
Nos aisla, por tanto, un circuito de baja potencia de uno de alta potencia.

Generalmente las controladoras cuentan con bastantes más parámetros que, según su valor, regirán el comportamiento del vehículo en diversas situaciones (limitación de revoluciones del motor, reducción de potencia para evitar sobrecalentamiento, protección de baterías, condiciones de arranque,...).

Baterías.

Si no se dispone de una propulsión mixta (por ejemplo la diesel-eléctrica mostrada anteriormente), y el vehículo no está en contacto con ningún elemento que le proporcione una corriente eléctrica (trolebuses, trenes de alta velocidad,...), con total probabilidad la alimentación eléctrica para el motor de nuestro vehículo vendrá dada por un conjunto de baterías.

Las siguiente baterías ya se han ganado un nombre en el campo de la autopropulsión eléctrica [11]:

- Ácido-plomo.
  Son las más baratas pero suponen entre un 25% y un 30% del peso final del vehículo.
  Su eficiencia ronda el 70-75% y cuentan con una densidad de energía de entre 30 y 40 Wh/Kg.
  Pueden emitir oxígeno, hidrógeno y azufre al cargarse y cuando están en operación (en cantidades inofensivas si se operan de forma correcta).
- Hidruro de niquel.
  El rendimiento de estas baterías ronda el 60-70% pero cuentan con una densidad de energía de entre 30-80 Wh/kg.
  Presentan una larga vida útil siendo la reducida eficiencia el mayor de sus problemas.
- ZEBRA.
  Cuentan con una densidad de energía de 120 Wh/kg.
  Permiten un elevado número de cargas y no son tóxicas, el principal inconveniente es la necesidad de calentar el electrolito.
- Ión de litio.
  Este tipo de baterías, muy comunes en dispositivos electrónicos, también se han ganado un hueco en el mundo de la autopropulsión eléctrica.
  Su densidad de energía puede superar los 200 Wh/kg y cuenta con una eficiencia de carga/descarga que ronda el 80-90%.
  Como contrapartida presentan una alta degradación con el tiempo y el número de ciclos de carga no es muy elevado. Además se deben tratar con especial precaución (las nuevas baterías suelen disminuir las posibilidades de que se prendan fuego sacrificando otros parámetros como la densidad de energía).

Frenos.

La teoría detrás de los mecanismos de frenado es sencilla, para frenar un objeto en movimiento debemos transformar su energía cinética en otro tipo de energía (generalmente térmica).

Lejos quedan los bloques de madera aplicados mediante palancas heredados de los carruajes que, con el aumento de la velocidad de los vehículos y la aparición de los neumáticos de goma, quedaron rápidamente obsoletos.
Los posteriores avances y patentes nos llevaron hasta los principales mecanismos de frenado que usamos en la actualidad.
Los más comunes en los vehículos eléctricos son:

Freno de tambor.

Este tipo de frenos transforman energía cinética en térmica. Por normal general son accionados mediante un sistema hidráulico, como se puede observar en la figura 1.9 [12].
Al accionar el freno la presión del líquido de frenado aumenta, esto provoca que unas zapatas entren en contacto con el tambor, disminuyendo la velocidad del vehículo.
El tambor es un elemento móvil solidario a la rueda, generalmente de fundición (bajo precio y alta conductividad térmica).
Las zapatas están formadas, generalmente, por chapas de acero con un revestimiento que proporcione buena resistencia al desgaste y a la fricción.
Por norma general son frenos relativamente baratos.

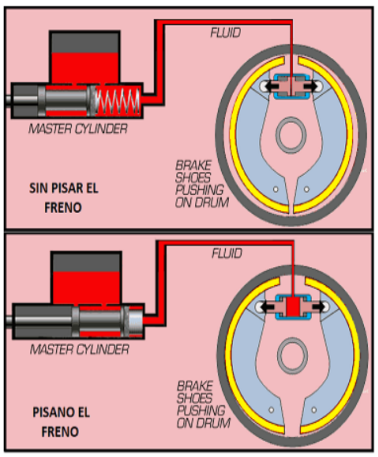

Figura 1.9: Freno de tambor

Freno de disco.

Transforman energía cinética en térmica.

Sulen accionarse, como en el caso anterior, mediante un circuito hidráulico.

Al accionar el freno la presión del líquido de frenado aumenta, esto provoca que los brazos de fricción de una pinza entren en contacto con un disco que gira solidario a la rueda, produciendo la reducción de velocidad del vehículo (figura 1.10 [13]).

Mejoran la frenada respecto a los frenos de tambor ya que están montados al aire (mejora la disipación de calor), como contrapartida nos encontramos el precio.

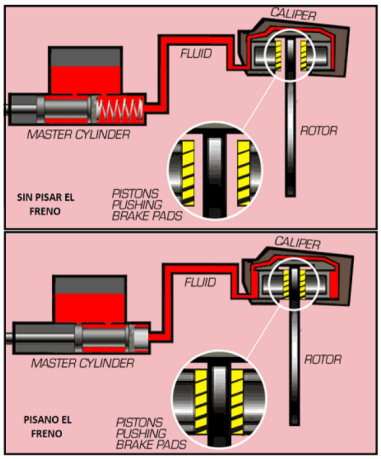

Figura 1.10: Freno de disco

Freno dinámicos.

Muy comunes en los vehículos eléctricos ya que basan su funcionamiento en las propiedades inherentes de sus motores.

Decimos que un motor eléctrico funciona como freno dinámico cuando, en lugar de generar movimiento a partir de electricidad, genera electricidad a partir de movimiento.

Si esta electricidad se disipa mediante un banco de resistencias será de tipo reostático. Si es devuelta a la alimentación será de tipo regenerativo.

En la figura 1.11 [14] se puede diferenciar el flujo de energía cuando proviene del motor (izquierda), de cuando se dirije al mismo (derecha).

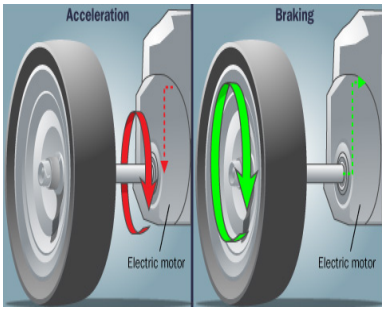

Figura 1.11: Freno dinámico

1.2 VEHÍCULOS AUTÓNOMOS.

Llamamos vehículo autónomo a aquel capaz de prescindir de un piloto en su funcionamiento.

Todo coche autónomo contará con los siguientes bloques.

- Sensores.
  Para obtener la información necesaria del medio.
- Unidad de procesado.
  Partiendo de la información de los sensores realiza el procesado y computo de la misma. El objetivo es obtener las señales adecuadas para los actuadores.
- Actuadores.
  Transforman las señales eléctricas de la unidad de procesado en cambios medibles del movimiento del vehículo.

La presentación más longeva de la que se tiene constancia data de 1939, donde Norman Bel Geddes expuso un vehículo eléctrico controlado por un circuito embebido en el pavimento de una carretera.

Algo menos de un siglo después el avance realizado en este campo es impresionante: Waymo (figura 1.12 [15]), el proyecto de coche autónomo de Google, es una clara muestra de ello, en mayo de 2017 alcanzó la cifra de 4 828 032 km recorridos en modo autónomo.

Figura 1.12: Waymo

Capítulo 2

VISIÓN GLOBAL.

El vehículo del que se dispone en el laboratorio y sobre el que se centra el presente trabajo es un cuadriciclo eléctrico (parecido a un carrito de golf), fabricado por la empresa TEYCARS, en concreto el modelo ALSO2 de 2008.
Cuenta con una velocidad máxima inferior a 50 km/h y una autonomía de unos 90 km.
En la figura 2.1 podemos ver dicho vehículo experimental durante una de sus pruebas por el recinto de la universidad.

Figura 2.1: Cuadriciclo experimental

2.1 OBJETIVO DEL DEPARTAMENTO.

Solo basta con dar una vuelta alrededor del vehículo para intuir el objetivo del departamento: obtener un vehículo completamente autónomo.

El presente trabajo, junto con un trabajo paralelo donde se está buscando un sistema de control para el volante, forman la base para poder actuar sobre el vehículo. Si miramos el volante comprobamos que efectivamente se ha instalado un motor y un encoder para tal fin, como se puede comprobar en la figura 2.2.

Figura 2.2: Actuador del volante

El vehículo cuenta además con una serie de sensores encargados de obtener información del medio y enviarla al sistema de control inteligente:

Sensor LIDAR

El nombre proviene de ”Light Detection and Ranging / Laser Imaging Detection and Ranging”.
Este sensor nos permite medir distancias al detectar los reflejos de un pulso láser previamente enviado.
Conocido el tiempo entre el envío de la señal y la detección del reflejo, así como la diferencia en la longitud de onda de ambas señales, podemos obtener una nube de puntos que formarán una representación tridimensional del entorno.
En la figura 2.3 se observa dicho sensor en el techo del vehículo.

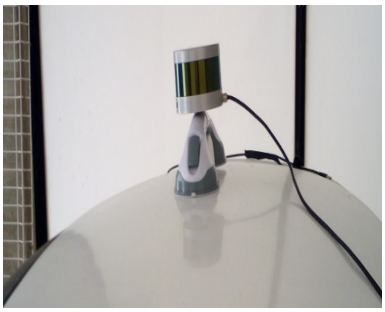

Figura 2.3: LIDAR

Cámaras

Podemos encontrar diversas cámaras a lo largo del vehículo (en el capó, en los espejos retrovisores,...).

La mostrada en la figura 2.4 se corresponde con la localizada en el capó.

Figura 2.4: Cámara frontal

Se ha mostrado una pequeña parte de lo que nos encontramos alrededor del vehículo, pero nos sirve para corroborar lo dicho anteriormente: se busca un coche completamente autónomo.

Todos estos elementos forman parte de trabajos que se están realizando en paralelo al aquí presente.

Cada uno de ellos formará un sistema individual que deberá comunicarse con el sistema de control inteligente, el cuál se encargará de calcular las señales a enviar a los sistemas encargados de controlar los actuadores.

El valor de dichas señales dependerá, tanto de la información recibida de los sensores, como de la ruta previamente programada.

2.2 OBJETIVO DEL TRABAJO.

El presente trabajo se centra en el control de velocidad longitudinal.

Partiendo de una velocidad de referencia que nos facilitará un sistema de control inteligente, debemos obtener y enviar hacia el sistema de aceleración y de freno las señales que mejor adapten la velocidad actual a la deseada.

De lo anterior se deduce la necesidad de una realimentación, de esta forma el sistema de control dispondrá tanto de la velocidad actual del vehículo como de la deseada, y podrá actuar en consecuencia.

2.3 REQUISITOS.

Se han impuesto los siguientes requisitos de diseño:

1. El sistema de control debe ser capaz de procesar la información que le llega y de actuar sobre el sistema, una vez cada decima de segundo.
   Dificilmente un ser humano puede procesar información en este espacio de tiempo por lo que se espera que sea un valor suficiente para lograr nuestro objetivo, más aún si tenemos en cuenta las bajas velocidades del cuadriciclo.
2. Se deben evitar aceleraciones bruscas fuera de la zona de comfort humana.

2.4 PARTE FÍSICA.

Analizando lo anterior llegamos a la conclusión de que necesitamos un sistema de lazo cerrado como el mostrado en la figura 2.5.

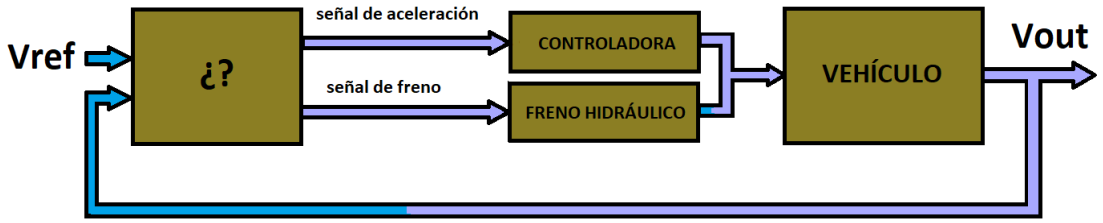

Figura 2.5: Esbozo inicial

En la figura observamos una incógnita en el bloque del controlador, esto se debe a que aún se tiene que elegir como se va a implementar dicho componente.

Se ha decidido, por su versatilidad, que ese lugar lo ocupe un microcontrolador (nuestro controlador se programará por software), además, para evitar tener que trabajar con demasiados componentes discretos, se ha optado por usar una placa de desarrollo.

Las placas de desarrollo están formadas por un microcontrolador y una serie de componentes electrónicos interconectados mediante circuitos impresos. Son muy útiles a la hora de ayudarnos a conocer las posibilidades que nos ofrece un determinado microcontrolador, así como a la hora de realizar prototipos y proyectos, facilitándonos la parte hardware para poder centrarnos en el software.
Como contrapartida no siempre usaremos todos los elementos que forman parte de la placa de desarrollo y, en ocasiones, no contará con los elementos que necesitemos.

Una de las placas que más ha destacado en los últimos años es la denominada como Arduino UNO [16], por lo que cuenta con una gran relación calidad precio y una amplia disponibilidad en multitud de comercios, por ello, y por ser lo suficientemente potente para conseguir nuestros objetivos, hemos decidido hacer uso de ella.

Es importante destacar que las placas de desarrollo son muy útiles como punto de partida, no obstante, si quisiéramos comercializar el sistema, probablemente podríamos aumentar el margen de beneficio utilizando un microcontrolador con poder de calculo acorde a lo que se le va a exigir, y haciendo uso unicamente de los componentes discretos necesarios.

Trás esta modificación el sistema toma la forma indicada en la figura 2.6.

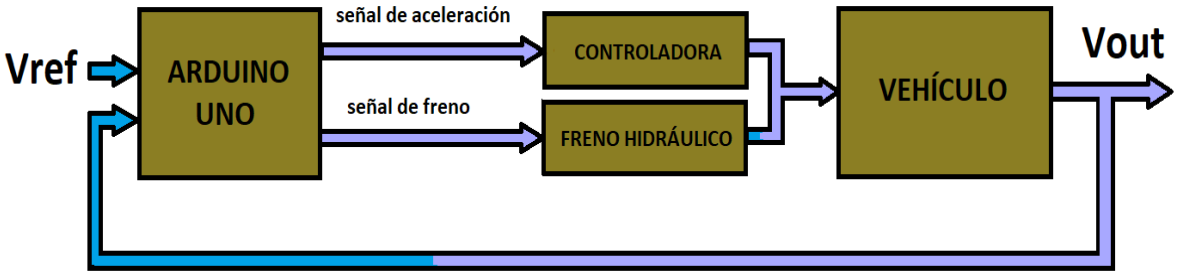

Figura 2.6: Esbozo inicial modificado

2.5 PARTE LÓGICA.

Al haber escogido una placa de desarrollo como controlador del sistema, será indispensable poder programarlo para ajustarlo a nuestra aplicación.

Arduino dispone de un entorno de desarrollo propio basado en C++ (denominado **IDE de Arduino**), que nos facilita enormemente la programación de esta familia de placas y del cuál haremos uso.

2.6 COMUNICACIÓN.

Tenemos que elegir un protocolo de comunicación para hacer factible un intercambio fiable de datos entre la placa de Arduino y el sistema de control inteligente.

2.6.1 Sistema de control inteligente.

El sistema inteligente encargado de procesar la información se implementará en un ordenador industrial con ventilación pasiva (de ahí que el chasis sea de aluminio, un buen conductor térmico a la vez que liviano), en concreto un NISE 3500 (mostrado en la figura 2.7 [17]).

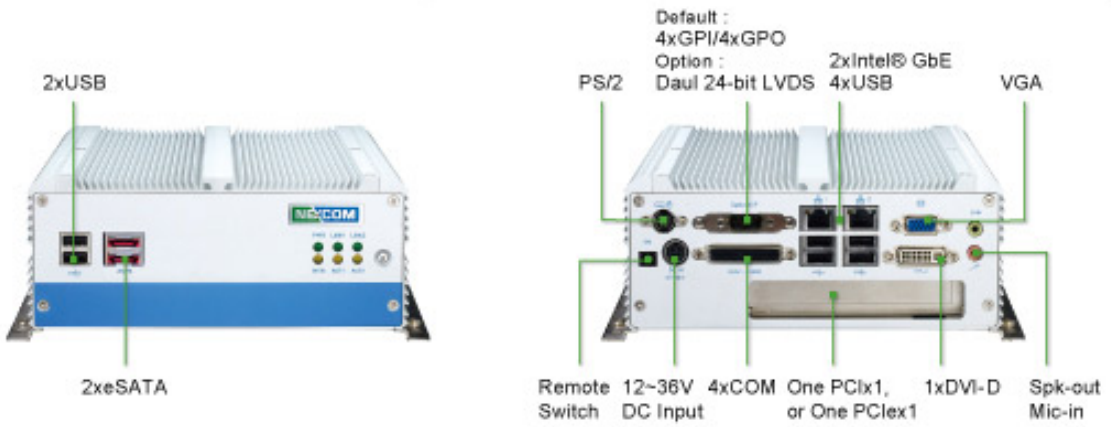

Figura 2.7: Sistema inteligente de control (hardware)

Este tipo de ordenadores están adpatados a entornos industriales, donde las condiciones de trabajo suelen ser peligrosas para un ordenador convencional.

Al prescindir de ventiladores:

- Eliminamos la posibilidad de que estos introduzcan partículas de polvo en su interior (evitando los fallos asociados).
- El ordenador se vuelve silencioso (eliminamos las principales fuentes de ruido).
- Aumenta su robustez (se eliminan componentes con piezas móviles).
- Disminuye el consumo eléctrico.

Lo único que aún no se ha decidido es el software encargado del procesado de la información, no obstante, es algo que no afecta al desarrollo del presente trabajo.

A lo largo de este proyecto se toma MATLAB como referencia (se programará en este entorno cuando se quiera simular la comunicación con el sistema inteligente encargado, entre otras cosas, de decidir cuál es la velocidad de referencia).

2.6.2 Protocolos de comunicación.

Arduino nos permite hacer uso de un gran número de métodos de comunicación, en ocasiones integrados en la placa y en ocasiones con ayuda de módulos externos (más conocidos como ”shields”).
Podemos usar comunicación serie por USB, Ethernet, SPI, I2C, Bluetooth, Wifi...

En lo que a este trabajo se refiere vamos a emplear dos tipos de comunicación:

1. Comunicación serie por puerto USB.
   Para las pruebas de laboratorio se escoge esta opción debido a la comodidad que supone emplear el mismo cable para alimentar la placa, cargar los programas, y establecer la comunicación con el sistema inteligente de control.
2. Comunicación Ethernet.
   La comunicación del sistema final empleará Ethernet UDP.
   Facilitará la programación de las comunicaciones del futuro coche autónomo cuando existan multitud de subsistemas trabajando de manera simultánea.
   Además nos evitara tener que preocuparnos de las distancias de cableado ya que podemos usar cables de hasta 100 metros de longitud.

2.7 MECANISMO DE CONTROL.

Lo que denominamos como teoría de control es un amplio campo de estudio cuyo objetivo último es el control de una/s variable/s de salida.

Un microondas es un ejemplo de ello, podemos modificar una serie de entradas (tiempo, potencia...) para influir en una variable final (la temperatura del alimento).
Este ejemplo se corresponde con un mecanismo de control de ”lazo abierto” (llamamos así a los mecanismos de control en los que, tras seleccionar la/s entrada/s, no se tiene en cuenta si el resultado final es el esperado).

En el ejemplo anterior, si introducimos alimentos a distinta temperatura inical, obtendremos una temperatura final diferente en cada situación, haciendose necesaria una modificación manual de los parámetros de entrada si queremos una temperatura final estable.
Incluso puede darse el caso de que, introduciendo siempre el mismo alimento a la misma temperatura inicial obtengamos temperaturas de salida distintas. Esto es debido a que en los sistemas pueden aparecer ”perturbaciones” tanto internas como externas.

En nuestro caso, donde necesitamos variar la velocidad del vehículo en base a una velocidad de referencia, un sistema de lazo abierto no nos sirve.
Un coche es un sistema muy complejo donde muchos factores entran en juego (entre ellos las perturbaciones), la única manera de poder solucionar este inconveniente es mediante un sistema de control de ”lazo cerrado”.
Los mecanismos de control de lazo cerrado tienen en cuenta el valor de la/s salida/s en todo momento y actúan en consecuencia. A este fenómeno se le conoce como realimentación.

Un ejemplo de sistema de lazo cerrado es un sistema de calefacción automático.
Cuando el termostato detecta una caída de temperatura por debajo del límite inferior programado se activa el sistema de calefacción, y cuando la temperatura alcanza el límite superior se desactiva (este es el controlador de lazo cerrado más sencillo que podemos encontrar y se denomina controlador todo o nada).

En la figura 2.8 [18] se muetra el esquema básico de un sistema de control de lazo cerrado.

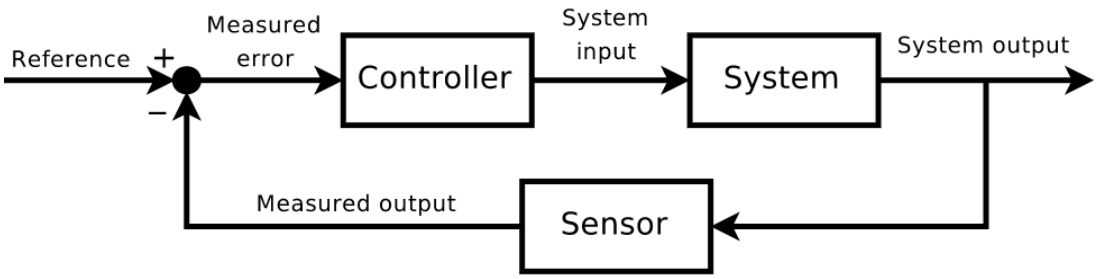

Figura 2.8: Sistema de control de lazo cerrado

Como ya hemos visto, el software que actuará como controlador se programará en un Arduino UNO pero, ¿qué tipo de controlador programamos?

Un controlador todo o nada como el comentado anteriormente no es factible en nuestro caso (si solo pudiésemos variar la velocidad aplicando una entrada máxima y una entrada nula la experiencia de conducción sería nefasta, el coche funcionaría a trompicones de manera permanente).
Necesitamos por tanto un controlador más complejo.

El controlador más extendido en sistemas de lazo cerrado a nivel industrial es el controlador PID [19].
Debido a su facilidad de implementación y versatilidad es el controlador escogido para este proyecto.

Un controlador toma una señal de error y, tras realizar una serie de cálculos, nos devuelve la entrada que debemos introducir a la planta.
Mediante este dispositivo nos aseguramos de que la salida del sistema se aproxime lo máximo posible a la señal de referencia deseada, aún en presencia de perturbaciones tanto internas como externas.

En el caso del PID nos podemos hacer una primera idea de los cálculos que se realizan atendiendo a su nombre (anagrama de Proporcional, Integral y Derivativo).

En la figura 2.9 [20] se muestra el esquema básico de este controlador en un sistema de lazo cerrado.

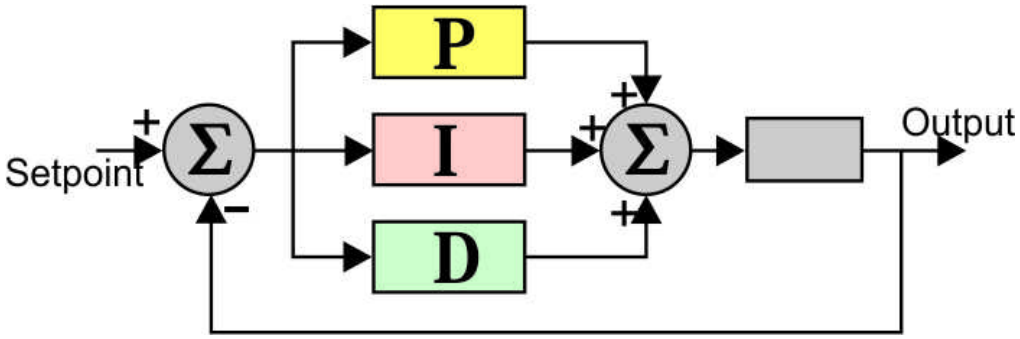

Figura 2.9: Controlador PID

Un PID consta tres acciones, cada una de ellas nos brinda un parámetro que podemos modificar para adaptar la respuesta a nuestro gusto.

- Acción proporcional.
  Esta acción es la que más afecta al resultado final.
  Su salida es directamente proporcional a la señal de error.
  Un valor de la constante proporcional (K_P) muy elevado puede llevar el sistema hacia la inestabilidad, no obstante, si el valor es demasiado bajo, el tiempo de subida puede ser elevado así como su error en estado estacionario.

$$P = K_P \cdot e(t)$$

- Accion integral.
  El objetivo de esta acción es eliminar el error de estado estacionario que permanece tras aplicar la acción proporcional.
  Un valor muy alto de la constante integral (K_I) llevará al sistema a la sobreoscilación.

$$I = K_I \int_0^t e(\tau)\, d\tau$$

- Acción derivativa.
  Se suele recurrir a esta componente cuando se busca mejorar el tiempo de establecimiento y la estabilidad del sistema, no obstante, se debe tener cuidado al usarla (actúa como amplificador de ruido de altas frecuencias pudiendo obtener un comportamiento totalmente opuesto al esperado, es decir, haciendo el sistema inestable).
  Intentaremos evitar esta componente siempre que nos sea posible.

$$D = K_D \frac{d}{dt}(e(t))$$

Expresión general del PID.
Se obtiene sumando las expresiones anteriores.

$$PID = K_P \cdot e(t) \; + \; K_I \int_0^t e(\tau)\, d\tau \; + \; K_D \frac{d}{dt}(e(t))$$

Es muy común encontrarnos con dicha fórmula expresada en el dominio de Laplace.

$$PID(s) = K_P \; + \; \frac{K_I}{s} \; + \; K_D \cdot s$$

En el caso de que el sistema sea lineal y tiempo invariante (LTI) el dominio de Laplace es una gran herramienta (en el mundo real no existen sistemas LTI, no obstante, en muchas situaciones, podemos recurrir a linealizar nuestro sistema en torno a un punto de trabajo y, suponiendo parámetros invariantes, podemos obtener resultados relativamente buenos).

A continuación se indica como programar un PID en la placa Arduino UNO.

Como nuestro programa se repite cada 0.1 segundos aprovecharemos este intervalo de tiempo para realizar los cálculos del PID, es decir, será nuestro periodo de muestreo.

Al disponer de un periodo de muestreo constante podemos pasar las expresiones anteriores de su forma continua a su forma discreta.

$$\underline{\text{Accion proporcional}} \rightarrow P = K_P\,(Error\ Actual)$$

$$\underline{\text{Acción integral}} \rightarrow I = K_I\left(\frac{Error\ Actual + Error\ Anterior}{2}\ T_S\right)$$

$$\underline{\text{Accion derivativa}} \rightarrow D = K_D\left(\frac{Error\ Actual - Error\ Anterior}{T_S}\right)$$

Para mejorar la aproximación de la integral, en lugar de usar la típica integral hacia adelante o hacia atrás, se ha empleado una integral trapezoidal (el resultado es análogo a la media de estos dos métodos pero mediante un código más sencillo). En la figura 2.10 [21] se muestra como se obtiene el área bajo la curva en este tipo de integral.

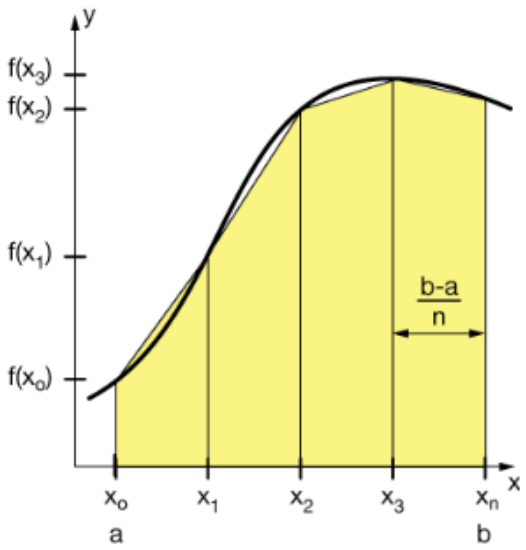

Figura 2.10: Integración trapezoidal

Código Arduino.

El código correspondiente a esta parte se puede consultar en el **ANEXO III: CÓDIGOS PARCIALES DEL CONTROL DE VELOCIDAD**.

Diferencias que encontramos en el código respecto al PID ideal:

1. División del error por la velocidad máxima.

$$\frac{Error}{V_{max}} = \frac{Referencia - Salida}{V_{max}} = \frac{40-0}{40} = 1$$

$$\frac{Error}{V_{max}} = \frac{Referencia - Salida}{V_{max}} = \frac{0-40}{40} = -1$$

   Como se puede observar se normaliza el error.

2. Cuando la entrada de referencia varía, la señal de error también lo hace. Debido al pequeño intervalo de tiempo que presenta el periodo de muestreo, un pequeño cambio en la señal de referencia supone un valor momentáneo en la salida de la acción derivativa muy elevado, a este fenómeno se lo conoce como derivative kick [22].
   Pese a que, como se comentó, se intentará evitar el uso de la acción derivativa del PID (debido principalmente a los problemas con el ruido), este problema en concreto es tan sencillo de mitigar que se ha tenido en cuenta en el código. La solución más sencilla se desprende de la propia definición de la señal de error. Si aplicamos la primera derivada respecto al tiempo a esta señal obtenemos la siguiente expresión:

$$\frac{d}{dt}(Señal\ de\ Error) = \frac{d}{dt}(Señal\ de\ Referencia) - \frac{d}{dt}(Salida\ del\ sistema)$$

   Cuando la señal de referencia es constante la derivada del error es igual a la derivada negativa de la salida del sistema:

$$\frac{d}{dt}(Señal\ de\ Error) = -\frac{d}{dt}(Salida\ del\ sistema)$$

   La segunda parte de la expresión anterior, ante un cambio brusco en la referencia, se ajusta de forma menos violenta.

Para mitigar el ”derivative kick” usamos dicha expresión.

En la figura 2.11 se compara el derivative kick de cada expresión.

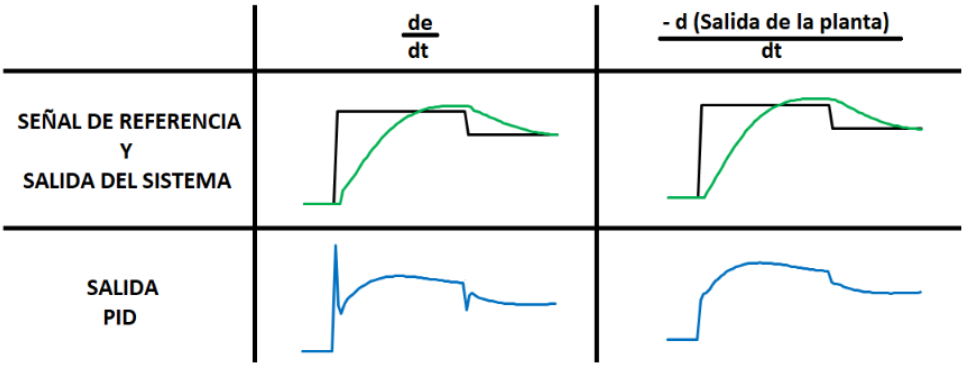

Figura 2.11: Derivative-kick

3. La última modificación llevada a cabo tiene como objetivo desprendernos de otro problema común asociado a los PID, el Wind-Up [23].
   Este fenónemo aparece cuando el controlador cree estar enviando valores correctos y, sin embargo, están fuera del rango de acción de la planta o de la propia salida del PID.
   Si no tenemos en cuenta este fenómeno, con total probabilidad nos aparecerán retardos indeseados en la respuesta del sistema cada vez que salgamos del rango de acción.

   En la figura 2.12 se muestra dicho retardo.

Figura 2.12: Wind-up

Para deshacernos de este inconveniente es necesario indicar de alguna manera al controlador cuales son los límites de operación (existen diversos métodos). Se ha escogido uno de los más sencillos, limitar la salida del PID y la acción integral al rango deseado.

Por mucho que aumente la suma de la acción integral, una vez alcanzado el límite físico de la acción del PID su aumento no supondrá un cambio en la salida (es por tanto un cálculo innecesario), es más, es éste alto valor el que provocará los retardos asociados al windup una vez la salida vuelva a entrar en el rango controlable, ya que el valor deberá de descender en consecuencia.

* El uso de 1e-6 (10^{-6}) después de los tiempos de muestreo es debido al uso de microsegundos (para aumentar la precisión).
Con esta operación pasamos el tiempo a segundos.

Capítulo 3

SUBSISTEMA DE ACELERACIÓN.

Este subsistema será el encargado de acelerar el vehículo hasta la velocidad indicada por el sistema de control inteligente.

3.1 ELEMENTOS PRINCIPALES.

3.1.1 Motor.

El motor encargado de hacer andar a nuestro vehículo es un motor de corriente continua de 48 V y excitación shunt, en la figura 3.1 se muestran las características principales del mismo.

Figura 3.1: Motor del vehículo experimental.

En los motores DC tipo shunt el devanado inductor y el devanado inducido están conectados en paralelo a la alimentación.
Se emplean cuando se busca una velocidad constante que varíe poco con la carga.

3.1.2 Alimentación.

El vehículo cuenta con 8 baterías de 6V cada una localizadas bajo los asientos (figura 3.2), lo que nos da un total de 48 V.

Figura 3.2: Baterías del vehículo experimental.

Para la carga de las mismas se dispone de un rectificador que transforma los 220 voltios de alterna de un enchufe común, a los 48 voltios necesarios para cargar el vehículo.
Podemos ver dicho rectificador en la figura 3.3.

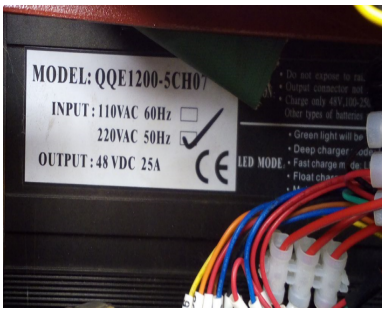

Figura 3.3: Rectificador.

Se dispone además de un conversor que reduce la tensión de las baterías a la tensión disponible en cualquier vehículo convencional (12 V), para poder alimentar las luces, el claxon...
En la figura 3.4 se observa el conversor instalado.

Figura 3.4: Convertidor.

3.1.3 Controladora.

La controladora es el cerebro de nuestro vehículo.

Contamos con una controladora electrónica Curtis modelo 1268 (figura 3.5).

Figura 3.5: Controladora

En la hoja de características se nos indica que es una controladora para motores de excitación separada, no obstante, podrá controlar nuestro motor shunt debido a la similutid entre ambas configuraciones (en un motor de excitación independiente tanto el devanado inductor como el inducido poseen alimentaciones independientes, en el caso de los motores shunt comparten la misma [4]).

En la figura 3.6 se muestra el cableado básico de la controladora.

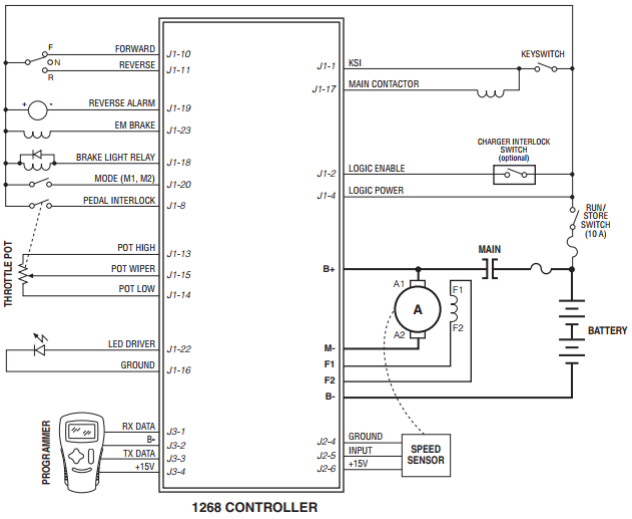

Figura 3.6: Cableado de la controladora

3.2 AJUSTES INICIALES.

En esta sección se recogen todos los ajustes previos necesarios para que la placa de desarrollo pueda actuar sobre el sistema.

3.2.1 Posibilidad de alternar entre modo de conducción autónomo y manual.

En conducción manual el pedal de aceleración controla el valor de un potenciómetro de 5kΩ que, a su vez, está conectado a los pines de la controladora encargados de transformar una señal de [0-5]V, en una excitación del motor proporcional al [0-100]%.

Si consultamos las hojas de características encontramos rápidamente los pines de la controladora asignados para tal fin (figura 3.7).

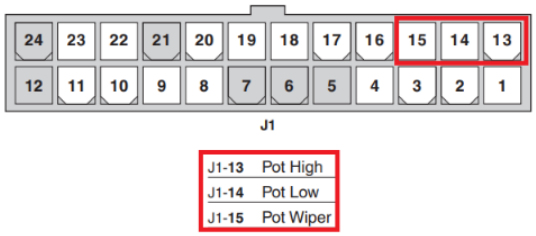

Figura 3.7: Pines para la señal de aceleración de la controladora

Con ayuda de unas clemas se ha creado un circuito en paralelo para poder seleccionar el modo de control del vehículo: autónomo o manual (figura 3.8).

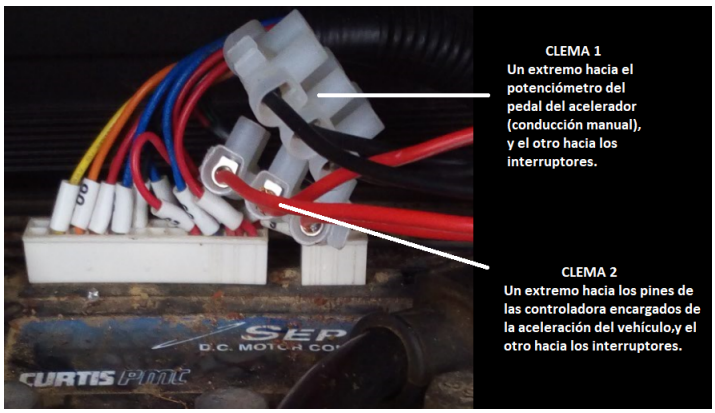

Figura 3.8: Clemas para alternar tipo de conducción

Ambos circuitos se conmutan mediante tres interruptores SPDT (simple pole double throw) como se muestra en la figura 3.9.

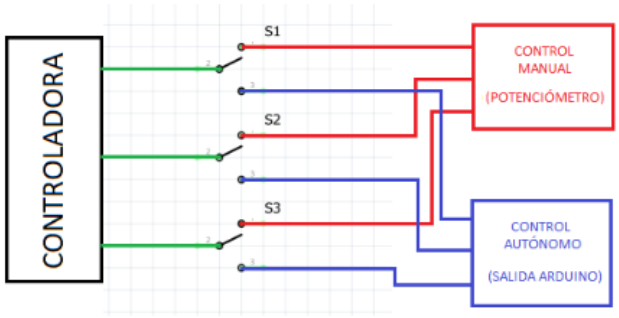

Figura 3.9: Alternar entre modo manual y autónomo

En la figura 3.10 se muestran los interruptores que habrá que instalar en el cuadro de mandos del vehículo.

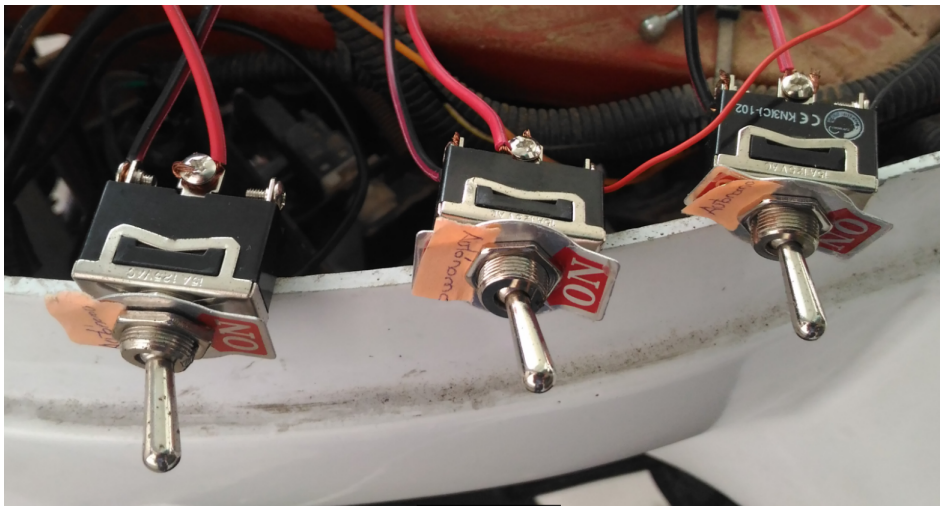

Figura 3.10: Interruptores de selección de modo

*Un único interruptor 3PDT puede hacer la misma función. La razón de haber usado tres SPDT en su lugar se debe a que se disponía de stock en el laboratorio.

3.2.2 Actuar sobre los pines de aceleración de la controladora.

La señal de aceleración se generará con Arduino.
La placa de desarrollo de Arduino solo puede proporcionar salidas digitales o PWM.

La modulación por ancho de pulsos (PWM) es una técnica de modulación en la que se codifica información por medio de variaciones de tensión de una determinada señal, se puede emplear como equivalente de una señal analógica siempre y cuando el receptor lo permita, por ejemplo: cambiar la intensidad de una luz (siempre que la frecuencia de la señal sea mayor a 60 Hz nuestros ojos no apreciarán las pulsaciones), para alimentar un motor (gracias a la inercia),...

Denominamos ciclo de trabajo al porcentaje de tiempo que la señal está activa respecto al tiempo de un ciclo de la misma, este será el factor que modificaremos para variar la tensión equivalente a la salida.

En la figura 3.11 [24] se puede observar el efecto de variar el ciclo de trabajo a una señal PWM.

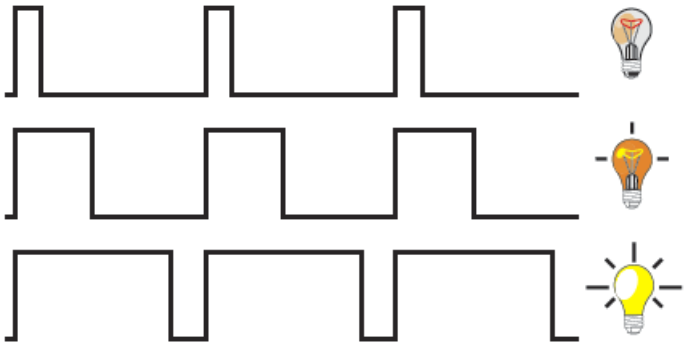

Figura 3.11: Señal PWM

En el caso de Arduino la salida puede tomar un valor o bien de 0V, o bien de 5V.
El ciclo de trabajo acepta un rango desde 0 (la señal completa a 0 V), hasta 255 (la señal completa a 5V).

En la hoja de características de la controladora se nos indica que la señal de control tiene que ser una señal analógica de entre 0 y 5 V de corriente continua (la señal PWM de Arduino no nos sirve).
Por suerte existe una manera muy sencilla de transformar una señal PWM en su equivalente analógica: filtrarla.

Vamos a usar un filtro RC [25] junto con un amplificador operacional (LM358) en modo seguidor de tensión (el filtro será por tanto activo).
Al no conocer los componentes que forman la circuitería interna de la controladora, añadir un operacional en modo seguidor de tensión nos elimina la necesidad de pensar en problemas de variaciones de carga, ya que la impedancia de salida del filtro será muy baja.

En la figura 3.12 [26] se observa el filtro que vamos a emplear, en nuestro caso R=10 kΩ y C=1pF.

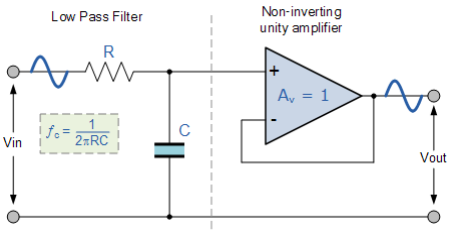

Figura 3.12: Filtro activo

En el **ANEXO I: CÁLCULO DE FILTROS** se muestra como se ha llegado al filtro anterior.

3.2.3 Obtener la velocidad del vehículo experimental.

Se va a obtener la velocidad del vehículo del sensor de efecto Hall localizado en el propio motor.

Para ello buscamos los pines de la controladora que reciben dicho valor (figura 3.13), y sacamos un circuito paralelo para poder usar la señal en el control autónomo.

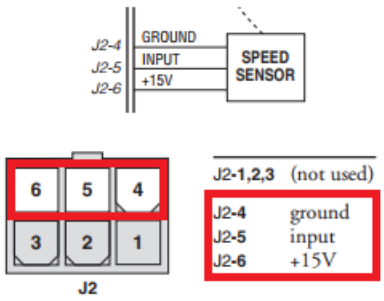

Figura 3.13: Pines de la controladora para el sensor Hall

Como no se dispone de la información del sensor de efecto Hall del motor se ha realizado un sencillo circuito que nos permitirá conocer el número de pulsos que envía el sensor por cada vuelta de la rueda.

Se ha enganchado un imán a una de las ruedas motrices y, mediante un sensor Hall discreto conectado a Arduino, se ha creado un programa que muestra por pantalla el número de pulsos enviados por el sensor Hall del motor cada vuelta de la rueda (figura 3.14).
La velocidad deseada la seleccionamos desde el propio Arduino modificando el valor de la salida correspondiente.

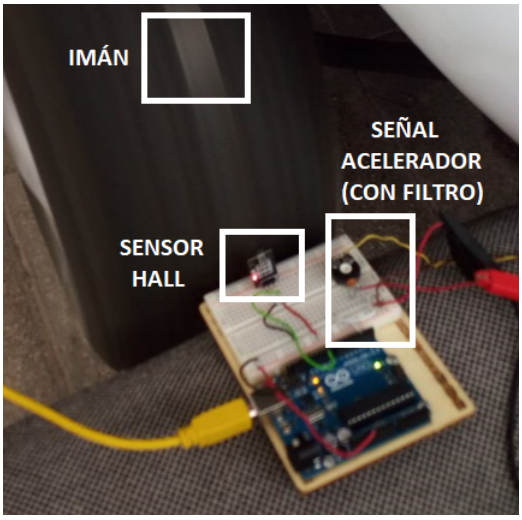

Figura 3.14: Prueba para medir los pulsos del sensor Hall del motor

El código empleado se puede consultar en el **ANEXO III: CÓDIGOS PARCIALES DEL CONTROL DE VELOCIDAD**.

El resultado obtenido es de 80 pulsos por cada vuelta de la rueda.

Ahora que conocemos el número de pulsos que le corresponden a una vuelta completa de la rueda, ya disponemos de todos los datos para calcular la velocidad del vehículo en base a esta variable.

$$\text{Distancia [m]} = \frac{\text{Pulsos}}{\text{Pulsos por vuelta}}[\text{vueltas}] * \text{Distancia cada vuelta}[\frac{\text{m}}{\text{vuelta}}]$$

$$\text{Velocidad [km/h]} = \frac{\text{Distancia recorrida [m]}}{\text{Tiempo empleado [s]}} * \frac{1\ [\text{km}]}{1000\ [\text{m}]} * \frac{3600\ [\text{s}]}{1\ [\text{h}]}$$

El tiempo empleado entre iteraciones será de 0.1 segundos (requisito del sistema) y, además, la distancia recorrida por vuelta es constante y coincide con la longitud de la rueda (1,76 m).

La ecuación contiene por tanto una única variable, los pulsos recibidos cada intervalo de tiempo.

$$\text{Velocidad [km/h]} = \frac{\frac{\text{Pulsos}}{80}[\text{vueltas}]*1.76[\frac{\text{m}}{\text{vuelta}}]}{0.1\ [\text{s}]} * \frac{1\ [\text{km}]}{1000\ [\text{m}]} * \frac{3600\ [\text{s}]}{1\ [\text{h}]}$$

Este cálculo se realizará una vez cada iteración del bucle del programa principal.

Para llegar a esta solución hubo que solventar un problema.
Al comenzar las pruebas el programa devolvía valores increíblemente elevados a velocidades reducidas, clara indicación de la existencia de algún tipo de error en la medición. Esto llevó a usar un osciloscopio de mano y comprobar la señal.
De esta forma se pudo observar que el problema era causado por el alto nivel de ruido contenido en la misma, ¿la solución? filtrar.

Se ha usado un filtro RC de primer orden [25] con R=3kΩ y C=1nF.

En el **ANEXO I: CÁLCULO DE FILTROS** se puede observar como se ha llegado a dichos valores.

3.2.4 Parámetros de la controladora.

Al tratarse de un vehículo de segunda mano la controladora ya nos venía programa con los parámetros escogidos por su anterior propietario.
Nosotros no disponemos del programador especifico para poder modificar dichos parámetros a nuestro gusto, por tanto, vamos a intentar obtener el valor de las características principales y comprobar si podemos ajustarnos a ellos para evitar tener que adquirirlo.

A continuación se muestran los parámetros que podemos modificar en esta controladora (divididos por categorías):

Parámetros de aceleración.

1. El tiempo que tardará la controladora en pasar del 0 al 100% de aceleración.
2. El tiempo que tardará la controladora en pasar de un 100% de alimentación en la armadura a un 0% (tiempo que tardará el vehículo en frenar).
3. El resto de parámetros son para programar el freno de campo (freno que se activa a velocidades bajas para aumentar la frenada).

Parámetros de velocidad.

1. El límite de velocidad del vehículo.
2. El límite de velocidad del vehículo marcha atrás.
3. Los otros parámetros son para configurar el sensor Hall del motor.

Parámetros de pedal del acelerador.

1. Tensión mínima a la entrada de la controladora para que empiece a moverse el vehículo.
2. Tensión a partir de la cuál se supone una señal del 100% de aceleración.
3. La forma de la respuesta ante el aumento de la señal de aceleración (figura 3.15).

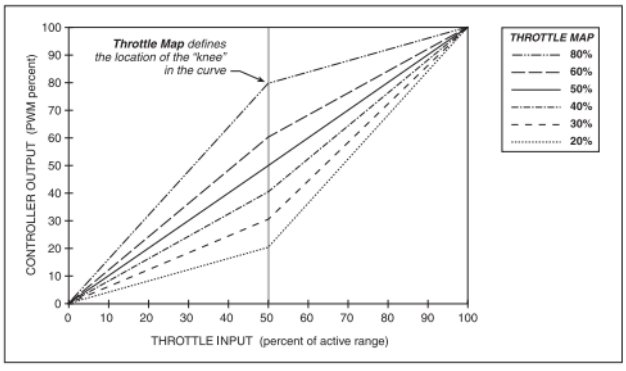

Figura 3.15: Respuesta al acelerar

4. El restro de parámetros son para marcar fallo si la tensión es menor o mayor que un cierto valor.

Parámetros de limitación de corriente.

Con estos parámetros modificamos la corriente que la controladora proporcionará al motor en distintas situaciones de conducción.

Parámetros de freno.

Estos parámetros determinan el comportamiento del freno regenerativo.

Parámetros de campo.

Con estos parámetros se determina cuanta corriente de campo se aplica ante una determinada corriente de la armadura.

Parámetros de fallo.

Estos parámetros están pensados para proteger el vehículo y mostrar avisos.

Trás realizar las pertinentes pruebas destacan los siguientes comportamientos:

1. Aumentar la salida PWM de Arduino a razón de 40 por segundo, es decir, a razón de 0.78 V/s, provoca que salte una protección y el vehículo se pare. Esto se puede observar en la figura 3.16.

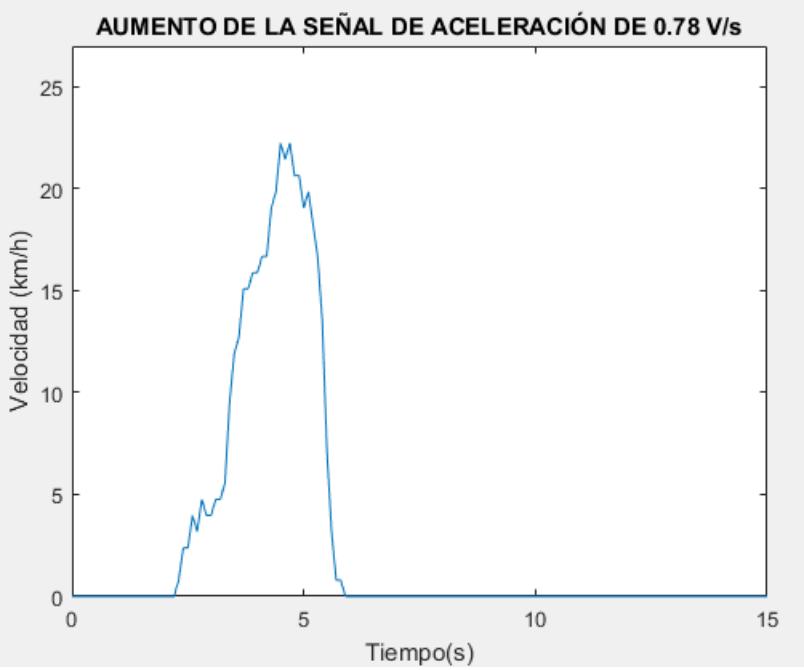

Figura 3.16: Aceleración máxima

El vehículo se se ha detenido antes de alcanzar los 25 km/h.

Partiendo de los puntos marcados en la figura 3.17 podemos calcular cuál es la aceleración que experimenta en el momento de su detección.

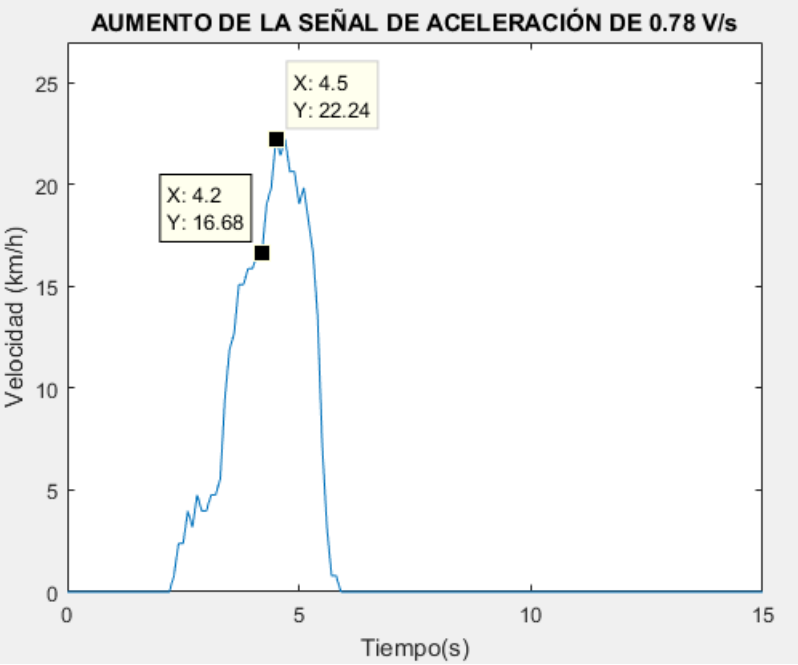

Figura 3.17: Aceleración máxima con anotaciones

$$22.24\ km/h\ -\ 16.68\ km/h\ =\ 5.56\ km/h\ =\ 1.54\ m/s$$

$$\frac{1.54\ m/s}{4.5\ s\ -\ 4.2\ s} = 5.13\ m/s^2$$

La aceleración de comfort es bastante subjetiva pero en este trabajo se toma como límite $3\ m/s^2$.

La controladora permite una aceleración superior de la máxima que nos hemos autoimpuesto ($5.13\ m/s^2$ frente a $3\ m/s^2$.), por lo que éste parámetro no nos supondrá un inconveniente.

2. También es interesante saber como se ha programado la forma de la respuesta ante un aumento constante de la señal de aceleración.
   Para esta prueba se ha aumentado la salida de Arduino a razón de 2 por segundo (0.04 V/s), y se obtiene la respuesta mostrada en la figura 3.18.

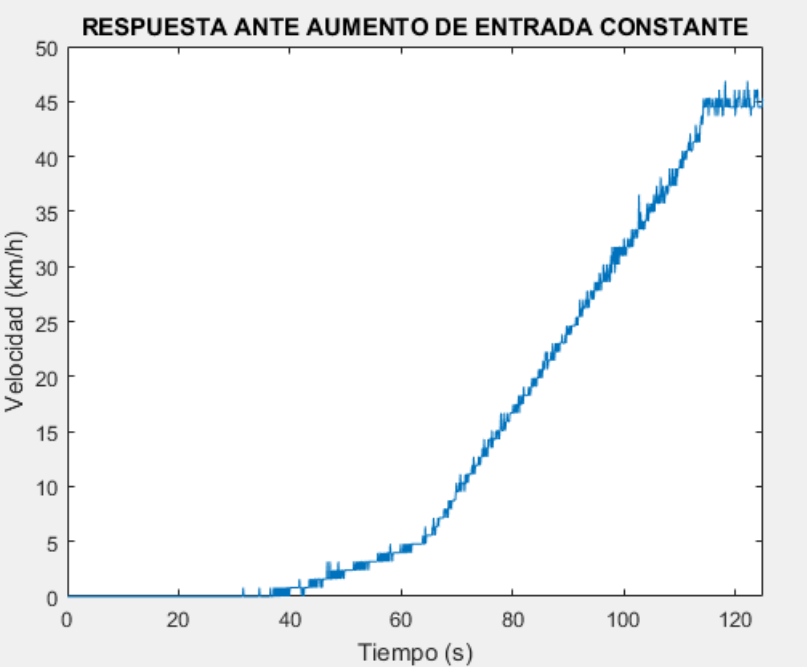

Figura 3.18: Forma de la respuesta ante un aumento de tensión constante

   Un incremento de tensión constante en la entrada supone una mayor aceleración a velocidades más elevadas (el cambio en la respuesta se produce en un codo localizado a la mitad del rango de acción).

3. En las simulaciones anteriores se puede observar la existencia de un rango muerto por debajo del cual el vehículo no comienza a moverse (otro de los parámetros que se podían programar en la controladora).

3.3 IMPLEMENTACIÓN.

Ya tenemos las entradas y salidas del sistema de aceleración funcionando de forma correcta, solo resta programar el controlador en Arduino y buscar los parámetros del PID que mejor adapten la respuesta a la deseada.

El programa en cuestión se puede consultar en el **ANEXO III: CÓDIGOS PARCIALES DEL CONTROL DE VELOCIDAD**.

3.3.1 Montaje del circuito.

El esquema del montaje eléctrico queda de la forma mostrada en la figura 3.19.

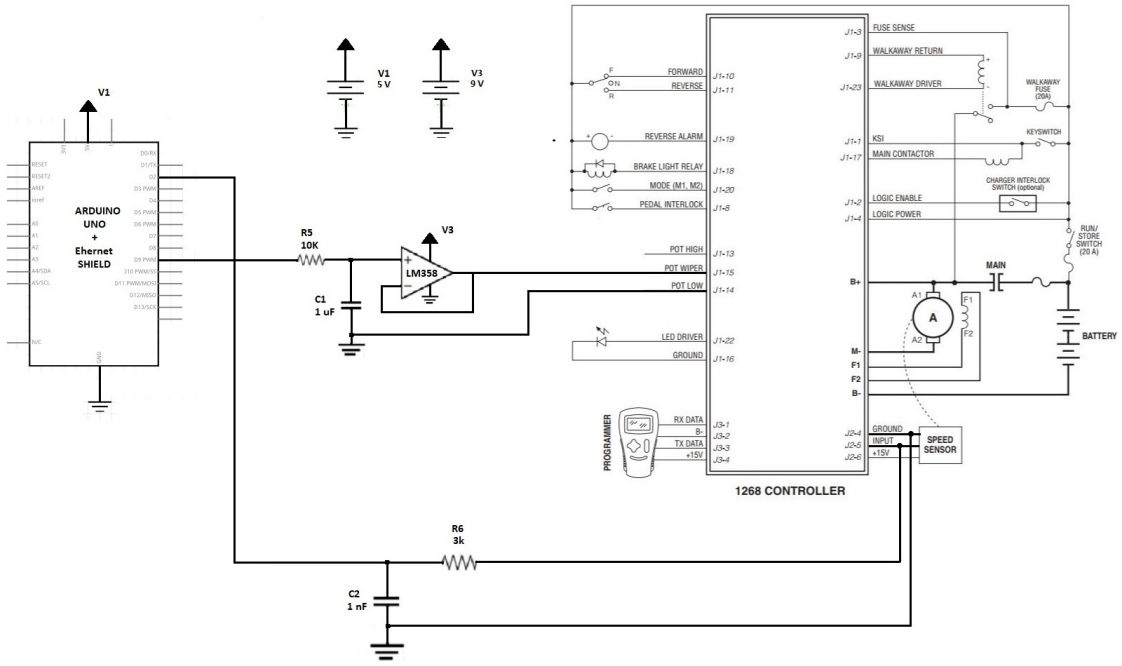

Figura 3.19: Esquema eléctrico subsistema de aceleración

En la figura 3.20 se observa como ha quedado el circuito al implementarlo en el laboratorio para una de las pruebas.

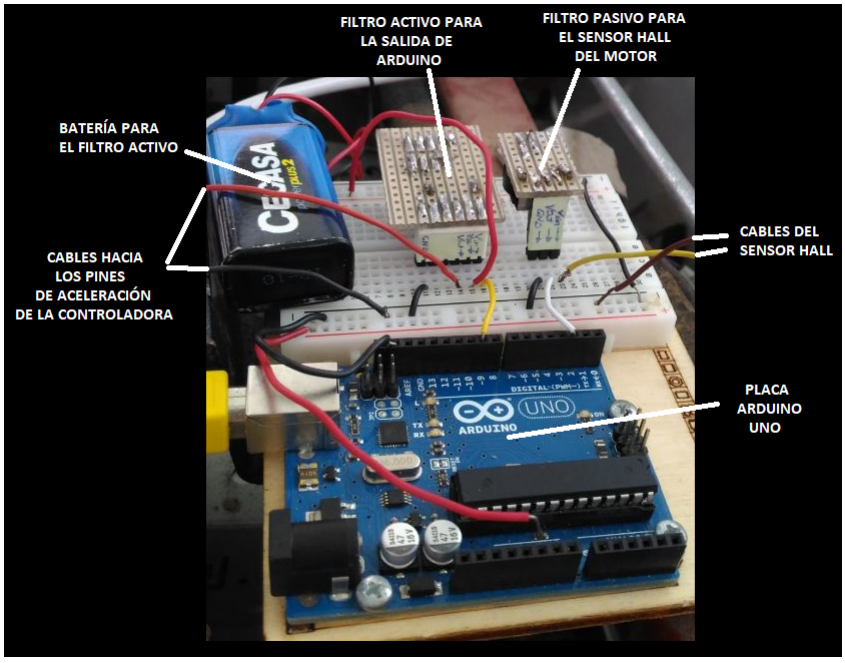

Figura 3.20: Circuito físico

3.3.2 Parámetros del controlador.

Como estamos en una fase de pruebas inicial el vehículo se encuentra elevado del suelo, además se ha tomado como velocidad máxima 40 km/h (no queremos que el vehículo sobrepase dicho valor).

A la hora de ajustar el PID y, debido a la fuerte no linealidad del sistema y a la ausencia de un modelo matemático, se ha recurrido a un ajuste manual persiguiendo dos objetivos:

1. Mantener una aceleración de comfort (requisito previo).
2. Evitar que salten las protecciones por aceleración elevada.

Se ha programado en MATLAB un escalón de 40 km/h (velocidad máxima que queremos para el vehículo) y, tras varias modificaciones, hemos llegado a unos parámetros que, ademas de cumplir los objetivos anteriores, nos evitan la necesidad de incluir la acción derivativa del PID:

$$K_P = 0.5$$

$$K_I = 0.1$$

$$K_D = 0$$

El controlador es por tanto de tipo PI.

En la figura 3.21 se muestra la respuesta del vehículo ante una entrada escalón de 40 km/h con los parámetros escogidos.

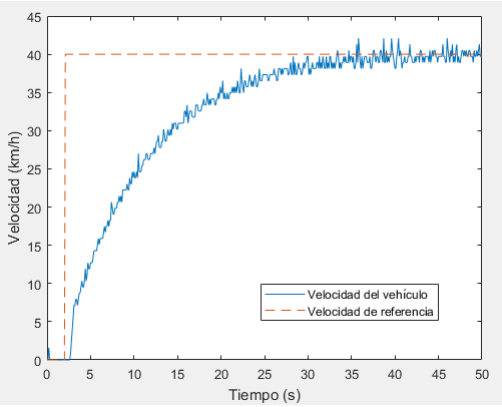

Figura 3.21: Respuesta con los parámetros del PID escogidos.

La máxima aceleración se produce al inicio y su valor viene dado por los puntos mostrados en la figura 3.22.

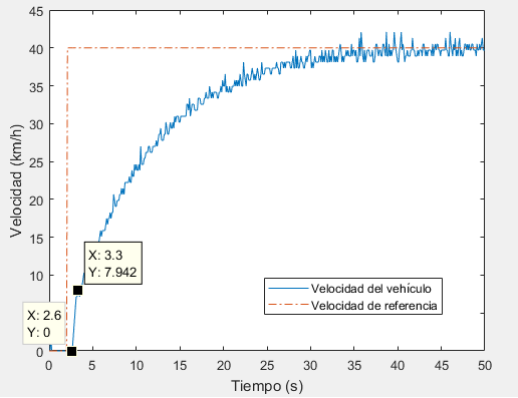

Figura 3.22: Respuesta con los parámetros del PID escogidos con anotaciones.

$$7.942\ km/h\ =\ 2.2061\ m/s$$

$$\frac{2.2061\ m/s}{3.3\ s\ -\ 2.6\ s} = 3.1516\ m/s^2$$

La aceleración excede por poco el límite marcado previamente, además, hay que tener en cuenta que la aceleración decrece con el paso del tiempo, por lo que si seguimos disminuyendo la aceleración máxima admisible la respuesta será demasiado lenta.

*Para comprobar como establecer la comunicación entre Arduino y Matlab dirijirse al **ANEXO II: COMUNICACIÓN ARDUINO-MATLAB**.

Capítulo 4

SUBSISTEMA DE FRENADO.

Ahora que hemos visto el modo de regular la velocidad haciendo uso del subsistema de aceleración nos centramos en un subsistema no menos importante, el de frenado.

El conjunto de motor, controladora y baterías nos proporciona un freno regenerativo cuyas propiedades podemos variar a través de la controladora, no obstante, en circulación de carretera, existirán situaciones en las que este tipo de freno no sea suficiente:

- Cuando necesitemos un frenado rápido.
- Cuando las baterías estén completamente cargadas (el freno regenerativo no estará disponible).
- ...

Otro punto en contra del freno regenerativo es que, como se comentó previamente, no disponemos del programador de la controladora para ajustar sus parámetros a nuestro gusto.

Para solventar estos problemas se dispone de un sistema de frenado específico para conducción autónoma que veremos a lo largo de éste capítulo.

4.1 INTRODUCCIÓN.

Nuestro vehículo experimental es capaz de desarrollar una velocidad máxima de, aproximadamente, 50 km/h. Si bien puede parecer una velocidad no demasiado elevada (en comparación con la que pueden alcanzar los vehículos que estamos acostumbrados a ver por las calles), no deja de ser una velocidad peligrosa.

Para hacerse una idea rápida del peligro que supone un impacto a una determinada velocidad, se suele recurrir a compararlo con una una caída libre de resultados análogos.

Basta igualar la energía cinética del vehículo a la velocidad deseada con su energía potencial a una determinada altura (no deja de ser una aproximación ya que, además de no tener en cuenta el rozamiento del aire, se supone un impacto frontal contra un elemento sólido inamovible):

$$\mathbf{E_C = E_P}$$

$$\mathbf{\frac{1}{2}\, m\, v^2 = m\, g\, h}$$

$$\mathbf{h = \frac{1}{2\, g}\, v^2} \;\rightarrow\; [m] = \frac{1}{\left[\frac{m}{s^2}\right]} \left[\left(\frac{m}{s}\right)^2\right]$$

Para poder expresar la velocidad en km/h tenemos en cuenta que:

$$\left[\left(\frac{km}{h}\right)^2\right] = \frac{\cancel{km^2}}{\cancel{h^2}} \frac{1000^2\, m^2}{1\, \cancel{km^2}} \frac{1\, \cancel{h^2}}{3600^2\, s^2} = \frac{25}{324}\left[\left(\frac{m}{s}\right)^2\right]$$

Por lo tanto:

$$\mathbf{h} \approx \frac{\mathbf{1}}{\mathbf{2 \cdot 9{,}8}}\, \mathbf{v^2}\, \frac{\mathbf{25}}{\mathbf{324}}$$

$$\mathbf{h}\ [m] \approx \mathbf{0{,}0039\, v^2}$$

Si evaluamos la expresión anterior para varias velocidad nos encontramos con una gráfica como la mostrada en la figura 4.1 [27] (velocidades en km/h y altura en metros).

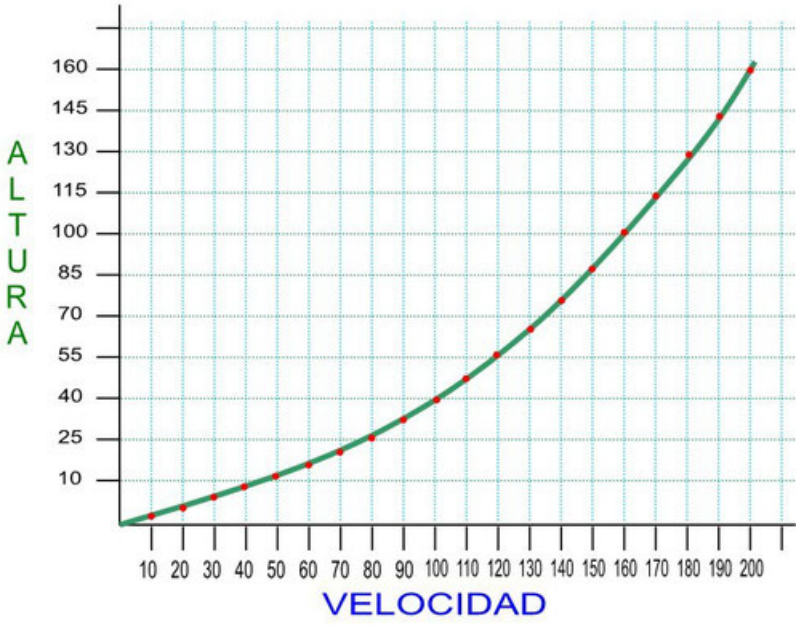

Figura 4.1: Impacto frontal de un vehículo

Un impacto a la velocidad máxima de nuestro vehículo se podría comparar con una caída líbre de unos 10 metros de altura, aproximadamente el tamaño de un edificio de tres plantas.
Es importante notar que doblar la velocidad supone cuadriplicar la altura.

$$\mathbf{h = k\ v^2}$$

Para poder evitar impactos en cualquiera de sus modos de operación, nuestro vehículo dispone de tres mecanismos de frenado:

1. **Freno regenerativo** (al levantar el pie del acelerador).
2. **Freno convencional** (al accionar el pedal de freno).
3. **Central hidráulica compacta** (al proporcionarla tensión).

Tanto el freno convencional como la central hidráulica compacta tienen como objetivo final el accionamiento de los dispositivos de frenado instalados en las ruedas del vehículo, en nuestro caso **frenos de disco en las ruedas delanteras** y **de tambor en las traseras**.

Esta disposición es bastante común ya que al frenar es el tren delantero el que recibe mayor carga de trabajo, dejando los frenos de tambor (de menor potencia de frenado) para el eje trasero.
De esta forma se reducen costes y disminuye la posibilidad de bloqueo en las ruedas traseras en caso de frenado brusco.
En la figura 4.2 se observa gráficamente lo indicado anteriormente.

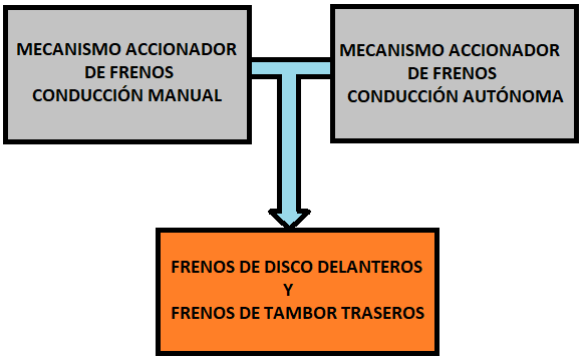

Figura 4.2: Accionamiento de los dispositivos de frenado.

La ventaja de tener ambos sistemas de frenado en paralelo radica en el hecho de que el conductor no pierde la posibilidad de accionar el freno, aún en conducción autónoma, ya que prevalece la presión más elevada.

4.2 MÉTODOS DE FRENADO.

4.2.1 Freno regenerativo.

El freno regenerativo entra en funcionamiento al disminuir la tensión presente en los correspondientes pines de la controladora, bien al reducir la presión ejercida en el pedal de freno en conducción manual (a través del potenciómetro al que se encuentra conectado), o bien al reducir la tensión directamente en conducción autónoma (en nuestro caso con la salida de Arduino).

Cuando esto pasa, el motor, por inercia, tendrá una fuerza contraelectromotriz más elevada que la tensión de alimentación, esto provoca que el sentido de corriente se invierta recargando las baterías hasta que las tensiones se vuelvan a igualar.

4.2.2 Frenos en conducción manual.

La configuración del mecanismo de freno para conducción manual es de lo más sencillo que podemos encontrarnos en un vehículo de estas características.

Cuando el conductor oprime el pedal de freno acciona el vástago del cilindro maestro provocando que el líquido de frenos en el interior del circuito de frenado aumente de presión.
Este aumento de presión provoca que los frenos de disco de las ruedas delanteras, y los de tambor de las traseras, comiencen a trabajar transformando energía cinética en térmica y reduciendo, por tanto, la velocidad del vehículo (figura 4.3 [28]).

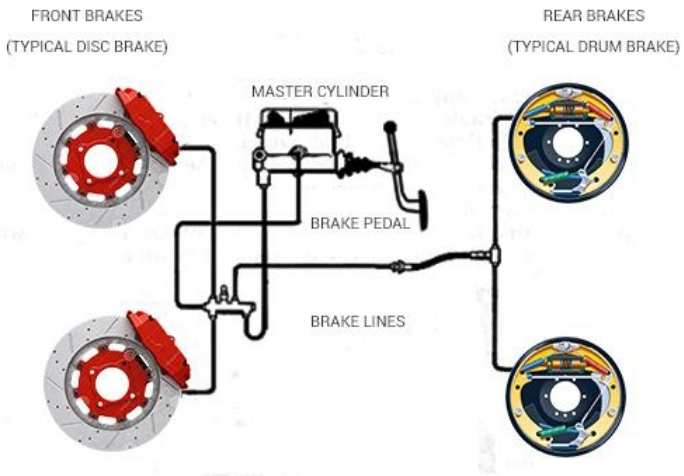

Figura 4.3: Freno en conducción manual

Como se ha comentado el mecanismo es muy sencillo, ni siquiera cuenta con elementos comunes en la inmensa mayoría de vehículos homologables como puede ser el servofreno (mecanismo que minimiza la fuerza de frenado que debe ejercer el piloto), o de seguridad como el ABS (sistema antibloqueo de ruedas), o el ESP (sistema de control de estabilidad).
Esto se debe principalmente a dos razones:

1. El vehículo está pensado para uso experimental.
2. Debido a su peso y rango de velocidades no es extrictamente necesario incluir estos elementos.

4.2.3 Frenos en conducción autónoma.

El sistema de accionamiento de frenos para conducción autónoma (en paralelo con el mecanismo de freno convencional), es una central compacta marca ”HAWE Hydraulik” del tipo NPC (figura 4.4), que será la encargada de aumentar la presión en el circuito de frenos en este tipo de conducción.

Figura 4.4: Central hidráulica compacta HAWE NPC

Una central hidráulica compacta es un sistema capaz de aumentar la energía de un fluido incompresible a partir de energía mecánica. La energía mecánica se obtiene, por normal general, de un motor diesel o eléctrico.
En este tipo de centrales es común que tanto el motor como la bomba compartan eje, lo cual permite una considerable reducción en tamaño y peso.

En la figura 4.5 se muestra la central hidráulica compacta ya instalada en el vehículo.

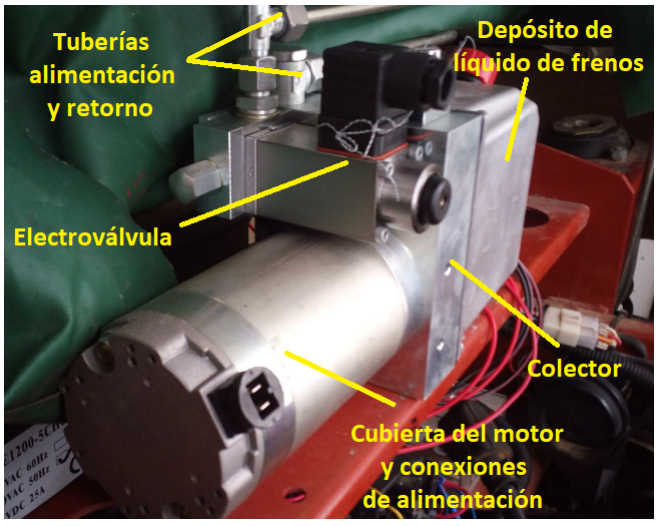

Figura 4.5: Central hidráulica compacta del vehículo experimental

4.3 CENTRAL HIDRÁULICA COMPACTA.

Al ser el elemento clave para accionar los frenos en conducción autónoma, se va a emplear esta sección para analizar sus componentes y su conexión [29].

4.3.1 Partes principales.

En la figura 4.6 se recojen las partes principales de nuestra central hidráulica compacta.

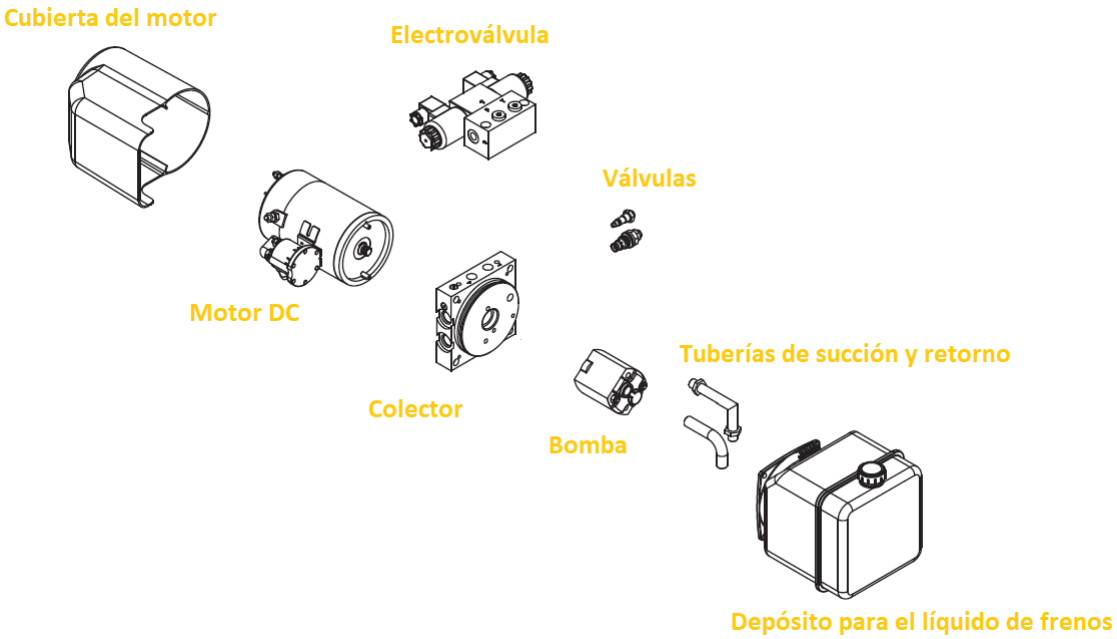

Figura 4.6: Componentes principales de la central hidráulica compacta

Motor DC.

En nuestro caso el elemento que hace girar el eje que acciona la bomba es un motor de corriente continua de 24V.

Bomba.

La bomba es la encargada de aplicar presión al líquido de frenos, en nuestro caso es de engranajes.

En las bombas de engranajes, el eje del motor mueve uno de los engranajes provocando el movimiento del otro en sentido contrario, esto provoca que en la zona donde se separan los dientes (entrada) se produzca una depresión que fueza al fluido del tanque a rellenar el nuevo espacio.

Este fluido se moverá a la par que los engranajes por la parte exterior de la cámara de la bomba.

Al llegar a la zona donde los dientes vuelven a engranar, el fluido no puede seguir avanzando con ellos, esto provoca que sea expulsado hacia la tubería de salida (figura 4.7 [30]).

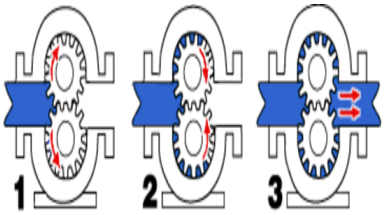

Figura 4.7: Bomba de engranajes

Electroválvula.

Esta válvula es la encargada de aplicar o liberar presión al circuito de frenado, para accionarla simplemente la alimentamos a su tensión nominal, en nuestro caso 24V.

En la imagen observamos la electroválvula ya instalada en la central, podemos observar que es una válvula 3/2 accionada por un solenoide con retorno a la posición de reposo mediante muelle.

Se dice que una válvula es 3/2 cuando puede trabajar en dos posiciones, cada una de las cuales altera de distinta manera la configuración de las tres entradas/salidas.

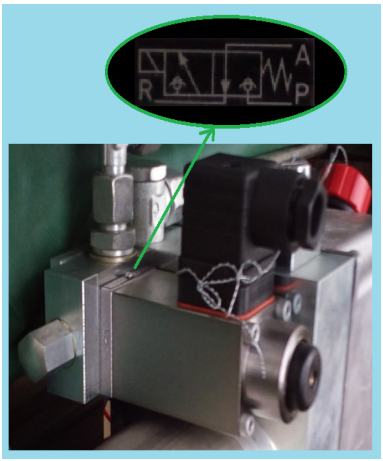

Figura 4.8: Electroválvula

En la figura 4.8 se muestra la electroválvula del vehículo experimental.

Se pueden observar tres tuberías: "A" es la tubería que alimenta al circuito de frenado, "P" es la tubería que sale de la bomba, y "R" es la tubería de retorno al depósito.

4.3.2 Esquema de conexión.

Por lo visto podemos afirmar que el conjunto motor/bomba de la central compacta nos permite elevar la presión del líquido de frenos del depósito.
Además, si la electroválvula 3/2 está activada, dicha presión pasa al circuito de frenado accionando los frenos de las ruedas. En caso contrario permite liberar presión del circuito hacia el depósito de líquido de frenos.
Para proteger la central, a la salida de la bomba existen dos válvulas:

- Válvula antirretorno.
  En caso de que la electroválvula este activada y la presión en el circuito de frenado sea superior a la generada por la bomba, protege la bomba impidiendo el paso hacia la misma.

- Válvula de alivio.
  Si por alguna razón se genera una presión demasiado elevada para nuestro circuito a la salida de la bomba, esta válvula permite reducir la presión en la central compacta enviando el líquido de frenos al tanque de líquido de frenos principal.

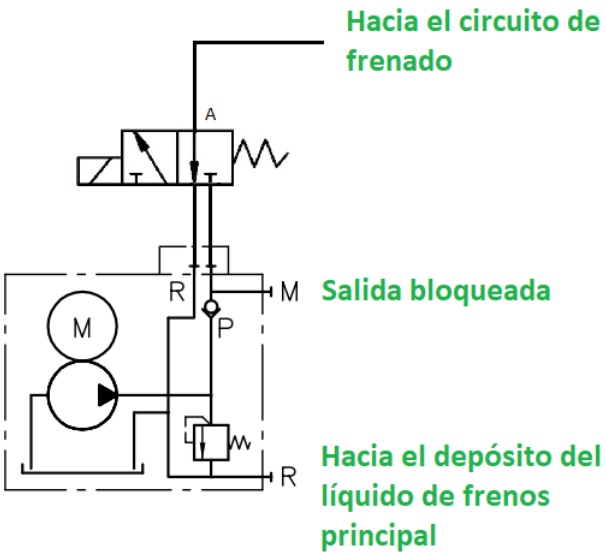

Figura 4.9: Esquema de conexión central compacta

4.4 AJUSTES INICIALES.

En esta sección se comprueba el correcto funcionamiento de la central hidráulica compacta y de la electroválvula, además se indica como se regulará la presión en el circuito de frenos.

4.4.1 Cableado.

Tanto las baterías, como el motor de la central hidráulica compacta, disponen de bornes accesibles por lo que no suponen un problema a la hora de realizar el cableado.

Por otra parte, necesitamos sacar las conexiones de la electroválvula.
El conector de la misma esta pensado para poder aislar los cables al insertar un conector de dimensiones adecuadas, no obstante, para facilitar las pruebas, se han realizado unas conexiones provisionales.
En la siguiente imagen se muestra el despiece del conector y su conexionado.

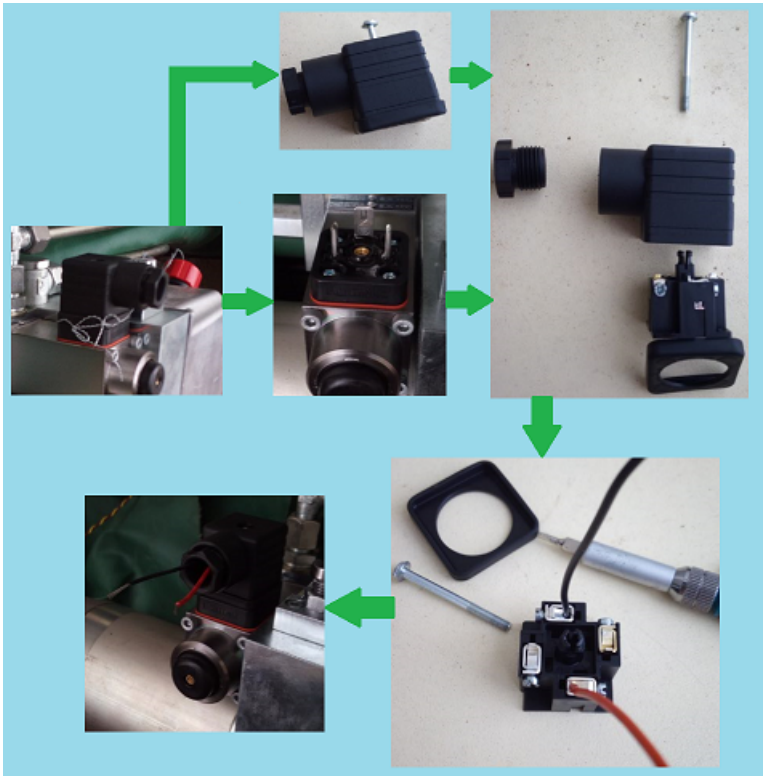

Figura 4.10: Conexiones de la electroválvula

4.4.2 Alimentación.

Para la alimentación del motor y de la electrovalvula vamos a utilizar las baterías del propio coche.
Con 4 de las baterías del cuadriciclo en serie obtendremos, teoricamente, 24V justos, tensión que se corresponde con la nominal de funcionamiento del motor y de la electrovalvula.

En caso de poder aislar el circuito de control del circuito de alimentación de la central compacta (figura 4.11), podríamos coger 4 baterías cualesquiera como fuente de alimentación.

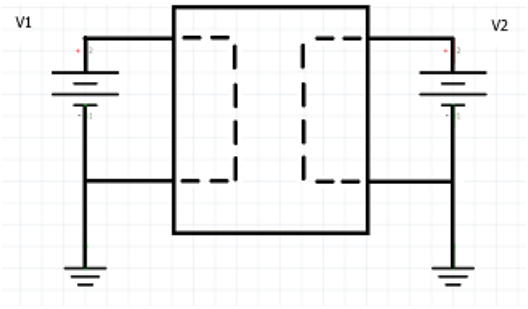

Figura 4.11: Circuitos aislados

En el caso de que se escoja un circuito con tierra común para el circuito de control y de alimentación de la central compacta [figura 4.12], será necesario obtener la tensión de alimentación de las 4 primeras baterías.
Esto se debe a que las señales empleadas en el subsistema de aceleración (tanto la señal de aceleración como la señal del sensor Hall), están referenciadas a la masa de la primera batería, asimismo, al unificar las masas, reducimos potenciales riesgos de cortocircuito por lo que el sistema global será más seguro.

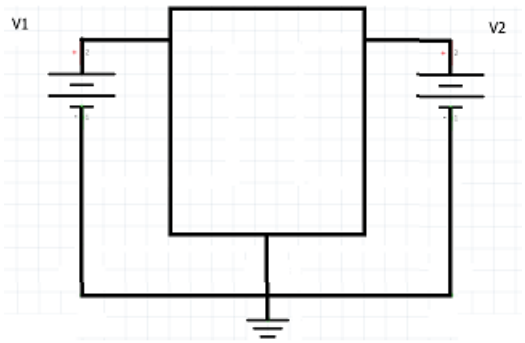

Figura 4.12: Circuitos con masa común

Por lo comentado se emplean las 4 primeras baterías.

4.4.3 Comprobación de funcionamiento a tensión nominal.

Tanto la selección de la central hidráulica compacta como su instalación se han realizado de forma previa a este trabajo.
Se da por hecho que a tensión nominal la fuerza de frenado será máxima y suficiente para detener el vehículo de forma satisfactoria.

En la primera prueba, tras llenar el depósito de líquido de frenos, se procedió a alimentar tanto la central hidráulica compacta como la electroválvula a sus tensiones nominales para comprobar su correcto funcionamiento.
Se comprobó lo siguiente:

- Motor ON, electroválvula ON.
  Las ruedas se bloquean.
- Motor ON, electroválvula OFF.
  Las ruedas no se bloquean.
- Motor OFF, electroválvula ON.
  Las ruedas sólo estarán bloqueadas si en la situación previa también los estaban.
- Motor OFF, electroválvula OFF.
  Las ruedas no se bloquean.

El funcionamiento es por tanto el esperado.

De lo anterior se deduce que las ruedas solo se bloquean cuando tanto el motor como la electroválvula estén alimentados.

Asimismo se observa que una vez bloqueadas las ruedas, la única forma de desbloquearlas es eliminar la tensión de la electroválvula.

4.4.4 Variación de presión en el circuito de freno.

Para poder regular la presión en el circuito de frenado necesitamos realizar un control de velocidad sobre el motor de corriente continua de la central hidráulica compacta y, además, habrá que controlar el estado de la electroválvula.

La velocidad del rotor de un motor de corriente continua (suponiendo pérdidas en el hierro y mecánicas nulas y un circuito magnético lineal), viene dada por la siguiente ecuación [4]:

$$n = \frac{E}{k\,\Phi}$$

De esta expresión podemos deducir las opciones que tenemos para realizar un control de velocidad:

1. Regular la tensión aplicada al inducido.
2. Regular el flujo magnético (variando la corriente del inductor).

En este caso, por sencillez, se ha escogido la primera opción.

Resumiendo, tenemos que conseguir variar la tensiòn a la entrada del motor de la central compacta partiendo de una alimentación fija de 24 V (que hemos obtenido de las baterías del propio coche), mediante una de las salidas PWM de Arduino.
En el caso de la electroválvula, como solo buscamos dos estados (encendido o apagado), podemos usar una de las salidas digitales de Arduino.

Los microcontroladores trabajan con corrientes muy reducidas, en el caso de Arduino el máximo de corriente que podemos extraer o suministrar en cada pin es de 40 mA (se recomienda no pasar de 20 mA), no obstante, en la hoja de características de la central compacta se nos muestra que la corriente nominal puede llegar a alcanzar 22 A, en el caso de la electroválvula hablamos de 1.6 A (estos niveles de corriente destruirán el microcontrolador de manera inmediata).

Cuando necesitamos controlar un circuito de corrientes elevadas mediante dispositivos electrónicos entramos en el campo de la electrónica de potencia [31].

Se va a emplear lo que se denomina un circuito chopper.
Estos circuitos funcionan como interruptores controlados electrónicamente, permitiendo aislar la señal de control del circuito principal.

Podemos obtener un sencillo circuito chopper haciendo uso de un único transistor MOSFET de potencia (enriquecimiento).
Para evitar consumos innecesarios se trata de evitar en la medida de lo posible la zona lineal, es decir, idealmente trabaja únicamente en las zonas de:

Corte.

Si la tensión puerta-fuente no excede la tensión umbral del transistor, nos encontramos en zona de corte.
El transistor funciona como un interruptor abierto impidiendo el paso de corriente a través del mismo.

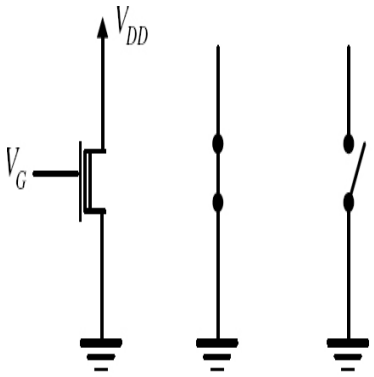

Figura 4.13: MOSFET como interruptor

Saturación.

Si la tensión puerta-fuente es mayor a la tensión umbral del transistor y, además, la tensión drenador-fuente es mayor que la diferencia entre la tensión puerta-fuente y la tensión umbral, entonces nos encontramos en la región de saturación.
En esta zona el transistor funcionará como un interruptor cerrado, es decir, permite el paso de corriente.

Un interruptor MOSFET ideal (figura 4.13 [32]) tiene una resistencia o bien nula o bien infinita, por tanto su consumo será nulo [33].
En la realidad, cuando el MOSFET trabaja en saturación, si existe una resistencia equivalente entre bornes que tendremos que tener presente a la hora de elegir entre uno u otro.
Una corriente elevada atravesando esta resistencia (aunque sea pequeña), puede producir una potencia por encima del rango de disipación del MOSFET que acabe destruyéndolo.

En la siguiente imagen se muestra la configuración tanto para la central compacta como para la electroválvula si se elige este circuito chopper.

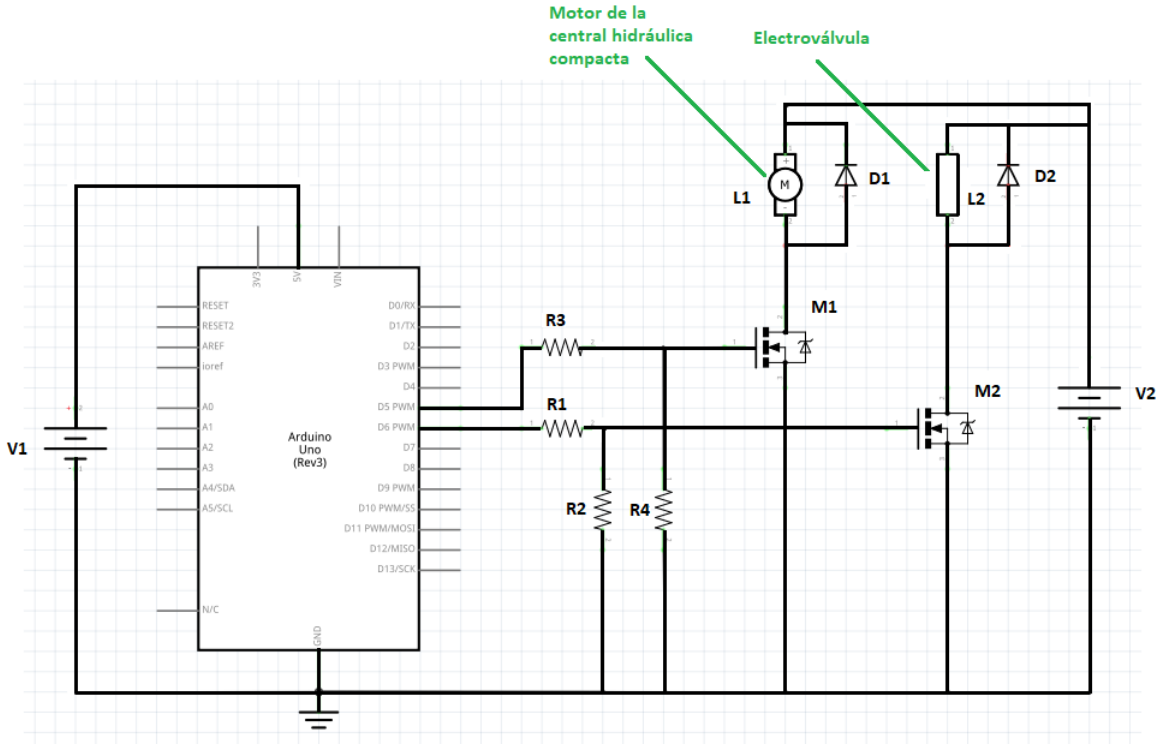

Figura 4.14: Circuito de control para freno en conducción automática

Donde V1 representa la fuente de alimentación de Arduino, V2 los 24 V que hemos sacado de las baterías del vehículo, M1 y M2 los MOSFET de control, la carga L1 es el motor de la central hidráulica compacta y la carga L2 la electroválvula.

Se pueden observar un par de añadidos no comentados previamente, pero necesarios para el correcto funcionamiento del circuito:

- Las resistencias R1 y R3.
  Los transistores MOSFET son dispositivos controlados por tensión, no obstante, en las conmutaciones pueden llegar a demandar una corriente excesiva para nuestro microcontrolador (en ese instante se comportan como un condensador).
  Por medio de estas resistencias nos aseguramos de que no se excedan los límites, con un valor de 220 Ω la corriente máxima será de poco mas de los 20 mA recomendados.

- Las resistencias R2 y R4.

  Estas resistencias evitarán que la puerta del mosfet se quede flotando cuando Arduino se conecte/desconecte o cuando carguemos programas (los pines de Arduino suelen pasar por un estado de alta impedancia).

  Si la puerta se queda flotando podría darse el caso de que las cargas se activen sin ser esa nuestra intención.

  Se selecciona un valor de 10 kΩ que, comparado con los 220 Ω, es un valor elevado que no disminuirá notablemente la tensión en la puerta del MOSFET, pero por otro lado, es un valor despreciable comparado con los aproximadamente 100 MΩ que supone un pin de Arduino en estado de alta impedancia (poniendo en este caso la puerta del MOSFET a tierra).

- Los diodos D1 y D2.

  Estos diodos se conocen como diodos flyback.

  Son necesarios sobre todo cuando tenemos cargas inductivas (como es nuestro caso), ya que este tipo de cargas se oponen al cambio de corriente que las atraviesa, generando un pico de tensión que puede destruir facilmente el MOSFET.

  Con ayuda de estos diodos creamos un circuito aislado para que la carga inductiva pueda disminuir su corriente de manera gradual y, por tanto, evitando los picos de tensión.

La mayor parte de MOSFET de potencia no pueden ser usados como interruptor si la señal de control proviene de un microcontrolador, esto se debe principalmente a dos razones:

1. Los 5V de tensión máxima que proporciona Arduino no son suficientes para poner a trabajar al transistor en la zona de saturación.
2. El nivel de corriente demandado por las puertas de los transistores en las conmutaciones es demasiado elevado para los pines de un microcontrolador.

Lo anterior nos obliga a usar un driver MOSFET, estos circuitos permiten aumentar el nivel de tensión a la salida de Arudino y suministrar una corriente mayor a la que puede proporcionar el microcontrolador, permitiendo que el condensador de las puertas de los mosfet se cargue mucho más rápido.

Existen no obstante unos transistores MOSFET especiales, denominados MOSFET de nivel lógico, que están pensados para interactuar directamente con la salida de un microcontrolador, como contrapartida suelen ser más lentos y presentar una resistencia de conducción mayor.

Pese a todo se ha encontrado un MOSFET de nivel lógico que se adapta a nuestra aplicación, el IRLZ44n de INFINEON (empresa adquirida por INTERNATIONAL RECTIFIER), perteneciente a la familia de transistores HEXFET.

1) A temperatura ambiente permite el paso de una corriente continua de 47 A (aproximadamente el doble que la corriente máxima mostrada en las hojas de características de la central compacta, y picos de corriente de 160 A).
2) La tensión drenador-fuente máxima está bastante por encima de nuestros 24V.
En el datasheet se muestran pruebas con frecuencias de 1MHz, más frecuencia de la que vamos a emplear.
3) Con una Vgs de 5V y una corriente de drenador de 20 A la resistencia de conducción en saturación será de 0.022 ohmios.

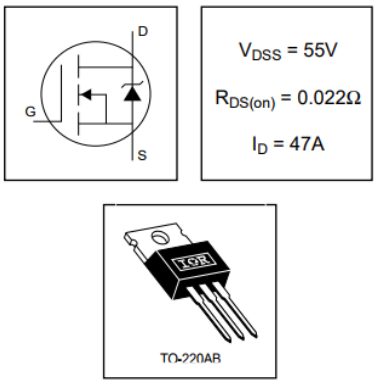

Figura 4.15: IRLZ44n

4.5 IMPLEMENTACIÓN.

Solo resta obtener el software de control.
Se usa como base el código empleado en el subsistema de aceleración añadiendo otra salida PWM, que se activará cuando la señal del PID sea negativa, y que controlará la tensión aplicada a la central hidráulica.
Además, cuando aumente la señal de frenado, debemos activar la electroválvula y, en caso contrario, desactivarla.

Este código se puede observar en el **ANEXO III: CÓDIGOS PARCIALES DEL CONTROL DE VELOCIDAD**.

Capítulo 5

SISTEMA DE CONTROL DE VELOCIDAD FINAL.

5.1 HARDWARE.

El esquema eléctrico final se puede observar en la figura 5.1 y en la figura 5.2 (conectadas entre sí mediante los cables a y b marcados en las mismas).

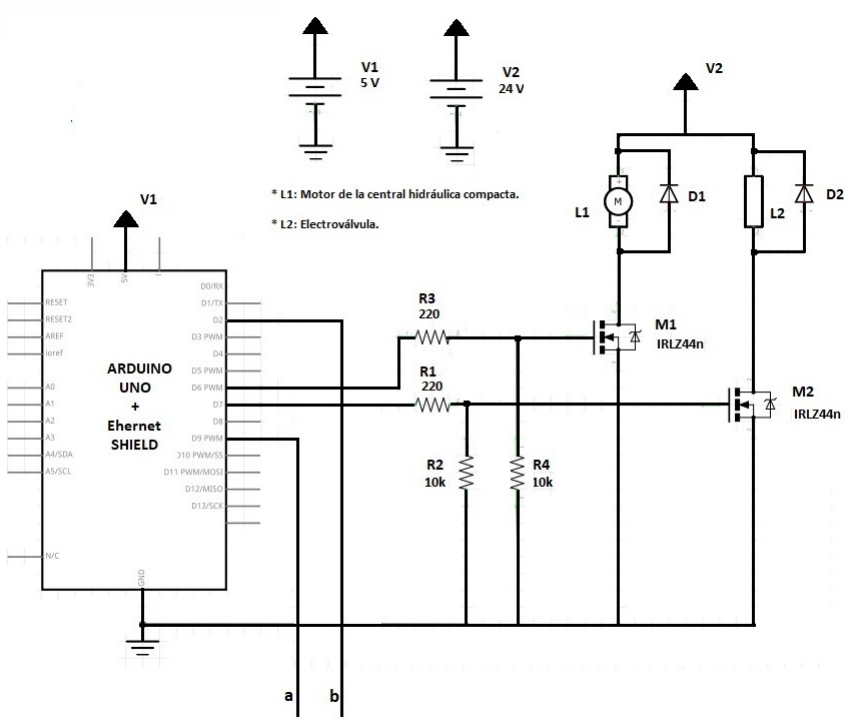

Figura 5.1: Esquema eléctrico final 1/2

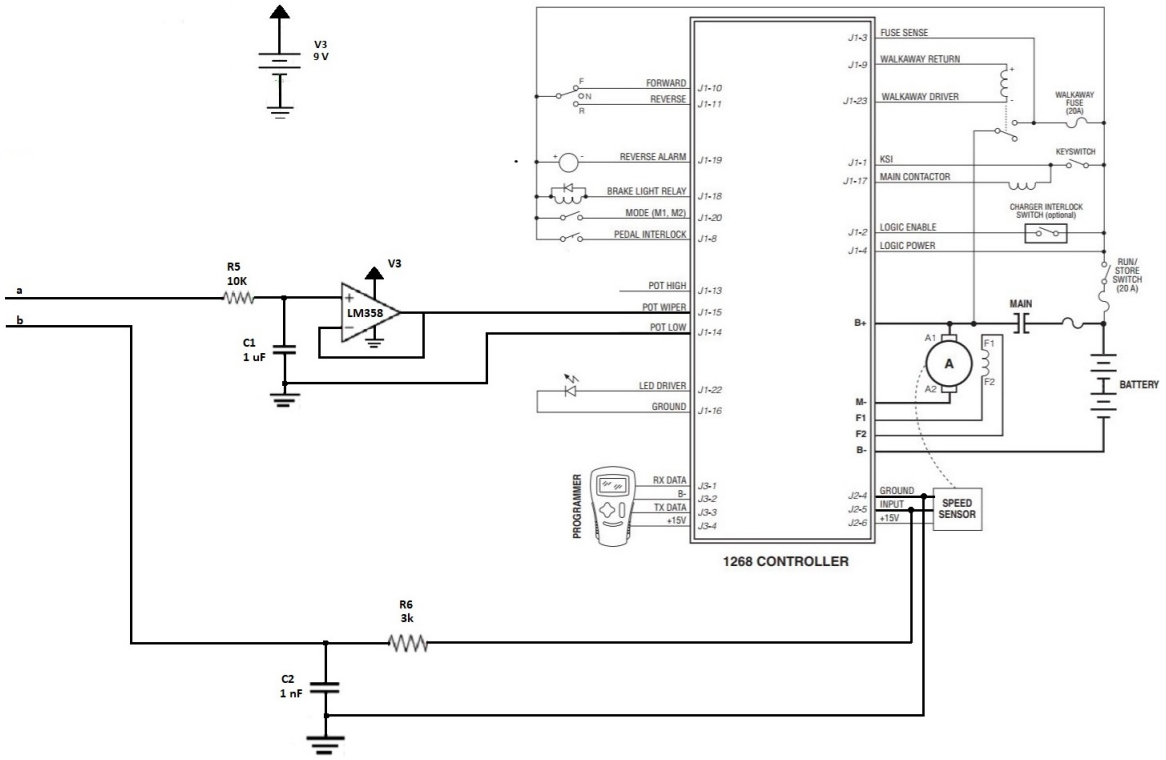

Figura 5.2: Esquema eléctrico final 2/2

5.2 SOFTWARE

Juntando el código de los apartados correspondientes al subsistema de aceleración y al subsistema de frenado obtendremos el control de velocidad final.
Además pasamos la comunicación a ethernet como ya se comentó al inicio del documento.

El código respectivo al control de velocidad final se puede observar en el **ANEXO IV: CÓDIGO DEL CONTROL DE VELOCIDAD FINAL**.

Capítulo 6

CONTINUACIÓN DE LA LÍNEA DE INVESTIGACIÓN.

6.1 MEJORA DEL SISTEMA DE CONTROL DE VELOCIDAD.

El siguiente paso será bajar el coche a tierra, esto provocará un cambio en el modelo del vehículo y la necesidad, por tanto, de un nuevo reajuste de parámetros usando los anteriores como base.

Obtener los parámetros del PID de forma manual es un buen punto de partida, a partir de aquí se necesitará un modelo matemático del vehículo para poder buscar parámetros más eficientes.
Debido a lo complejo del sistema se recomienda obtener el modelo matemático mediante algún método experimental, por ejemplo, mínimos cuadrados para sistemas no lineales.

Una vez se disponga del modelo matemático se podría recurrir a redes neuronales o control difuso para obtener los parámetros del controlador, estos métodos son útiles cuando tenemos sistemas fuertemente no lineales como es nuestro caso.

6.2 TABLERO DE ABORDO.

Como se ha liberado la comunicación serie por USB se propone emplearla para obtener un tablero de abordo digital que muestre al piloto información en tiempo real del vehículo.

Se procede con la realización del mismo.
Se ha supuesto que disponemos de información adicional a parte de la velocidad del vehículo (si se decide instalarlo será facil incluir los sensores oportunos).

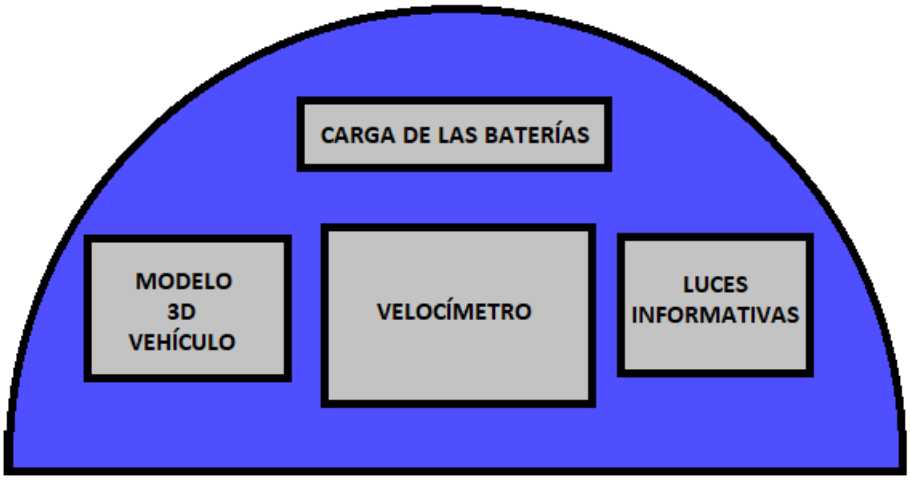

Figura 6.1: Boceto inicial del tablero de abordo

El tablero propuesto (figura 6.1) contiene la información detallada a continuación:

1. **Velocímetro.** Para mostrar la velocidad del coche calculada en este trabajo.
2. **Modelo 3D del vehículo.** Muestra un modelo tridimensional del coche. Se podría mejorar añadiendo al modelo información relevante en tiempo real, por ejemplo, la localización de alguna avería.
3. **Luces informativas y carga de las baterías.** Estos datos se han añadido con vistas a una posible mejora del vehículo (de momento no se dispone de dicha información).

6.2.1 Software empleado.

Se ha decidido hacer uso de PROCESSING [34] para la programación del tablero. Este software, de código abierto y descarga gratuita, nos permite realizar programas interactivos donde las artes visuales adquieren un rol predominante.

La información necesaria para el tablero la enviamos desde ARDUINO haciendo uso del puerto serie (ha quedado libre tras modificar el programa del control de velocidad para usar comunicación ethernet).

Debido al tiempo que requiere dibujar formas complejas tridimensionales en processing se hace uso de BLENDER [35] para tal fin (software libre y de codigo abierto para diseño tridimensional).

En la figura 6.2 se muestra un resumen gráfico de lo que se ha comentado.

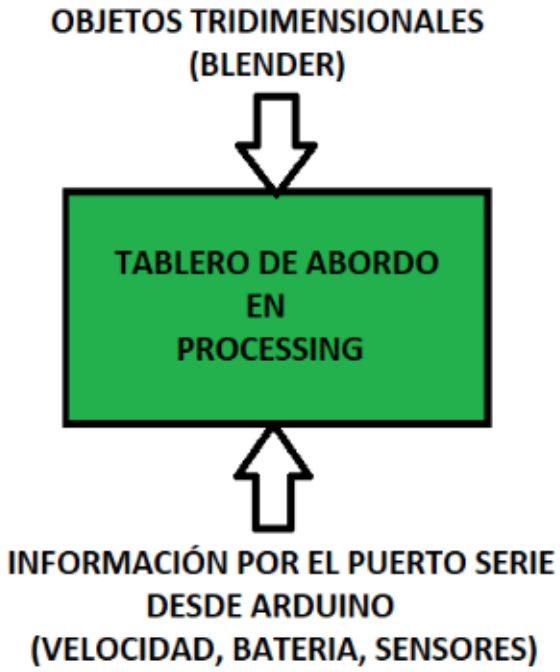

Figura 6.2: Programas para el tablero de abordo

6.2.2 Creación de objetos tridimensionales con BLENDER.

Panel de fondo.

Comenzamos creando un fondo para el tablero (figura 6.3).

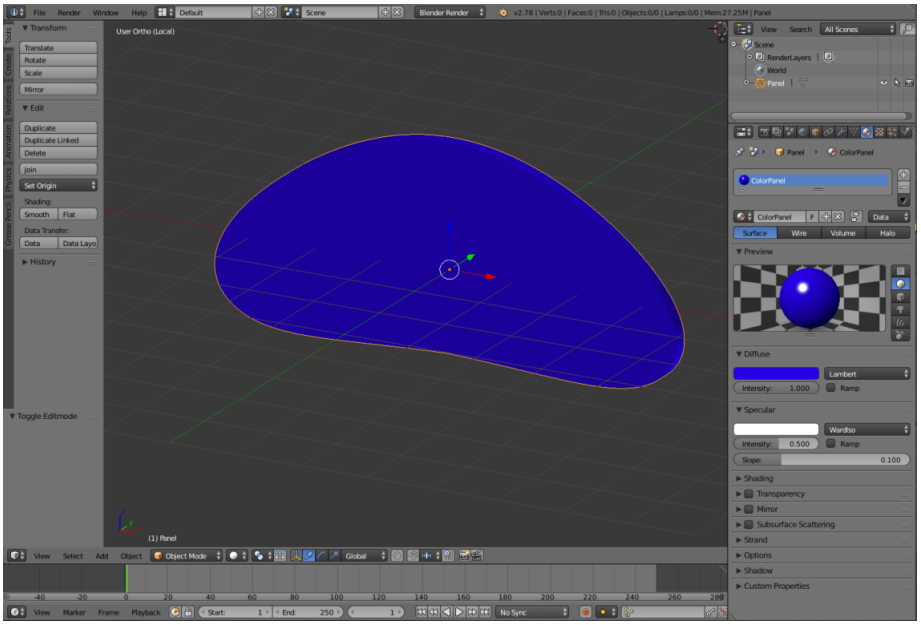

Figura 6.3: Panel de fondo

Velocímetro.

Creamos el velocímetro (partiendo de un cilindro al que posteriormente se han añadido las marcas de velocidad), y le damos color (figura 6.4).

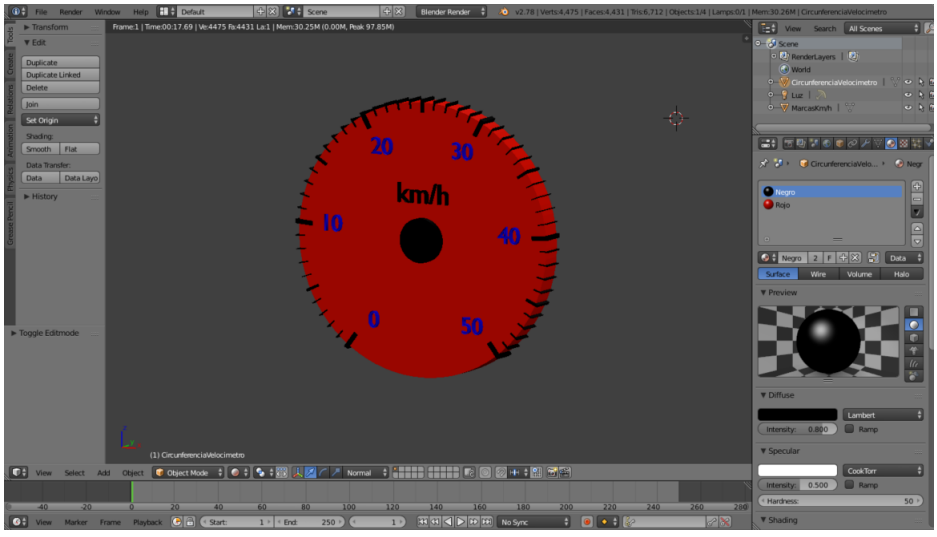

Figura 6.4: Velocimetro

Cuadriciclo 3D.

Para crear el vehículo se ha comenzado dibujando su perfil (figura 6.5).

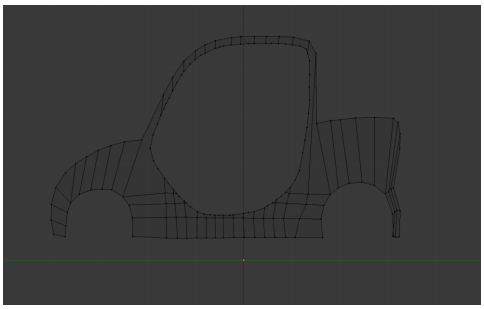

Figura 6.5: Perfil del vehículo

Una vez que tenemos el perfil se extruye, esto nos lleva de un elemento en el plano bidimensional al tridimensional.
Poco a poco se modifica la posición de las distintas líneas que forman la malla, de manera que se ajuste lo máximo posible al objeto que estamos dibujando (figura 6.6).

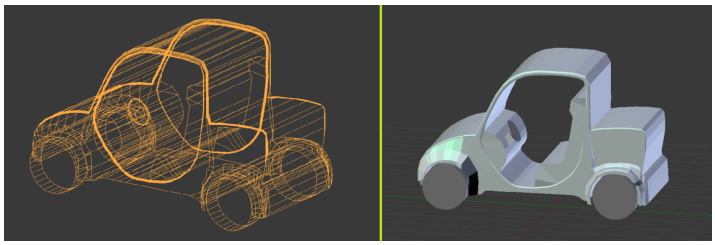

Figura 6.6: Modelo del vehiculo extruido

Para poder añadir imágenes externas al modelo se seleccionan las zonas donde se desean incluir y se hace un mapeo UV bidimensional.
Se puede descargar dicho mapeo y modificar con ayuda de un programa de edición de imagenes (figura 6.7).

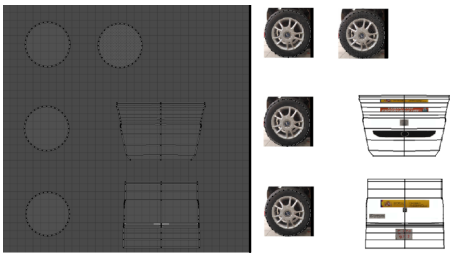

Figura 6.7: UV del modelo del vehiculo

Finalmente se añaden los colores y efectos deseados al resto de zonas del modelo tridimensional (figura 6.8).

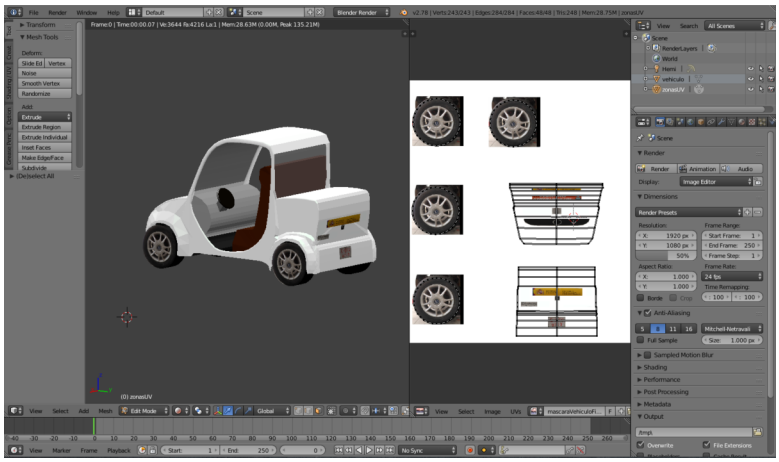

Figura 6.8: Modelo del vehiculo con UV

Ya tenemos el modelo del coche listo para exportar (figura 6.9).

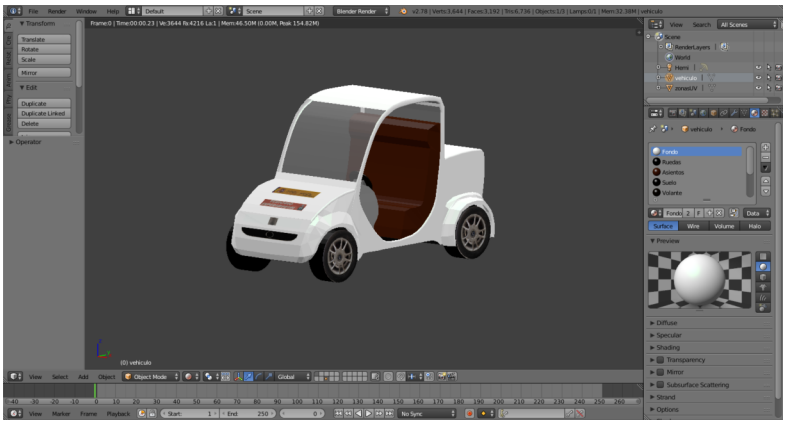

Figura 6.9: Modelo final

Exportar.

Una vez terminados los modelos tridimensionales los exportamos en formato .obj para importar posteriormente en processing.
Nos dirijimos a file, export, wavefront (.obj) y tras seleccionar un nombre y localización para los archivos se nos genera el .obj buscado, y otro archivo importante, el .mtl, que contiene la información referente a colores y texturas básicas usados en el modelo.

6.2.3 Tablero en processing.

Importar.

Para importar los modelos generados en el paso previo debemos de guardar los archivos .obj , .mtl , y las imágenes ligadas al modelo en la carpeta data en la ruta del programa.
Tras esto podemos llamarlos facilmente desde processing generando un elemento del tipo PShape al que asignamos el .obj correspondiente.

Para el caso de las imágenes el proceso es análogo, esta vez se genera un elemento de tipo PImage y se le asigna la imagen correspondiente.

Dibujar.

Con ayuda de los comandos scale (escalar), rotate (rotar) y translate (trasladar) colocamos cada uno de los elementos previamente importados en el lugar deseado.

Para que las transformaciones espaciales realizadas no afecten al sistema de coordenadas original, se coloca el código entre los comando pushMatrix() y popMatrix().

Para mostrar la información recibida desde Arduino necesitamos que ciertas partes del programa varíen en consecuencia:

- **Batería.** Se emplea un rectángulo de ancho variable. La anchura dependerá del porcentaje de batería restante que se nos indica por el puerto serie.
- **Testigos.** Se dibuja un circulo semitransparente encima de la imagen del testigo activado.
- **Velocímetro.** Para simular la aguja del velocímetro se dibuja una linea con un extremo en el origen del velocímetro, y el otro girando en torno a dicho punto según la velocidad obtenida.

Lo anterior se puede observar en el código y las pruebas que se realizan en los apartados siguientes.

6.2.4 Establecer comunicación con Arduino.

CÓDIGO ARDUINO.

Se ha aislado del código principal del control de velocidad por simplicidad, en caso de querer añadir el tablero al vehículo basta con añadir este código al general.

```
1
2  //_____________________________________________________________________________________
3  //                              DEFINICIONES PREVIAS
4  //_____________________________________________________________________________________
5
6  // Para el envio del byte de inicio.
7  int   byteInicio  = 0                              ;
8
9  // Para almacenar la velocidad, la bateria restante y el estado de los sensores.
10 float velocidad         = 0                        ;
11 byte  *p_velocidad      = (byte*)  &velocidad      ;
12 int   nivelBateria      = 0                        ;
13 int   sensor1   = 0, sensor2    = 0, sensor3    = 0 ;
14 int   sensor4   = 1, sensor5    = 1, sensor6    = 1 ;
15
16
17
18 //_____________________________________________________________________________________
19 //                                FUNCIÓN INICIAL
20 //_____________________________________________________________________________________
21
22 void setup() {
23
24     // Abrir puerto serie con la configuración deseada.
25     Serial.begin(9600,SERIAL_8N1) ;
26
27     // Dar un valor al byte de inicio.
28     byteInicio  = 200   ;
29
30 }
31
32
33 //_____________________________________________________________________________________
34 //                                     BUCLE
35 //_____________________________________________________________________________________
36
37 void loop(){
38
39     // Valores de muestra (en el programa original se obtienen del coche).
40     velocidad       = 50                           ;
41     nivelBateria    = 100                          ;
42     sensor1   = 1, sensor2    = 1, sensor3    = 1  ;
43     sensor4   = 1, sensor5    = 1, sensor6    = 1  ;
44
45
46     // Enviar byte de comienzo seguido de la velocidad y el estado de los sensores.
47     Serial.write(byteInicio)                                                    ;
48     Serial.write( p_velocidad , 4 )                                             ;
49     Serial.write(nivelBateria)                                                  ;
50     Serial.write(sensor1); Serial.write(sensor2); Serial.write(sensor3)    ;
```

```
    Serial.write(sensor4); Serial.write(sensor5); Serial.write(sensor6)    ;

    // Simular tiempo entre iteraciones del bucle del programa principal.
    delay(100)     ;

}
```

CÓDIGO PROCESSING.

Este código es el encargado de dar vida al tablero de abordo.
Incluimos los modelos e imagenes deseados que posteriormente colocaremos, leemos los datos que le llegan desde Arduino por el puerto serie y, en consecuencia, modificamos la escena de una u otra forma.

```

//_____________________________________________________________________________
//                              DEFINICIONES PREVIAS
//_____________________________________________________________________________

// Libreria y elemento tipo serie.
import  processing.serial.*                     ;
Serial  puertoArduino                           ;

// Variables para comunicación con Arduino.
int     byteInicio      =  0                    ;
int     byteInformacion =  0                    ;

// Variables para almacenar información de la comunicación serie.
float   velocidad     =  0.0                    ;
int     nivelBateria  =  0                      ;
int     sensor1  = 0, sensor2  = 0, sensor3  = 0  ;
int     sensor4  = 0, sensor5  = 0, sensor6  = 0  ;
int[]   array = new int[11];

// Crear elementos para guardar los archivos .obj.
PShape  panel                                   ;
PShape  velocimetro                             ;
PShape  vehiculo                                ;

// Crear elementos para guardar imagenes.
PImage  bateria                                 ;
PImage  testigos                                ;

// Para girar coche 3D.
float   giro    =  0.0                          ;

// Para posicionar aguja del velocímetro.
float anguloAguja     =  0.0                    ;

// Para el ancho del rectángulo que representa la carga de la bateria.
float anchoBateria    =  0.0                            ;

```

```
40 //_____________________________________________________________________________________
41 //                                FUNCIÓN INICIAL
42 //_____________________________________________________________________________________
43
44 public void setup() {
45
46     // Tamaño de la pantalla e imagenes por segundo.
47     size(1280,720,P3D);
48     frameRate(60);
49
50     // Selección de puerto y parámetros de la comunicación.
51     String portName = Serial.list()[0];
52     puertoArduino = new Serial(this, portName, 9600);
53     puertoArduino.buffer(12);
54
55     // Cargar .obj e imagenes.
56     panel = loadShape("panel.obj");
57     velocimetro = loadShape("velocimetro.obj");
58     vehiculo = loadShape("vehiculo3D.obj");
59     bateria=loadImage("bateria.png");
60     testigos=loadImage("testigos.png");
61
62 }
63
64
65 //_____________________________________________________________________________________
66 //                                    BUCLE
67 //_____________________________________________________________________________________
68
69 public void draw() {
70
71     // Color de fondo, luz de fondo y posición inicial.
72     background(100);
73     ambientLight(128, 128, 128);
74     directionalLight(128, 128, 128, 0, 0, 1);
75     lightFalloff(1, 0, 0);
76     lightSpecular(0, 0, 0);
77     translate(width/2,height/2,0);
78
79 //_____________________________________________________//
80 //          ESCENA AJENA A INFORMACIÓN DEL PUERTO SERIE       //
81 //_____________________________________________________//
82
83     // PANEL.
84     pushMatrix();
85       scale(width/8,-width/8,width/8);
86       shape(panel);
87     popMatrix();
88
89     // VELOCIMETRO.
90     pushMatrix();
91       translate(0,0,10);
92       scale(width/25,-width/25,width/25);
93       shape(velocimetro);
94     popMatrix();
95
96     // VEHICULO.
97     pushMatrix();
98       translate(-width/4.5,width/50,width/10.66);
99       scale(width/50,-width/50,width/50);
100      rotateY(giro);
```

```
101       shape(vehiculo);
102     popMatrix();
103     giro-=0.01;
104 
105     // BATERIA y TESTIGOS.
106     pushMatrix();
107       translate(0,0,width/42.6);
108       scale(1,-1,1);
109       fill(255);
110       image(bateria,-width/9,width/7,width/4.26,width/25.6);
111       image(testigos,width/6,-width/12,width/5.57,width/8.53);
112     popMatrix();
113 
114 //_____________________________________________________________//
115 //   ACTUALIZAR ESCENA CON LA INFORMACIÓN DEL PUERTO SERIE     //
116 //_____________________________________________________________//
117 
118     // AGUJA DEL VELOCÍMETRO EN SU POSICIÓN CORRESPONDIENTE.
119     // Formar float con los 4 primeros bytes del array teniendo en cuenta que la información ha
          llegado en littleEndian.
120     velocidad=Float.intBitsToFloat(int(array[3] << 24) | int(array[2] << 16) | int(array[1] << 8)
          | int(array[0]));
121     // Para actualizar el dato anterior la velocidad tiene que estar entre 0 y 50 km/h.
122     if (velocidad>=0 && velocidad<=50){
123         // Mostrar en la consola.
124         println("Velocidad:␣␣" + "\t\t" + velocidad + "␣km/h");
125         // Dibujar aguja.
126         pushMatrix();
127         stroke(255);
128         strokeWeight(3);
129         // Mapear la velocidad al rango de posiciones de la aguja del velocimetro.
130         anguloAguja=map(velocidad,0,50,PI/5.2,PI*1.8);
131         rotate(anguloAguja);
132         line(0, 0, width/4.267, 0, width/21.33, width/4.267);
133         popMatrix();
134     }
135 
136 
137     // NIVEL DE CARGA DE LA BATERÍA, VERTICALES SOBRE LA MISMA.
138     // Obtener dato del array y actualizar valor anterior si esta comprendido entre 0 y 100%.
139     nivelBateria=array[4];
140     if (nivelBateria>=0 && nivelBateria<=100){
141         // Mostrar en la consola.
142         println("Nivel␣de␣bateria:␣" + "\t" + nivelBateria + "␣%");
143         // Dibujar.
144         pushMatrix();
145         noStroke();
146         fill(0,255,0);
147         translate(0,0,width/25.6);
148         // Mapear la carga de la bateria al rango de posiciones del rectángulo que la representa.
149         anchoBateria=map(nivelBateria,0,width/12.8,0,width/5.12);
150         rect(-width/10.24,-width/5.95,anchoBateria,width/42.67);
151         popMatrix();
152     }
153 
154     // CIRCULO SEMITRANSPARENTE SOBRE TESTIGOS ACTIVO.
155     // Obtener datos del array
156     sensor1=array[5]; sensor2=array[6]; sensor3=array[7]; sensor4=array[8]; sensor5=array[9];
          sensor6=array[10];
157     // Mostrar en la consola.
158     println("Sensores␣␣:" + "\t\t" + sensor1 + "\t" + sensor2 + "\t" + sensor3 + "\t" +
```

```
159     sensor4 + "\t" + sensor5 + "\t" + sensor6 + "\n");
160     pushMatrix();
161     noStroke();
162     translate(0,0,width/25.6);
163     // Si el sensor está activo dibujar circulo semitransparente encima de su correspondiente
            icono.
164     if (sensor1==1){
165         fill(255,0,0,200);
166         ellipse(width/5.35,-width/180,width/20,width/20);
167     }
168     if (sensor2==1){
169         fill(255,0,0,200);
170         ellipse(width/4.04,-width/180,width/20,width/20);
171     }
172     if (sensor3==1){
173         fill(255,0,0,200);
174         ellipse(width/3.25,-width/180,width/20,width/20);
175     }
176     if (sensor4==1){
177         fill(255,0,0,200);
178         ellipse(width/5.35,width/19,width/20,width/20);
179     }
180     if (sensor5==1){
181         fill(255,0,0,200);
182         ellipse(width/4.04,width/19,width/20,width/20);
183     }
184     if (sensor6==1){
185         fill(255,0,0,200);
186         ellipse(width/3.25,width/19,width/20,width/20);
187     }
188     popMatrix();
189
190 }
191
192
193   //_____________________________________________________________________________________
194   //                                LECTURA DEL PUERTO SERIE
195   //_____________________________________________________________________________________
196
197 // Función que se activa cada 12 bytes en el buffer (especificado en el setup con el comando
        puertoArduino.buffer).
198 void serialEvent(Serial puertoArduino) {
199
200     // Leemos el primer byte y comprobamos que se corresponde con el byte de inicio enviado por
            arduino.
201     byteInicio=puertoArduino.read();
202     if (byteInicio==200){
203         // En caso afirmativo introducimos los 11 bytes siguientes en un array.
204         for (int i=0; i<11; i=i+1){
205             byteInformacion=puertoArduino.read();
206             array[i]=byteInformacion;
207         }
208     }
209     else{
210         // Si el primer byte no se corresponde con el inicio de comunicación se muestra un error
                en la consola.
211         println("Error: byte de inicio incorrecto.");
212     }
213
214 }
```

6.2.5 Probando el código.

Se va a comprobar el correcto funcionamiento del código mostrando el tablero de abordo y la consola de processing ante supuestos valores de las entradas.
Para tal fin modificamos las variables correspondiente en el programa de Arduino y ejecutamos ambos códigos.

PRUEBA 1.

```
// Valores de muestra (en el programa original se obtienen del coche).
velocidad       = 0                             ;
nivelBateria    = 0                             ;
sensor1   = 0, sensor2    = 0, sensor3    = 0   ;
sensor4   = 0, sensor5    = 0, sensor6    = 0   ;
```

Figura 6.10: Sensores al mínimo (Arduino)

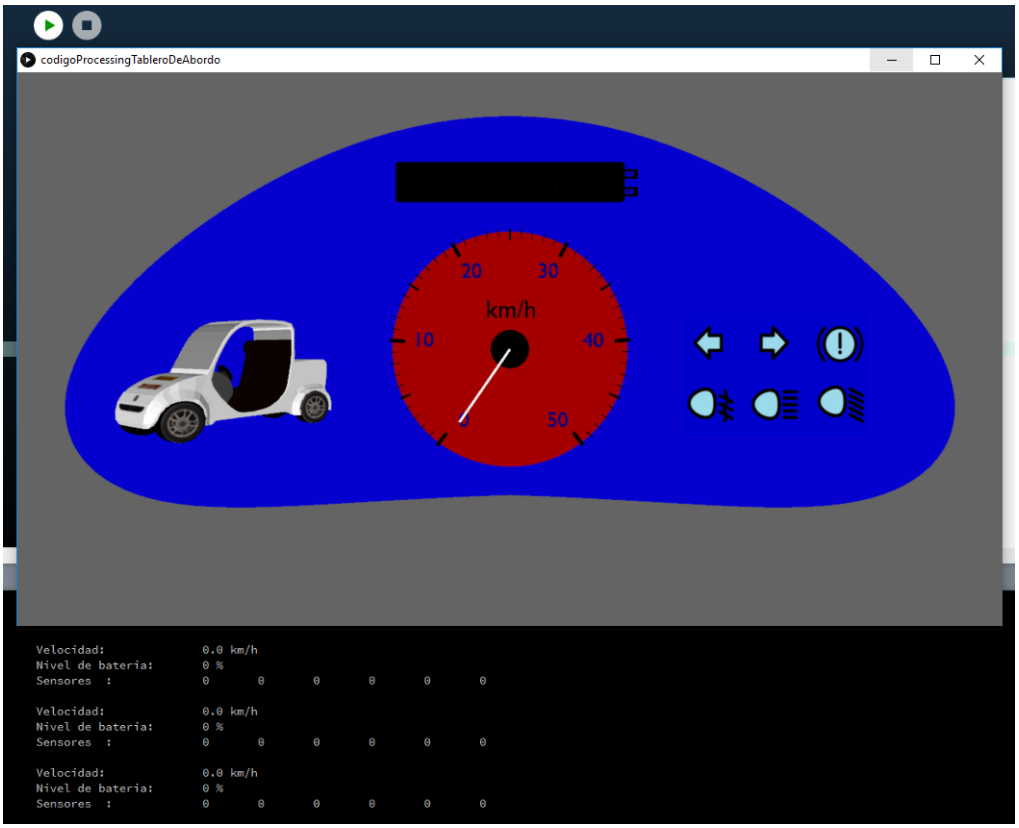

Figura 6.11: Sensores al mínimo (Processing)

Con todos los sensores a cero el resultado gráfico es el esperado.
La consola de procesing nos muestra que la comunicación está funcionando correctamente.

PRUEBA 2.

```
// Valores de muestra (en el programa original se obtienen del coche).
velocidad       = 50                            ;
nivelBateria    = 100                           ;
sensor1   = 1, sensor2    = 1, sensor3    = 1   ;
sensor4   = 1, sensor5    = 1, sensor6    = 1   ;
```

Figura 6.12: Sensores al máximo (Arduino)

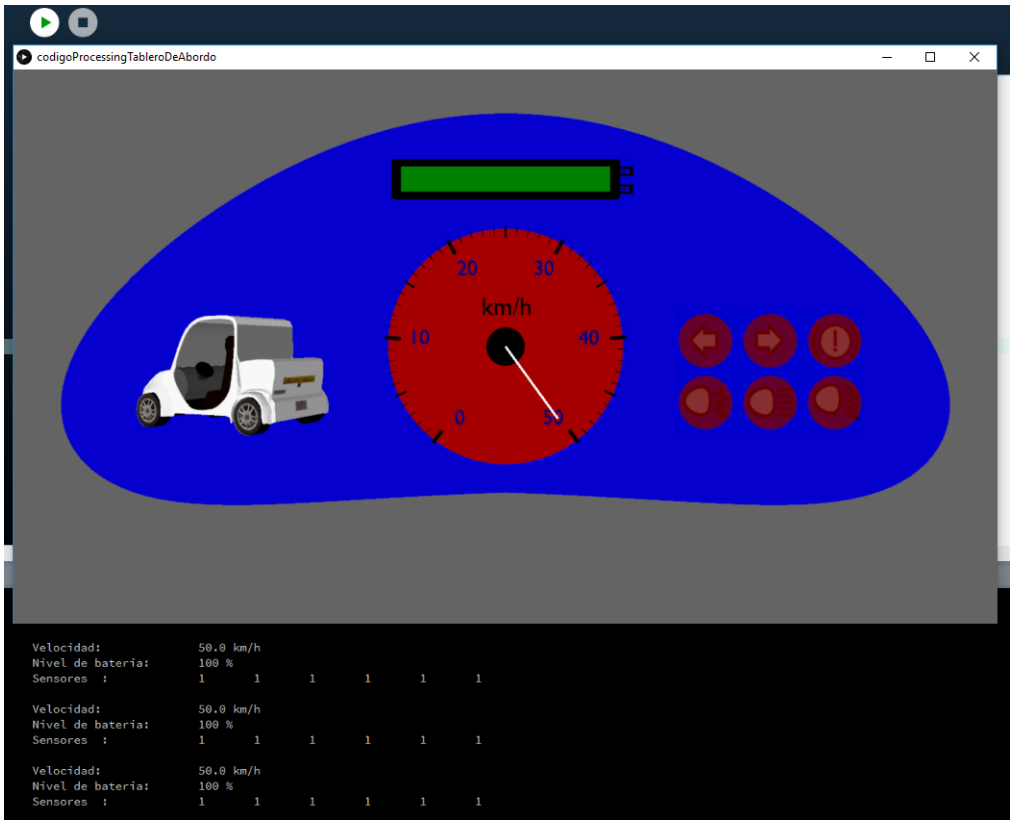

Figura 6.13: Sensores al máximo (Processing)

Con todos los sensores a valor máximo el resultado gráfico es el esperado.
La consola de procesing nos muestra que la comunicación está funcionando correctamente.

PRUEBA 3.

```
// Valores de muestra (en el programa original se obtienen del coche).
velocidad       = 35.56                         ;
nivelBateria    = 42                            ;
sensor1   = 0, sensor2    = 1, sensor3    = 0   ;
sensor4   = 1, sensor5    = 0, sensor6    = 1   ;
```

Figura 6.14: Sensores con valor aleatorio (Arduino)

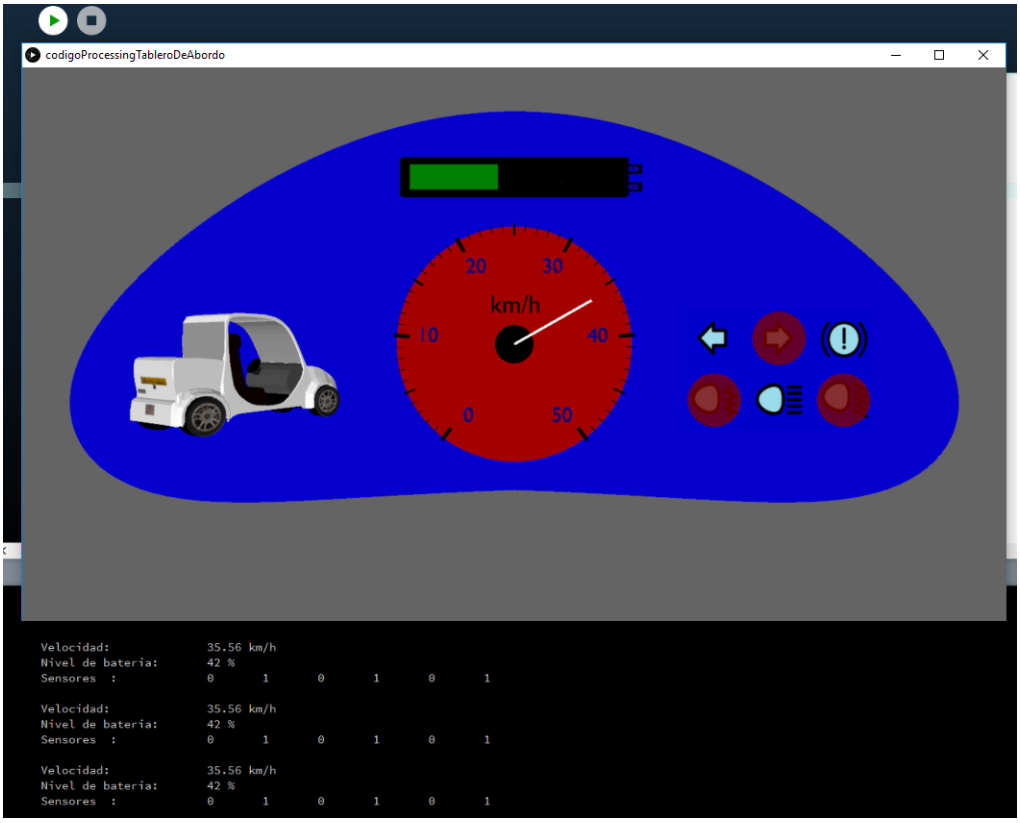

Figura 6.15: Sensores con valor aleatorio (Processing)

Con valores de los sensores aleatorios el resultado gráfico es el esperado.
La consola de procesing nos muestra que la comunicación está funcionando correctamente.

6.2.6 Mejoras del panel.

Por problemas de tiempo no se ha llegado a usar el tablero en el vehículo, sin embargo nos deja el camino abierto de cara a una posible mejora del mismo.

Si quisiesemos enviar información real (exceptuando la velocidad que ya la tenemos), sería necesario añadir los respectivos sensores en el vehículo y mandar dichas señales a Arduino:

- Para las luces podríamos sacar algunos cables en paralelo de la propia instalación y, previo acondicionamiento de las señales, conectarlas a las entradas digitales del Arduino.
- Para el nivel del líquido de frenos se podría añadir un sensor de nivel en el tanque, con la salida conectada a una de las entradas de Arduino.
- Para el nivel de carga de la batería podríamos controlar la tensión entre bornes.

Los sensores elegidos para el tablero son una simple muestra, siguiendo los pasos mostrados en este bloque se podrá modificar el tablero a gusto del usuario.

ANEXO I:
CÁLCULO DE FILTROS.

REPASO FILTROS

Los filtros electrónicos nos permiten eliminar las frecuencias no deseadas presentes en una determinada señal. También nos permiten modificar la ganancia y la fase de las distintas frecuencias puestas en juego.

Según su función los filtros pueden ser:

Paso bajo.

Permiten el paso de todas las frecuencias por debajo de la frecuencia de corte.

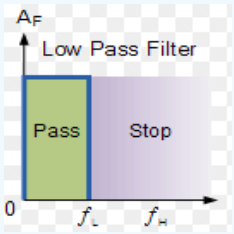

Figura 6.16: Filtro paso bajo

Paso alto.

Permiten el paso de todas las frecuencias por encima de la frecuencia de corte.

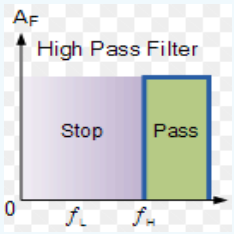

Figura 6.17: Filtro paso alto

Pasa banda.

Permiten el paso de todas las frecuencias comprendidas entre la frecuencia de corte superior e inferior.

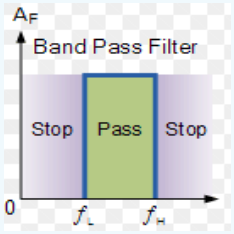

Figura 6.18: Filtro pasa banda

Rechaza banda.

Permiten el paso de todas las frecuencias por debajo de la frecuencia de corte inferior y por encima de la frecuencia de corte superior.

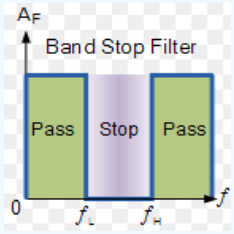

Figura 6.19: Filtro rechaza banda

Pasa todo.

Permiten el paso de todas las frecuencias por igual. Se usan cuando se requiere una modificación en la respuesta en fase.

Idealmente, como hemos visto en las ilustraciones precedentes, los filtros tienen una ganancia constante para todas las frecuencias a las que permiten el paso y nula para el resto, además, el cambio al atravesar la frecuencia de corte es inmediato.

Lo anterior no es cierto en el mundo real, según el filtro usado la ganancia puede variar bastante y el atravesar la frecuencia de corte no supone un cambio repentino en la misma, sino que se produce de manera paulatina. La frecuencia de corte pasa a considerarse como la frecuencia a la cual la señal sufre una caida de 3dB.

En la siguiente imagen se pueden observar los principales parámetros a tener en cuenta a la hora de diseñar un filtro real tomando un filtro paso bajo como ejemplo.

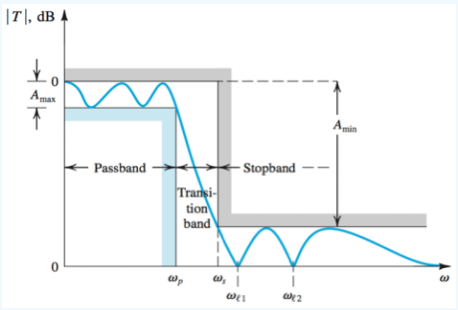

Figura 6.20: Parámetros filtro paso bajo

Cuanto se aproxime la respuesta ideal a la real al atravesar la frecuencia de corte depende en gran medida del orden del filtro.

En la siguiente imagen se muestra la respuesta en frecuencia de un filtro Butterworth (una de las configuraciones más empleadas que proporciona una ganancia en la zona de paso bastante plana) variando el orden del mismo. Al aumentar el orden la respuesta se vuelve más ideal pero el número de

componentes, precio y complejidad del circuito aumenta.

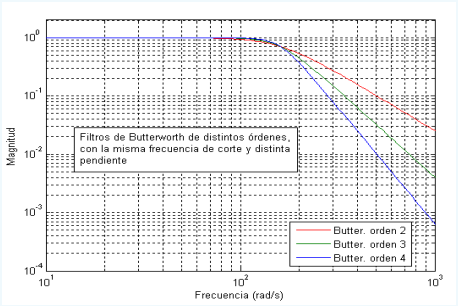

Figura 6.21: Filtro Butterworth con distintos ordenes

Según la forma de implementar el filtrado distinguimos filtros analógicos y filtros digitales.

Un **filtro digital** puede llegar a ser más preciso pero será necesario un muestreo previo de la señal y devolver la misma al mundo analógico tras su filtrado. Al trabajar con filtros digitales es importante atender al teorema de muestreo de Nyquist-Shannon, que nos permite reconstruir la señal original sin pérdida de información (la frecuencia de muestreo debe ser mayor al doble de la frecuencia más alta presente en la señal a muestrear). Junto con los filtros digitales se usa un filtro antialiasing (filtro analógico paso bajo encargado de eliminar frecuencias superiores a la frecuencia de Nyquist, es decir, la mitad de la frecuencia de muestreo, y evitar por tanto introducir información errónea en el muestreo que podría solaparse con la señal original).

Los **filtros analógicos** pueden ser pasivos (construidos unicamente con elementos pasivos, es decir, aquellos que disipan o almacenan energía: resistencias, bobinas y condensadores) o activos (en su construcción aparece algún elemento activo, en concreto elementos que introducen una ganancia: operacionales, transistores....).

FILTRADO DE LA SALIDA PWM DE ARDUINO.

Necesitamos una frecuencia de corte del filtro inferior a la frecuencia de la señal PWM de Arduino.
Las frecuencias por defecto de las salidas PWM de Arduino son:

- **Pines 3, 9, 10, 11** : 490 Hz.

- **Pines 5, 6** : 980 Hz.

Se está haciendo uso de la salida 9 por tanto la frecuencia es de 490 Hz.
Los valores de la resistencia y del condensador vendrán dados por la siguiente ecuación,

$$f_c = \frac{1}{2\pi RC}$$

Se supone un filtro RC con operacional para evitar caídas de tensión al conectar la controladora.
Se busca un frecuencia de corte lo suficientemente alejada de la frecuencia de Arduino, se toma un valor de 15 Hz.
Para esta frecuencia y partiendo de un condensador electrolítico de 1 µF obtenemos una resistencia de 10 kΩ.

En la figura 6.22 se muestra el montaje del filtro.

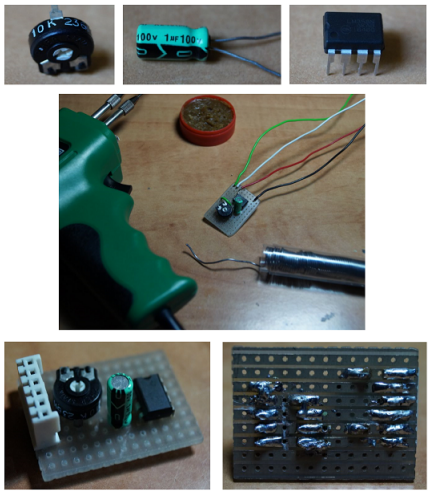

Figura 6.22: Implementación filtro activo

Pese a priori haber considerado el rizado lo suficientemente pequeño, una vez instalado el filtro se quiso mejorar la precisión disminuyendo dicho rizado.
Existían por tanto dos opciones:

1) Cambiar el filtro por otro con menor frecuencia de corte.

2) Aumentar la frecuencia de la señal PWM de Arduino.

Se tomó la segunda opción.

Las frecuencias vistas anteriormente (490 Hz y 980 Hz) no son las frecuencias base de los timer de Arduino, llevan un prescaler asociado.
Del datasheet del microcontrolador podemos obtener los siguientes datos:

- **Timer 0.** Asociado a los pines 5 y 6 con una frecuencia base de 62500 Hz con prescalers disponibles de 1, 8, 64, 256 y 1024.

- **Timer 1.** Asociado a los pines 9 y 10 con una frecuencia base de 31250 Hz con prescalers disponibles de 1, 8, 64, 256 y 1024.

- Timer 2. Asociado a los pines 3 y 11 con una frecuencia base de 31250 Hz con prescalers disponibles de 1, 8, 32, 64, 128, 256 y 1024.

Si modificamos la frecuencia del Timer0 y/o del Timer2 podemos interferir en el correcto funcionamiento de las funciones millis() y micros(), por lo tanto estos timer no podemos modificarlos (haremos uso de esas funciones).

Sin embargo, el Timer1 influiye en el correcto funcionamiento de alguna que otra librería (por ejemplo la librería servo.h), como en este programa no se usa ninguna de ellas podemos modificar la frecuencia de este timer sin problema.
El pin asignado previamente a la salida del acelerador ha sido el 9, que está dentro de los pines asignados al timer1 por lo que no necesitamos cambiar de pin.

Para cambiar la frecuencia de un timer tenemos que acceder a los registros del ATmega328p, en este caso como queremos preescaler de 1 en el timer1 basta con introducir el siguiente comando en el setup:

$$TCCR1B = 0b00000001;$$

FILTRADO SEÑAL SENSOR HALL.

Se han tomado muestras de la señal de pulsos al excitar la controladora con tensiones de 1.6V, 2.5V, 4.3V y 5V, y sus respectivas salidas se muestran en la figura 6.23 de izquierda a derecha y de arriba hacia abajo.

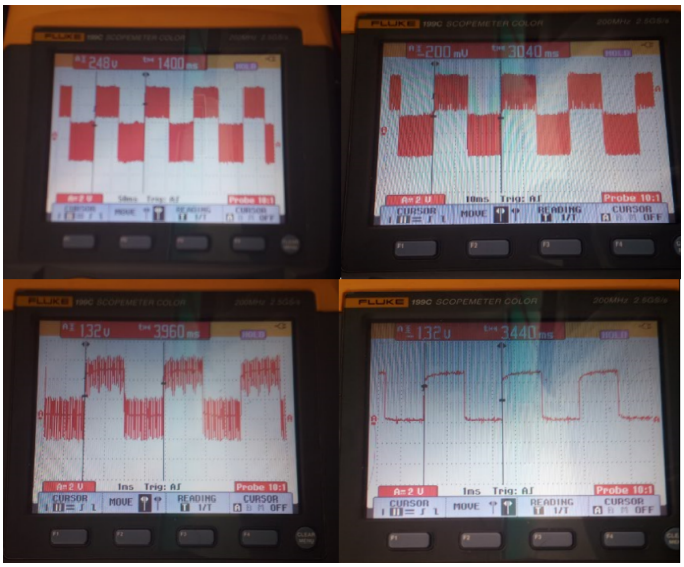

Figura 6.23: Salida sin filtrar

Necesitamos un filtro que permita a Arduino identificar los escalones correctamente a bajas velocidades.
Suponiendo una alimentación de 5V, una tensión por encima 3V será considerada por Arduino como nivel alto y una por debajo de 1.5V será considerada como nivel bajo (figura 6.24). Debemos eliminar cualquier señal de ruido que se introduzca en dicho intervalo, ya que en él los valores que toma la salida son aleatorios, pudiendo provocar falsas activaciones.

28.2 DC Characteristics

T_A = –40°C to +125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)

Parameter	Condition	Symbol	Min.	Typ.	Max.	Unit
Input low voltage, except XTAL1 and $\overline{\text{RESET}}$ pin	V_{CC} = 2.7V to 5.5V	V_{IL}	–0.5		$0.3V_{CC}$ (1)	V
Input high voltage, except XTAL1 and $\overline{\text{RESET}}$ pins	V_{CC} = 2.7V to 5.5V	V_{IH}	$0.6V_{CC}$ (2)		V_{CC} + 0.5	V
Input low voltage, XTAL1 pin	V_{CC} = 2.7V to 5.5V	V_{IL1}	–0.5		$0.1V_{CC}$ (1)	V
Input high voltage, XTAL1 pin	V_{CC} = 2.7V to 5.5V	V_{IH1}	$0.7V_{CC}$ (2)		V_{CC} + 0.5	V

Notes: 1. "Max" means the highest value where the pin is guaranteed to be read as low
2. "Min" means the lowest value where the pin is guaranteed to be read as high

Figura 6.24: Tensiones entradas digitales ATmega328p

Un filtro paso bajo pasivo de primer orden será suficiente para este caso, la señal de pulsos de mayor frecuencia que se debe preservar es de unos 300 Hz, debemos asegurarnos de que la frecuencia de corte del filtro empleado se encuentre por encima de dicho valor.

$$f_c = \frac{1}{2\pi RC}$$

Suponemos una frecuencia de corte varias décadas por encima de los 300 Hz a mantener, se ha seleccionado una frecuencia de corte de 50 kHz.
Partiendo de esta frecuencia y un condensador cerámico de 1nF (102) obtenemos un valor para la resistencia de 3kΩ (valor comercial por si se desea sustituir por una resistencia fija). .

Simulación del filtro mediante Matlab (figura 6.25).

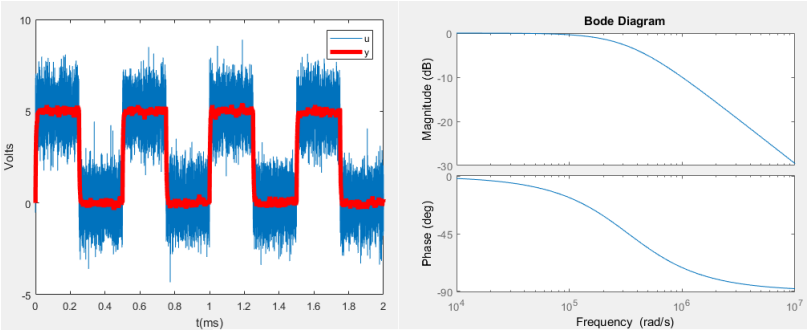

Figura 6.25: Filtro matlab

Se monta el circuito con los valores de resistencia y condensador previamente calculados (figura 6.26).

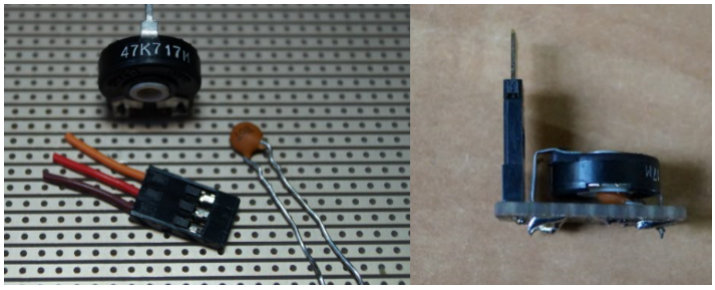

Figura 6.26: Filtro

Probamos el circuito en el vehículo, se muestra a continuación la salida ante el ciclo de trabajo más propenso al ruido, el 50% (el rizado es el más amplio).

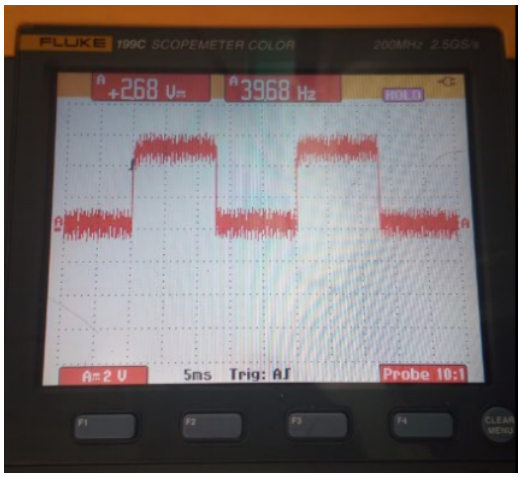

Figura 6.27: Señal filtrada

Como se ha visto previamente cualquier tensión por encima de 3V será tomada como alta (en el ejemplo todos los valores altos están por encima de 4V).
Las tensiones por debajo de 1.5V serán tomadas como valor bajo, en este caso estamos por debajo pero no por mucho, no obstante, el valor proporcionado en el datasheet es en el peor de los casos, el rango real suele ser algo mayor.

Tras realizar la misma lectura para las distintas tensiones aplicadas se comprueba que todas caen dentro del margen de error permitido.

ANEXO II: COMUNICACIÓN ARDUINO-MATLAB.

Como se ha comentado en la introducción, a lo largo de este trabajo se ha supuesto que MATLAB es el software presente en el sistema de control inteligente con el que se debe de comunicar nuestro control de velocidad.

En este anexo se muestra un ejemplo para establecer una comunicación tanto serie como ethernet entre dichos programas.

Notar el uso de punteros con objeto de conseguir una comunicación más eficiente.

PRUEBA COMUNICACIÓN SERIE.

Para comprobar el correcto funcionamiento de la comunicación serie se procede a la realización de un programa sencillo.

Se generará en matlab una señal senoidal que será leida por Arduino, donde, tras invertir y doblar su valor orginal, es devuelta a Matlab.
Para comprobar si la comunicación es correcta se dibuja en Matlab la señal original y la modificada.

Se ha escogido una comunicación 8N1, es decir, 8 bits de datos, sin bit de paridad con 1 bit de inicio y otro de parada. Esta comunicación presenta la ventaja de ser rápida y sencilla, como contrapartida, al no tener bit de paridad, no permite detectar errores de trasmisión dentro del propio programa.

Por otro lado la tasa de baudios (señales por segundo) será de 9600 bps (valor por defecto en muchos programas debido a su versatilidad).

$$9600\ bits\ por\ segundo\ = 960\ bits\ cada\ 0.1\ segundos$$

$$960\ bits\ cada\ 0.1\ segundos/10\ bits\ por\ byte\ de\ información = 96\ bytes\ de\ información$$

Nosotros solo tenemos que enviar 3 bytes cada 0.1 segundos por lo tanto este valor es más que suficiente.

Código MATLAB.

1. Se define un objeto serie donde se especifica el tipo de comunicación deseada.
2. Se crea una función senoidal de amplitud unitaria con frecuencia de 0.1Hz. Se muestrea cada 0.1 segundos almacenando los valores en la variable "y", y se inicializa otra variable del mismo tamaño para almacenar los datos de la senoide modificada.
3. Se crea una gráfica con estas dos funciones que se irá actualizando con cada muestreo del bucle.
4. Dentro del bucle limpiamos el buffer de entrada en la primera iteración (para sincronizar), leemos la señal original, enviamos la señal modificada, modificamos el byte recibido y actualizamos el gráfico.
5. Es importante eliminar el objeto serie para evitar conflictos si se desea realizar otra comunicación.

```
%------------------------------------------------------------------%
%             PRUEBA COMUNICACION SERIE                            %
%------------------------------------------------------------------%
% DESCRIPCION:                                                     %
% El programa comienza borrando cualquier paquete                  %
% disponible por motivos de sincronizacion,                        %
% a continuacion lee los bytes recibidos, almacena                 %
% el valor en una variable y envia un paquete con la               %
% informacion contenida en otra.                                   %
% Muestra en una grafica la informacion que se envia               %
% y la que se recibe.                                              %
%------------------------------------------------------------------%
%                                    Paulino San Miguel Herrera    %
```

```
%-------------------------------------------------------------------%

% Limpiar variables y ventana de comandos.
close all; clc

% Definir puerto serie y comunicacion.
Usb=serial('COM5');
set(Usb,'BaudRate',9600,'Parity','none','DataBits',8,'StopBits',1);
fopen(Usb);

% Inicializar senial senoidal original y modificada.
ts     =  0.1                ;
tFin   =  20                 ;
t      =  (0:ts:tFin)'       ;
f      =  0.1                ;
amp    =  1                  ;
y      =  amp*sin(2*pi*f*t)  ;
y2     =  zeros(size(t))     ;

% Grafica con ambas seniales.
h      =  plot(t ,[y, y2], '*')  ;
grid on                          ;
xlim([0 tFin])                   ;
ylim([-3 3])                     ;
drawnow                          ;

% Bucle para la comunicacion.
N = length(t); n = 0;

while n < N
    n = n+1;

    % Limpiar buffer de entrada en la primera iteracion.
    if n==1
       flushinput(Usb);
    end

     % Esperar disponibilidad de datos.
    while (Usb.BytesAvailable<4)
    end

    % Leer datos.
    y2(n) = fread(Usb,1,'single');

    % Enviar datos.
    fwrite(Usb,y(n),'single');

    % Actualizar senoide modificada.
    set(h(2),'ydata',y2);
```

```
63      drawnow;
64
65  end
66
67  % Cerrar puerto serie.
68  fclose(Usb); delete(Usb);
69  delete(instrfindall)
```

Código ARDUINO.

1. Se comienza definiendo las variables que aparecerán a lo largo del programa, cabe destacar el uso de punteros (variables que apuntan hacia el lugar de memoria ocupado por otras) cuando aparecen variables de tipo float en la comunicación.
   La opción más común al trabajar con flotantes en arduino es enviarlos en formato de cadena de texto, no obstante usando punteros el código será más eficiente ya que solo tenemos que enviar los 4 bytes que forman el float.

2. En el setup iniciamos la comunicación serie con la misma configuración usada en el otro extremo.

3. Ejecutamos el bucle cada 0.1 segundos (para simular el tiempo de ejecución del programa principal), enviamos la información por el puerto serie y, cuando se disponga de un byte para leer, lo recogemos y modificamos.

```
1   //--------------------------------------------------------------------//
2   //                      PRUEBA COMUNICACIÓN SERIE                     //
3   //--------------------------------------------------------------------//
4   // DESCRIPCIÓN:                                                       //
5   // Código para establecer una comunicación SERIE.                     //
6   // Enviamos de manera continua la información de una variable.        //
7   // Cuando detectamos información procedente del otro extremo la leemos,  //
8   // modificamos, y almacenamos en la variable que enviamos.            //
9   // Es necesario que ambos ambos extremos de la comunicación estén     //
10  // configurados igual para que puedan entenderse.                     //
11  //--------------------------------------------------------------------//
12  //                                          Paulino San Miguel Herrera //
13  //--------------------------------------------------------------------//
14
15
16  //----------------------//
17  // DEFINICIONES PREVIAS //
18  //----------------------//
19
20  // Variables temporales.
```

```
21 unsigned long Ts      = 100000  ;
22 unsigned long t       = 0       ;
23
24 // Variable para la senoide y puntero a su primer byte.
25 float sen   = 0.0 ;
26 byte  *p_sen = (byte*)  &sen ;
27
28
29 //----------------------//
30 //   FUNCIÓN INICIAL    //
31 //----------------------//
32
33 void setup( ) {
34
35     // Abrir puerto serie con la configuración especificada.
36     Serial.begin(9600, SERIAL_8N1);
37
38 }
39
40
41 //----------------------//
42 //        BUCLE         //
43 //----------------------//
44
45 void loop() {
46
47     // El código se ejecuta cada décima de segundo.
48     if (micros()-t >= Ts){
49         t = micros();
50
51         // Enviar senoide modificada.
52         Serial.write( p_sen , 4 ) ;
53
54         // Leer valor de la senoide.
55         Serial.readBytes(  p_sen , 4 ) ;
56         sen=-sen*2;
57     }
58
59 }
```

Resultados.

El programa lleva un retraso de una muestra, esto se debe a que la sinusoide se ha generado en Matlab y, sin embargo, se ha programado (por simplicidad), de manera tal que es Arduino quien comienza la comunicación.

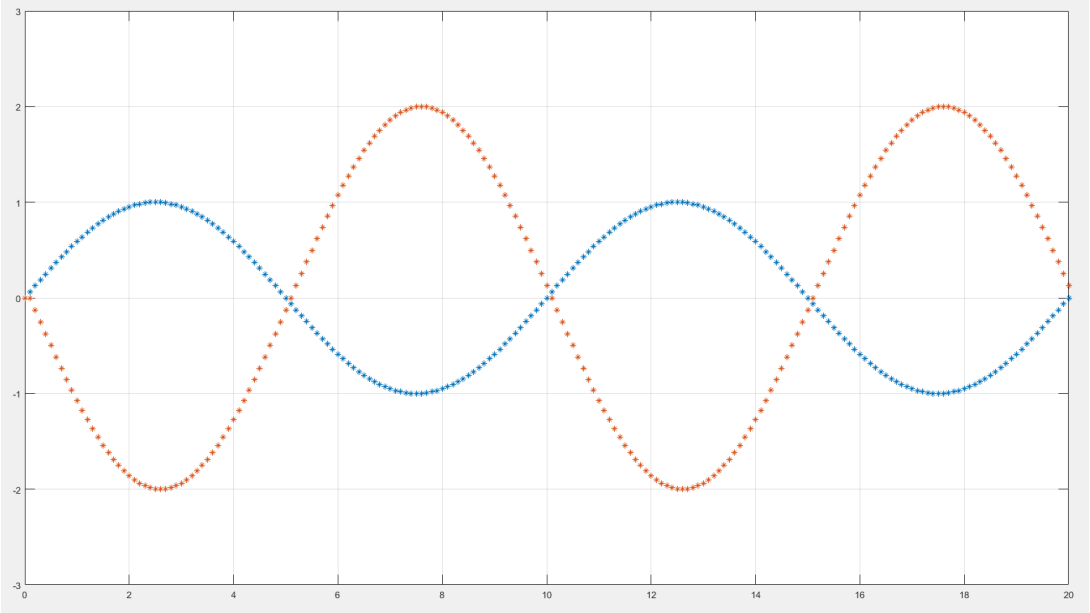

Figura 6.28: Prueba comunicación serie

PRUEBA COMUNICACIÓN ETHERNET.

Código MATLAB.

El programa es muy parecido al que se usa en la comunicación serie, varía el inicio del mismo donde en lugar de crear un objeto serie creamos un objeto udp.

Se ha creado una red de área local del tipo 10.x.x.x por ser una de las reservadas para conexiónes privada, no osbtante, como en principio no nos conectaremos con internet, podríamos haber escogido cualquiera.
Necesitamos indicar tanto las ip como los puertos de los extremos de la comunicación.

Otra cosa a tener en cuenta es la codificación de los bytes en los paquetes que se envían y reciben, en el caso especial de estos programas dicha codificación es distinta.

Por un lado Matlab hace uso de ”big endian” (el byte más significativo primero) y, por el otro, Arduino usa ”little endian” (el bit menos significativo primero).
La solución pasa por hacer coincidir la codificación de ambos, en este caso se ha modificado el objeto creado en Matlab para que trabaje con little endian (instrucción ”ByteOrder”).

```
%-------------------------------------------------------------------%
%                      PRUEBA COMUNICACION UDP                      %
%-------------------------------------------------------------------%
% DESCRIPCION:                                                      %
% El programa comienza borrando cualquier paquete                   %
% disponible por motivos de sincronizacion,                         %
% a continuacion lee el siguiente paquete recibido,                 %
% almacena el valor en una variable y envia un paquete              %
% con la informacion contenida en otra.                             %
% Muestra en una grafica la informacion que se envia                %
% y la que se recibe.                                               %
%-------------------------------------------------------------------%
%                                      Paulino San Miguel Herrera   %
%-------------------------------------------------------------------%

% Limpiar variables y ventana de comandos.
close all; clc

% Crear objeto para realizar comunicacion por UDP.
Udp = udp('10.0.0.2',8080,'localPort',8081,'LocalHost','10.0.0.1');

% Por defecto Matlab usa bigEndian, cambiamos a littleEndian
% de forma que Arduino y Matlab trabajen igual.
Udp.ByteOrder='littleEndian';

% Abrir puertos.
fopen(Udp);

% Inicializar senial senoidal original y modificada.
ts    =  0.1                ;
tFin  =  20                 ;
t     =  (0:ts:tFin)'       ;
f     =  0.1                ;
amp   =  1                  ;
y     =  amp*sin(2*pi*f*t)  ;
y2    =  zeros(size(t))     ;

% Grafica con ambas seniales.
h     =  plot(t ,[y, y2], '*')  ;
grid on                         ;
xlim([0 tFin])                  ;
```

```
ylim([-3 3])                    ;
drawnow                         ;

% Bucle para la comunicacion.
N = length(t); n = 0            ;

while n < N
    n = n+1;

    % Limpiar buffer de entrada en la primera iteracion.
    if n==1
       flushinput(Udp);
    end

    % Esperar disponibilidad de datos.
    while (Udp.BytesAvailable<4)
    end

    % Leer datos.
    y2(n) = fread(Udp,1,'single');

    % Enviar datos.
    fwrite(Udp,y(n),'single');

    % Actualizar senoide modificada.
    set(h(2),'ydata',y2);
    drawnow;

end

% Cerrar puerto serie.
fclose(Udp); delete(Udp);
delete(instrfindall)
```

Código ARDUINO.

El programa de Arduino también es muy parecido al usado en la comunicación serie. En esta ocasión necesitamos, no obstante, importar una serie de librerías:

- Librería SPI. Nos sirve para comunicar la placa de Arduino UNO con la shield de Ethernet.
- Librería Ethernet. Es la librería ethernet básica que nos permite el uso del protocolo TCP/IP.
- Librería EthernetUdp. Añade a la librería anterior la posibilidad de usar comunicación UDP.

A la hora de crear el objet udp necesitamos indicar, además de las ip y los puertos, la dirección MAC (dirección que define de forma global un dispositivo físico) de la shield Ethernet.
Si se da el caso de no venir impresa bastará con inventarse una que no posea ningún otro elemento de la red.

```
1  //--------------------------------------------------------------------//
2  //                      PRUEBA COMUNICACIÓN UDP                       //
3  //--------------------------------------------------------------------//
4  // DESCRIPCIÓN:                                                       //
5  // Código para establecer una comunicación UDP entre dos términales.  //
6  // Enviamos de manera continua la información de una variable.        //
7  // Cuando detectamos información procedente del otro extremo la leemos,//
8  // modificamos, y almacenamos en la variable que enviamos.            //
9  // Ajustar las direcciones IP y los puertos a las características de la //
10 // red particular.                                                    //
11 //--------------------------------------------------------------------//
12 //                                        Paulino San Miguel Herrera //
13 //--------------------------------------------------------------------//
14
15
16 //---------------------//
17 // DEFINICIONES PREVIAS //
18 //---------------------//
19
20 // Librerias.
21 #include <SPI.h>
22 #include <Ethernet.h>
23 #include <EthernetUdp.h>
24
25 // Direcciones IP y puertos.
26 IPAddress ipPC{10,0,0,1};
27 IPAddress ipArduino(10, 0, 0, 2);
28 int puertoPC=8081, puertoArduino = 8080;
29
30 // Direccion MAC shield ethernet.
```

```
byte mac[] = {
    0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};

// Instancia UDP.
EthernetUDP Udp;

// Variables temporales.
unsigned long Ts      = 100000  ;
unsigned long t       = 0       ;

// Variable para la senoide y puntero a su primer byte.
float  sen   =  0.0 ;
byte  *p_sen = (byte*)  &sen ;


//----------------------//
//   FUNCIÓN INICIAL    //
//----------------------//

void setup( ) {

    // Establerver comunicación UDP.
    Ethernet.begin(mac,ipArduino);
    Udp.begin(puertoArduino);

}


//----------------------//
//        BUCLE         //
//----------------------//

void loop() {

    // El código se ejecuta cada decima de segundo.
    if (micros()-t >= Ts){
        t = micros();

        // Enviar senoide modificada.
        Udp.beginPacket(ipPC, puertoPC);
        Udp.write(p_sen,4);
        Udp.endPacket();

        // Leer valor de la senoide.
        Udp.parsePacket();
        if(Udp.available()>=4){
            Udp.read(p_sen,4);
            sen=-sen*2;
        }
    }

}
```

Resultados.

En esta ocasión el retraso es de dos muestras:

- Uno de los retrasos coincide con la razón indicada en la comunicación serie.
- El otro se debe a que la primera vez que se ejecuta el comando de lectura no ha llegado aún ningún paquete de información.
  En el caso de la comunicación serie este retraso no aparecía debido al uso de Serial.readBytes(), función que espera a leer los bytes indicados antes de proceder con la ejecución del programa.
  En la comunicación Udp no disponemos de ninguna función como la anterior, por tanto se usa Udp.read() que lee los bytes disponibles (se elimina una función que puede bloquear el programa a consta de ralentizar la señal una muestra).

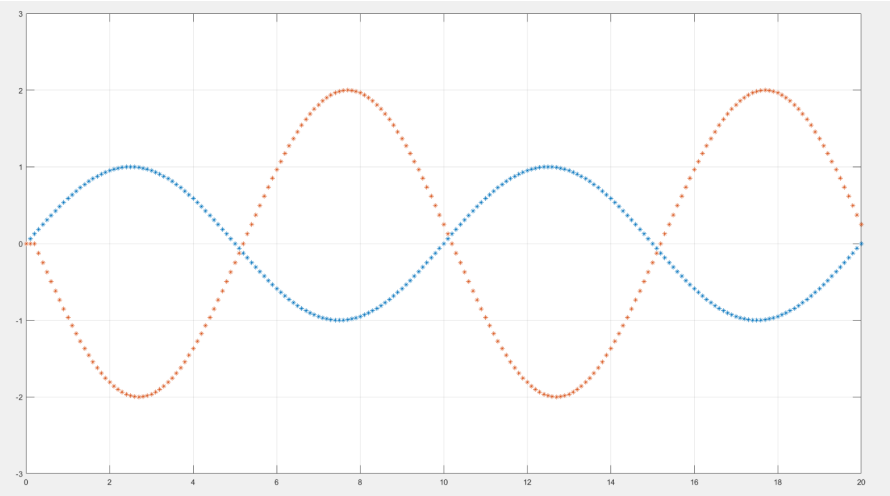

Figura 6.29: Prueba comunicación ethernet

ANEXO III:
CÓDIGOS PARCIALES DEL CONTROL DE VELOCIDAD.

CÓDIGO DEL PID.

```
1  //-------------------------------------------------------------------//
2  //                              PID                                  //
3  //-------------------------------------------------------------------//
4  // DESCRIPCIÓN:                                                      //
5  // En este código se ha programado un PID sencillo.                  //
6  // Esta pensado para añadirlo en otro programa.                      //
7  //-------------------------------------------------------------------//
8  //                                        Paulino San Miguel Herrera //
9  //-------------------------------------------------------------------//
10
11
12 //--------------------//
13 // DEFINICIONES PREVIAS //
14 //--------------------//
15
16 // Variables del PID.
17 const float         vMax  = 40.0                                ;
18
19 const float         kp      = 0.0                               ;
20 const float         ki      = 0.0                               ;
21 const float         kd      = 0.0                               ;
22 const float         Ts      = 0.1                               ;
23
24 float               t       = 0.0                               ;
25 float               velRef  = 0.0, velAct=0.0, velActAnt=0.0    ;
26
27
28 float               e       = 0.0  ,  ie    = 0.0  ,  de    = 0.0  ;
29 float               eAnt    = 0.0  ,  sPID  = 0.0                  ;
30
31
32
33 //--------------------//
34 //    FUNCIÓN INICIAL   //
35 //--------------------//
```

```
36
37  void setup(){
38  }
39
40
41  //--------------------//
42  //        BUCLE       //
43  //--------------------//
44
45  void loop(){
46
47      // Llamamos a la función cada 0.1 segundos.
48      if (micros()-t >= Ts){
49          t = micros();
50          controladorPID();
51      }
52
53  }
54
55
56  //--------------------//
57  //         PID        //
58  //--------------------//
59
60  void  controladorPID( ) {
61
62      // Variables para la siguiente iteración.
63      eAnt        =  e                        ;
64      velActAnt   =  velAct                   ;
65
66
67      // Calculamos error normalizado.
68      e       = ( velRef - velAct ) / vMax        ;
69
70      // Integral del error limitada al rango de salida.
71      ie   += ( e + eAnt  ) * 0.5 * Ts  * 1e-6    ;
72      if ( (ie*ki) > 1 ){
73          ie=1/ki;
74      }
75      else if ( (ie*ki) < -1 ){
76          ie=(-1)/ki;
77      }
78
79      // Derivada del error (usando menos derivada de la salida de la planta.)
80      velAct = velAct / vMax;
81      de     = - ( velAct - velActAnt  ) / ( Ts  * 1e-6  )     ;
82
83      // Salida PID.
84      sPID  = kp * e + ki  * ie  + kd  * de       ;
85
86      // Limitamos salida entre 1 y -1.
87      sPID  = max( -1.0  , min(  1.0 , sPID  ) )  ;
88
89  }
```

CÓDIGO DE CONTEO DE ACTIVACIONES DEL SENSOR HALL.

```
1  //-------------------------------------------------------------------//
2  //          CÓDIGO DE CONTEO DE ACTIVACIONES DEL SENSOR HALL          //
3  //-------------------------------------------------------------------//
4  // DESCRIPCIÓN:                                                      //
5  // En este código se cuenta el número de activaciones del            //
6  // sensor Hall del motor del cuadriciclo por cada vuelta             //
7  // de rueda.                                                         //
8  //-------------------------------------------------------------------//
9  //                                          Paulino San Miguel Herrera //
10 //-------------------------------------------------------------------//
11
12
13 //---------------------//
14 // DEFINICIONES PREVIAS //
15 //---------------------//
16
17 // Pin donde conectar nuestro sensor Hall.
18 byte                    iman       = 9   ;
19
20 // Pin del acelerador y valor en su salida.
21 byte                    acelerador = 10  ;
22 byte                    valor      = 160 ;
23
24 // Variable para almacenar pulsos del sensor Hall del motor.
25 volatile unsigned long  pulsos     = 0   ;
26
27 //---------------------//
28 //    FUNCIÓN INICIAL   //
29 //---------------------//
30
31
32 void setup()
33 {
```

```
34    // Iniciar comunicación serie.
35    Serial.begin(9600);
36    // Interrupción 0 (pin 2) para detectar cambios en
37    // el sensor Hall del motor.
38    //attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
39    attachInterrupt(0, sensorHall, CHANGE);
40    // Definir nuestro sensor Hall como entrada.
41    pinMode(iman,INPUT);
42    // Definir señal del acelerador como slaida.
43    pinMode(acelerador,OUTPUT);
44    // Señal deseada para el acelerador a la salida.
45    analogWrite(acelerador,valor);
46  }
47
48
49  //--------------------//
50  //        BUCLE       //
51  //--------------------//
52
53  void loop() {
54    // Cuando se activa nuestro sensor Hall mostrar por
55    // pantalla el número de pulsos enviados por el sensor
56    // Hall del motor.
57    if (digitalRead(iman)==LOW){
58  //    Serial.print("Pulsos detectados: ");
59      Serial.println(pulsos);
60      // Resetear contador.
61      pulsos=0;
62      // Un solo dato por cada detección de imán.
63      while (digitalRead(iman)==LOW){}
64    }
65  }
66
67
68  //--------------------//
69  //     INTERRUPCIÓN    //
70  //--------------------//
```

```
71
72 // Cada pulso del sensor Hall del motor aumenta el contador.
73 void sensorHall(){
74   pulsos++;
75 }
```

CÓDIGO DEL SUBSISTEMA DE ACELERACIÓN.

```
//-------------------------------------------------------------------------//
//                    CÓDIGO SUBSISTEMA DE VELOCIDAD                       //
//-------------------------------------------------------------------------//
// DESCRIPCIÓN:                                                            //
// En este código se realiza el control de velocidad                       //
// únicamente con el subsistema de aceleración.                            //
//-------------------------------------------------------------------------//
//                                              Paulino San Miguel Herrera //
//-------------------------------------------------------------------------//


//----------------------//
// DEFINICIONES PREVIAS //
//----------------------//

// Entradas y salidas de Arduino.
const byte  pinInt      = 2   ;
const byte  acelerador  = 9   ;
const byte  freno       = 10  ;

// Variables temporales.
unsigned long Ts      = 100000  ;
unsigned long t       = 0       ;

// Constantes.
const byte  ppv          = 80                        ;
const float pi           = 3.14159                   ;
const float f            = 0                         ;
const float velMax       = 40                        ;
const float longRueda    = 1.7650                    ;
const float k            = longRueda/ppv/(1e-6*Ts)*3.6 ;

// Variables del PID.
const float kp    = 0.5,    ki        = 0.0,    kd    = 0.0 ;
float       e     = 0.0,    ie        = 0.0,    de    = 0.0 ;
float       eAnt  = 0.0,    velActAnt = 0.0,    sPID  = 0.0 ;

// Variables para velocidad.
float velAct  = 0.0 ;
float velRef  = 0.0 ;

// Punteros al primer byte de la variable correspondiente.
byte  *p_velAct = (byte*)  &velAct ;
byte  *p_velRef = (byte*)  &velRef ;
byte  *p_sPID   = (byte*)  &sPID   ;

// Contador de la interrupción.
volatile unsigned long  pulsos  = 0 ;


//----------------------//
//    FUNCIÓN INICIAL   //
//----------------------//

void setup( ) {

  // Entradas y salidas.
```

```
  pinMode(acelerador, OUTPUT);
  pinMode(freno     , OUTPUT);

  // Aumento frecuencia timer1;
  TCCR1B=TCCR1B & 0b11111000 | 0x01;

  // Comunicación serie.
  Serial.begin(115200);

  // Interrupción.
  attachInterrupt(digitalPinToInterrupt(pinInt), activacionSensor, CHANGE);

  // Comenzamos con acelerador a cero.
  analogWrite(acelerador,0);

}


//---------------------//
//        BUCLE        //
//---------------------//

void loop(  ) {

  // Cada 0.1 segundos ejecutamos el bucle.
  if (micros()-t >= Ts){
      t = micros();
      velAct = f*velAct+(1-f)*pulsos*k;

      // Enviar velocidad actual, leer velocidad deseada.
      Serial.write( p_velAct  , 4 )              ;
      Serial.write( p_sPID  , 4 )              ;

      if (Serial.available() >= 4){
          Serial.readBytes(  p_velRef  , 4 ) ;

          // Obtener señal acelerador/freno.
          controladorPID();

          // Activar acelerador o freno según proceda.
          if( sPID >=  0 )  {
            analogWrite(  acelerador  , (byte)( sPID  * 255  ) )  ;
          }
          else  {
            analogWrite(  acelerador  , 0 )                     ;
          }
      }
      pulsos = 0;
  }

}


//---------------------//
//     INTERRUPCIÓN    //
//---------------------//

void  activacionSensor( ) {
  pulsos  ++  ;
}

```

```
119
120 //---------------------//
121 //         PID         //
122 //---------------------//
123
124 void  controladorPID( ) {
125
126     // Variables para la siguiente iteración.
127     eAnt         =  e                          ;
128     velActAnt    =  velAct                     ;
129
130
131     // Calculamos error normalizado.
132     e       = ( velRef - velAct ) / velMax        ;
133
134     // Integral del error limitada al rango de salida.
135     ie   += ( e + eAnt  ) * 0.5 * Ts  * 1e-6    ;
136     if ( (ie*ki) > 1 ){
137         ie=1/ki;
138     }
139     else if ( (ie*ki) < -1 ){
140         ie=(-1)/ki;
141     }
142
143     // Derivada del error (usando menos derivada de la salida de la planta.)
144     velAct = velAct / velMax;
145     de     = - ( velAct - velActAnt  ) / ( Ts  * 1e-6  )     ;
146
147     // Salida PID.
148     sPID  = kp * e + ki  * ie  + kd  * de       ;
149
150     // Limitamos salida entre 1 y -1.
151     sPID  = max( -1.0  , min(  1.0 , sPID  ) )  ;
152
153 }
```

CÓDIGO DEL SUBSISTEMA DE FRENADO.

```
//-------------------------------------------------------------------------//
//                    CÓDIGO SUBSISTEMA DE FRENADO                         //
//-------------------------------------------------------------------------//
// DESCRIPCIÓN:                                                            //
// En este código se muestran las modificaciones necesarias                //
// a realizar en el código del subsistema de aceleración para              //
// poder incluir el freno.                                                 //
// Solo se muestra como varía el bucle principal por lo que este           //
// código NO SE PUEDE EJECUTAR si previemente no se complementa con el     //
// código del subsistema de aceleración.                                   //
//-------------------------------------------------------------------------//
//                                              Paulino San Miguel Herrera //
//-------------------------------------------------------------------------//


//---------------------//
//        BUCLE        //
//---------------------//

void loop(  ) {

  // Cada 0.1 segundos ejecutamos el bucle.
  if (micros()-t >= Ts){
      t = micros();
      velAct = f*velAct+(1-f)*pulsos*k;

      // Enviar velocidad actual y señal PID.
      Serial.write( p_velAct  , 4 )              ;
      Serial.write( p_sPID  , 4 )                ;

      //leer velocidad deseada y obtener señales de control.
      if (Serial.available() >= 4){
          Serial.readBytes(  p_velRef  , 4 ) ;

          // Obtener señal acelerador/freno.
          controladorPID();

          // Activar acelerador o freno según proceda.
          if( sPID >=  0 )  {
            analogWrite(  acelerador  , (byte)( sPID  * 255  ) )  ;
            analogWrite(  freno  , 0 )                            ;
            // Siempre que la señal del pid sea positiva liberamos
            // la presión del circuito de frenos.
            digitalWrite(  eVal  , 0 )                            ;
          }
          else  {
            analogWrite(  acelerador  , 0 )                       ;
            analogWrite(  freno  ,      (byte)( -sPID  * 255  ) ) ;
            // Si la señal de freno aumenta en magnitud conectamos la
            // electroválvula aumentando la presión del circuito de
            // frenos, en caso contrario la desconectamos liberando
            // dicha presión.
            if (sPID < sPIDant){
              digitalWrite(  eVal  , 1 )                          ;
            }
            else{
              digitalWrite(  eVal  , 0 )                          ;
```

```
58              }
59              sPIDant=sPID;
60              }
61      }
62      pulsos = 0;
63   }
64
65 }
```

ANEXO IV:
CÓDIGO DEL CONTROL DE VELOCIDAD FINAL.

```
1  //-------------------------------------------------------------------//
2  //                  CÓDIGO CONTROL DE VELOCIDAD                      //
3  //-------------------------------------------------------------------//
4  // DESCRIPCIÓN:                                                      //
5  // En este código se realiza el control de velocidad final           //
6  // Se juntan los códigos códigos del subsistema de aceleración       //
7  // y del subsistema de freno y se aplica comunicación ethernet.      //
8  //-------------------------------------------------------------------//
9  //                                          Paulino San Miguel Herrera //
10 //-------------------------------------------------------------------//
11
12
13 //---------------------//
14 // DEFINICIONES PREVIAS //
15 //---------------------//
16
17 // Librerias.
18 #include <SPI.h>
19 #include <Ethernet.h>
20 #include <EthernetUdp.h>
21
22 // Direcciones IP y puertos.
23 IPAddress ipPC{10,0,0,1};
24 IPAddress ipArduino(10, 0, 0, 2);
25 int puertoPC=8081, puertoArduino = 8080;
26
27 // Direccion MAC shield ethernet.
28 byte mac[] = {
29     0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
30
31 // Instancia UDP.
32 EthernetUDP Udp;
33
34 // Entradas y salidas de Arduino.
35 const byte  pinInt      = 2   ;
36 const byte  acelerador  = 9   ;
37 const byte  freno       = 6   ;
38 const byte  eVal        = 7   ;
```

```
39
40  // Variables temporales.
41  unsigned long Ts      = 100000  ;
42  unsigned long t       = 0       ;
43
44  // Constantes.
45  const byte  ppv           = 80                          ;
46  const float pi            = 3.14159                     ;
47  const float f             = 0                           ;
48  const float velMax        = 40                          ;
49  const float longRueda     = 1.7650                      ;
50  const float k             = longRueda/ppv/(1e-6*Ts)*3.6 ;
51
52  // Variables del PID.
53  const float kp    = 0.5,    ki        = 0.0,    kd    = 0.0 ;
54  float       e     = 0.0,    ie        = 0.0,    de    = 0.0 ;
55  float       eAnt  = 0.0,    velActAnt = 0.0,    sPID  = 0.0 ;
56  float       sPIDant=0.0;
57
58  // Variables para velocidad.
59  float velAct  = 0.0 ;
60  float velRef  = 0.0 ;
61
62  // Punteros al primer byte de la variable correspondiente.
63  byte  *p_velAct = (byte*)  &velAct ;
64  byte  *p_velRef = (byte*)  &velRef ;
65  byte  *p_sPID   = (byte*)  &sPID   ;
66
67  // Contador de la interrupción.
68  volatile unsigned long  pulsos  = 0 ;
69
70
71  //----------------------//
72  //    FUNCIÓN INICIAL   //
73  //----------------------//
74
75  void setup( ) {
76
77    // Establerver comunicación UDP.
78    Ethernet.begin(mac,ipArduino);
79    Udp.begin(puertoArduino);
80
81    // Aumento frecuencia timer1;
82    TCCR1B=TCCR1B & 0b11111000 | 0x01;
83
84    // Interrupción.
85    attachInterrupt(digitalPinToInterrupt(pinInt), activacionSensor, CHANGE);
86
87    // Entradas y salidas.
88    pinMode(acelerador, OUTPUT);
89    pinMode(freno     , OUTPUT);
90    pinMode(eVal      , OUTPUT);
91    pinMode(pinInt    , INPUT );
92
93    // Comenzamos con el sistema en reposo.
94    analogWrite(acelerador,0)   ;
95    analogWrite(freno     ,0)   ;
96    digitalWrite(eVal     ,HIGH);
97
98  }
99
```

```
100
101 //---------------------//
102 //        BUCLE        //
103 //---------------------//
104
105 void loop(  ) {
106
107   // Cada 0.1 segundos ejecutamos el bucle.
108   if (micros()-t >= Ts){
109       t = micros();
110       velAct = f*velAct+(1-f)*pulsos*k;
111
112       // Enviar velocidad actual y señal PID.
113       Udp.beginPacket(ipPC, puertoPC);
114       Udp.write(p_velAct,4);
115       Udp.endPacket();
116       Udp.beginPacket(ipPC, puertoPC);
117       Udp.write(p_sPID,4);
118       Udp.endPacket();
119
120       // Leer velocidad deseada y obtener señales de control.
121       Udp.parsePacket();
122       if(Udp.available()>=4){
123           Udp.read(p_sen,4);
124
125           // Obtener señal acelerador/freno.
126           controladorPID();
127
128           // Activar acelerador o freno según proceda.
129           if( sPID >=  0 )  {
130             analogWrite(  acelerador  , (byte)( sPID  * 255  ) )  ;
131             analogWrite(  freno  , 0 )                            ;
132             // Siempre que la señal del pid sea positiva liberamos
133             // la presión del circuito de frenos.
134             digitalWrite(  eVal  , 0 )                            ;
135           }
136           else  {
137             analogWrite(  acelerador  , 0 )                       ;
138             analogWrite(  freno  ,      (byte)( -sPID  * 255  ) ) ;
139             // Si la señal de freno aumenta en magnitud conectamos la
140             // electroválvula aumentando la presión del circuito de
141             // frenos, en caso contrario la desconectamos liberando
142             // dicha presión.
143             if (sPID < sPIDant){
144               digitalWrite(  eVal  , 1 )                          ;
145             }
146             else{
147               digitalWrite(  eVal  , 0 )                          ;
148             }
149             sPIDant=sPID;
150             }
151       }
152       pulsos = 0;
153   }
154
155 }
156
157
158 //---------------------//
159 //     INTERRUPCIÓN    //
160 //---------------------//
```

```
161
162 void activacionSensor( ) {
163   pulsos ++ ;
164 }
165
166
167 //---------------------//
168 //        PID          //
169 //---------------------//
170
171 void controladorPID( ) {
172
173     // Variables para la siguiente iteración.
174     eAnt        = e                          ;
175     velActAnt   = velAct                     ;
176
177
178     // Calculamos error normalizado.
179     e       = ( velRef - velAct ) / velMax       ;
180
181     // Integral del error limitada al rango de salida.
182     ie   += ( e + eAnt  ) * 0.5 * Ts  * 1e-6    ;
183     if ( (ie*ki) > 1 ){
184         ie=1/ki;
185     }
186     else if ( (ie*ki) < -1 ){
187         ie=(-1)/ki;
188     }
189
190     // Derivada del error (usando menos derivada de la salida de la planta.)
191     velAct = velAct / velMax;
192     de     = - ( velAct - velActAnt  ) / ( Ts  * 1e-6  )     ;
193
194     // Salida PID.
195     sPID  = kp * e + ki  * ie  + kd  * de       ;
196
197     // Limitamos salida entre 1 y -1.
198     sPID  = max( -1.0  , min(  1.0 , sPID  ) )  ;
199
200 }
```

ANEXO V:
HOJAS DE CARACTERÍSTICAS.

- Microcontrolador de Arduino UNO rev3 (ATMega358p).
  http://www.atmel.com/images/atmel-42735-8-bit-avr-microcontroller-atmega328-328p_datasheet.pdf
  [Consulta: septiembre 2017].
- Microcontrolador de la Ethernet Shield (W5500).
  http://wizwiki.net/wiki/lib/exe/fetch.php?media=products:w5500:w5500_ds_v106e_141230.pdf
  [Consulta: septiembre 2017].
- Controladora Curtis 1268.
  http://curtisinstruments.com/Uploads/DataSheets/1268%20manual(12D).pdf
  [Consulta: septiembre 2017]
- Central hidráulica compacta HAWE HYDRAULIK tipo NPC.
  http://downloads.hawe.com/7/9/D7940-en.pdf
  [Consulta: septiembre 2017].
- Transistor de potencia de nivel lógico IRLZ44n.
  http://docs-europe.electrocomponents.com/webdocs/0791/0900766b807913d7.pdf
  [Consulta: septiembre 2017].
- Operacional LM358.
  http://www.ti.com/lit/ds/symlink/lm258a.pdf
  [Consulta: septiembre 2017].

ANEXO VI:
COSTE DEL SISTEMA.

En la siguiente tabla se recoge el precio que supone el presente control de velocidad (incluyendo componentes recogidos del propio laboratorio).
No se han tenido en cuenta los elementos previamente instalados en el vehículo experimental.

COMPONENTE	PRECIO UNITARIO (€/ud)	UNIDADES	SUBTOTAL
Placa de desarrollo Arduino UNO	21,25	1	21,25
Modulo Ethernet para Arduino sin PoE	19,98	1	19,98
Resistencias 10 kΩ	0,05	3	0,15
Resistencia 3kΩ	0,05	1	0,05
Resistencias 220 Ω	0,05	2	0,1
Condensador electrolítico 1uF	0,05	1	0,05
Condensador cerámico 1nF	0,05	1	0,05
Amplificador Operacional LM358	0,37	1	0,37
Transistor de potencia IRLZ44n	0,64	2	1,28
Diodo de conmutación DHG40C600HB	4,46	2	8,92
		TOTAL	52,2

Los precios son orientativos y sujetos a fluctuaciones, obtenidos del distribuidor RS componentes a septiembre de 2017.

Bibliografía

[1] https://i.kinja-img.com/gawker-media/image/upload/s--NSSejb_K--/c_scale,f_auto,fl_progressive,q_80,w_800/18h644h3y4tbmjpg.jpg
[Consulta: septiembre 2017]

[2] https://www.supercars.net/blog/wp-content/uploads/2016/04/1769_Cugnot_SteamTractor6.jpg
[Consulta: septiembre 2017]

[3] https://upload.wikimedia.org/wikipedia/commons/thumb/e/ea/Jamais_contente.jpg/300px-Jamais_contente.jpg
[Consulta: septiembre 2017]

[4] Fraile Mora, J. J., (2003), Máquinas eléctricas 5ed, Madrid, McGraw-Hill.

[5] https://i1.wp.com/submarinewarriors.com/wp-content/uploads/2012/02/submarine-power-system.jpg
[Consulta: septiembre 2017]

[6] https://upload.wikimedia.org/wikipedia/commons/thumb/3/3a/Roadster_2.5_windmills_trimmed.jpg/1200px-Roadster_2.5_windmills_trimmed.jpg
[Consulta: septiembre 2017]

[7] http://www.rimac-automobili.com/media/2202/exterior-design-overview-img.jpg?mode=pad&width=960&rnd=131030433260000000
[Consulta: septiembre 2017]

[8] https://www.tesla.com/tesla_theme/assets/img/models/v1.0/slideshow/Blue_Sunset-1440.jpg?20170420
[Consulta: septiembre 2017]

[9] https://www.nissanusa.com/content/dam/nissan/vehicles/2017/leaf/og-share.jpg
[Consulta: septiembre 2017]

[10] http://www.electric-cars-are-for-girls.com/electric-car-motors-ac-vs-dc.html
[Consulta: septiembre 2017]

[11] https://en.wikipedia.org/wiki/Electric-vehicle_battery
[Consulta: septiembre 2017]

[12] https://yourbrakes.com/how-brakes-work-brake-systems/
[Consulta: septiembre 2017]

[13] https://yourbrakes.com/how-brakes-work-brake-systems/
[Consulta: septiembre 2017]

[14] http://s.hswstatic.com/gif/regenerative-brake-diagram.jpg
[Consulta: septiembre 2017]

[15] http://hanabi.autoweek.com/sites/default/files/styles/gen-932-524/public/front-lights-view-1_0.jpg?itok=PbpiUqUV
[Consulta: septiembre 2017]

[16] Reyes Cortés, F., (2015), Arduino: aplicaciones en Robótica, mecatrónica e ingenierías, Mexico, Alfaomega, Barcelona, Marcombo.

[17] http://www.nexcom.com/UploadFiles/FCKeditor/image/Product_News/100504_NISE3500M/Info_NISE3500(1).jpg
[Consulta: septiembre 2017]

[18] https://upload.wikimedia.org/wikipedia/commons/2/24/Feedback_loop_with_descriptions.svg
[Consulta: septiembre 2017]

[19] J. Astrom, K. / Hägglund, T., (1995), PID controllers 2ed, North Carolina, Instrument Society of America.

[20] http://blascarr.com/wp-content/uploads/2016/08/pid.jpg.png
[Consulta: septiembre 2017]

[21] http://www.drcruzan.com/NumericalIntegration.html
[Consulta: septiembre 2017]

[22] http://brettbeauregard.com/blog/2011/04/improving-the-beginner%E2%80%99s-pid-reset-windup/
[Consulta: septiembre 2017]

[23] http://brettbeauregard.com/blog/2011/04/improving-the-beginner%E2%80%99s-pid-reset-windup/
[Consulta: septiembre 2017]

[24] https://learn.mikroe.com/ebooks/wp-content/uploads/sites/5/2016/01/pic-microcontrollers-programming-in-basic-chapter-03-image-68.gif?x85026
[Consulta: septiembre 2017]

[25] Winder, S., (2001), Filter Design, Oxford, Newnes.

[26] http://www.electronics-tutorials.ws/filter/fil20a.gif
[Consulta: septiembre 2017]

[27] https://image.jimcdn.com/app/cms/image/transf/dimension=512x10000:format=jpg/path/sc10897969bb6a5f1/image/i6a1091b2208ad23a/version/1365187838/image.jpg
[Consulta: septiembre 2017]

[28] https://images2.cardekho.com/images/carNewsEditorImages/20160301_162054/18054/others-guide0.jpg
[Consulta: septiembre 2017]

[29] http://www.target-hydraulics.com/hydraulic-power-pack-ebook/#ebook1.1
[Consulta: septiembre 2017]

[30] http://www.pumpschool.com/images/extgear2.gif
[Consulta: septiembre 2017]

[31] W. Erickson, R. / Maksimovic, D., (2001), Fundamentals of power electronics 2ed, Norwell, Massachusetts.

[32] https://www.safaribooksonline.com/library/view/introduction-to-digital/9780470900550/images/ch005-f002.jpg
[Consulta: septiembre 2017]

[33] S. Sedra, A. / C. Smith, K., (2011), Microelectronic circuits 6ed, New York, Oxford University Press.

[34] Reas, C. / Fry, B, (2015), MAKE: Getting Started with Processing 2ed, San francisco, Maker Media inc.

[35] https://www.blender.org/support/tutorials/
[Consulta: septiembre 2017]