Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Diseño, realización y control de un robot autoequilibrado de bajo coste basado en una Raspberry Pi

Autor: José Ignacio Cámara Molina

Tutores: Ignacio Alvarado Aldea

Pablo Krupa García

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Diseño, realización y control de un robot autoequilibrado de bajo coste basado en una Raspberry Pi

Autor:
José Ignacio Cámara Molina

Tutor:
Ignacio Alvarado Aldea
Pablo Krupa García

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Diseño, realización y control de un robot autoequilibrado de bajo coste basado en una Raspberry Pi

Autor: José Ignacio Cámara Molina
Tutores: Ignacio Alvarado Aldea
Pablo Krupa García

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Resumen

El presente trabajo trata de diseñar, construir y controlar un robot autobalanceado basado en la configuración del péndulo invertido, donde se aplicarán y compararán diferentes estrategias de control aprendidas durante los años de carrera.

En resumidas cuentas, el objetivo del trabajo es conseguir fabricar un vehículo de bajo coste que pueda ser replicado, complementado y, en su caso, mejorado por otros usuarios. Se parte de la idea de trabajos anteriores desarrollados en el departamento de Ingeniería Automática de la Escuela superior de Ingeniería de Sevilla (ETSI), donde a pesar de sus similitudes, presenta grandes diferencias que se irán vislumbrando a lo largo de los capítulos.

Se pretende conseguir que el vehículo, basado en la configuración del péndulo invertido, sea capaz de mantener su posición de equilibrio en todo momento, pudiendo desplazarse y girar sin que pierda dicha posición.

En primer lugar, se analizarán todos los componentes que formarán parte del vehículo, para poder diseñar el cuerpo acorde a sus dimensiones y características. Posteriormente, se diseñará el cuerpo del vehículo con programas de modelado tridimensional para su impresión en 3D. Se ha diseñado una PCB que aglutinará gran parte de los componentes electrónicos, reduciendo el uso de cables, mejorando con ello sus conexiones.

Una vez conseguido el hardware del vehículo, se procederá al análisis del modelo matemático y su posterior linealizado para aplicar técnicas de control. Se diseñarán diferentes controladores para mantener la estabilidad. Se diseñará un controlador LQR, posteriormente se añadirá un término integral al LQR y se finalizará con la implementación de un control MPC.

La última etapa del trabajo consistirá en implementar el control del vehículo en las dos plataformas utilizadas para ello, el microcontrolador Arduino para el control de los motores y la microcomputadora Raspberry Pi para el control en alto nivel. Aquí radica la gran diferencia con respecto a trabajos anteriores y que abre un gran abanico de posibilidades para trabajos futuros. La potencia de cálculo de la Rpi en comparación con Arduino nos posibilita la implementación de todo tipo de controladores, mejorando las prestaciones del vehículo.

Por último, se añade en la memoria un capítulo de resultados donde se puede apreciar el funcionamiento de las técnicas de control ante diferentes tipos de experimentos, como puede ser perturbaciones tipo pulso, variaciones en la referencia de velocidad de las ruedas o ascenso y descenso de una rampa, entre otros.

Índice

Índice de Figuras

Índice de Tablas

Notación

RPi	Raspberry Pi
RTC	Real Time System (Sistema en Tiempo Real)
3*D*	Tridimensional
A.C	Corriente alterna
D.C	Corriente continua
PWM	Modulación por Ancho de Pulso
NEMA	National Electrical Manufacturers Association
SPR	Steps Per Second (Pasos por segundo)
SPI	Serial Peripheral Interface (Interfaz de periféricos en serie)
IDE	Integrated Development Environment (entorno de desarrollo integrado)
ADC	Analog to Digital Converter (convertidor analógico-digital)
DAC	Digital to Analog Converter (convertidor digital-analógico)
I2C	Protocolo de comunicación inter-integrado de circuito
CPU	Central Processing Unit (unidad central de procesos)
USB	Universal Serial Bus (bus serie universal)
SRAM	Static Random Acess Memory (Memoria estática de acceso aleatorio)
MHz	Megahercios
I/*O*	Input/Output (entrada/salida)
mm	Milímetro
A	Amperio
V	Voltio
mV	Milivoltio
ms	Milisegundo
ns	Nanosegundo
μs	microsegundo
GND	Ground (Tierra/Masa)
IMU	Inertial Measurement Unit (Unidad de Medida Inercial)
PLA	Ácido poliláctico

1 Introducción

Todo ingeniero debe aspirar a poder crear su propio mundo, su propia tecnología. El mundo de la electrónica permite crear, analizar y simular cualquier sistema físico. Plasmar una idea en un proyecto físico es un desafío complejo, pero a la vez intrépido y gratificante.

1.1 Motivación

Muchas veces hemos intentado mantener en equilibrio una barra sobre la palma de nuestra mano, ante la inestabilidad que presenta, automáticamente nuestra respuesta es mover la mano para evitar que esta caiga, manteniéndola así en su posición de equilibrio. El mecanismo que actúa es simple, el cerebro actuará como un sensor que capta la inclinación de la barra, percibiendo que esta se va a caer, y automáticamente lanzará un estimulo para que se produzca un movimiento de la mano, evitando así que se produzca la caída. Sin darnos cuenta estábamos delante de lo que hoy es este trabajo fin de grado.

La idea anterior se asemeja al concepto de equilibrio de un péndulo invertido. Sin duda alguna, el péndulo invertido es uno de los problemas clásicos dentro del campo de la automática y del control que más se ha estudiado y que mayor interés suscitan entre los estudiantes y profesores de ingeniería. Todo ello debido a su relativa facilidad en comparación con otros experimentos, la posibilidad de implementar diferentes estrategias de control, así como la aplicación práctica de los diferentes conceptos teóricos estudiados en diferentes asignaturas a lo largo de los cursos de estudio universitario. En resumidas cuentas, es un gran sistema de pruebas para implementar diferentes estrategias de control.

El mecanismo del vehículo basado en el péndulo invertido se compone de una parte fija y una parte móvil, cuyo objetivo es mantener estable el vehículo, entendiendo como estable la propiedad que presenta el vehículo de mantenerse en equilibrio o de volver a dicho estado tras sufrir una perturbación. Si no dotáramos al mecanismo de un sistema de control, al desplazarlo de su punto de equilibrio, por la fuerza de la gravedad, simplemente se caería. Estamos presente ante un sistema no lineal e inestable.

1.2 Objetivo

El documento ante el que estamos presente trata de construir un robot autobalanceado sobre dos ruedas basado en la configuración del péndulo invertido, donde su centro de masas se encontrará por encima de su eje de rotación, lo que hace que se produzca un equilibrio inestable, es decir, cualquier perturbación haría al sistema perder su posición. Por ello, nuestro esfuerzo será devolver al vehículo a esta posición de equilibrio, siendo capaz de compensar las perturbaciones que pueda sufrir en su funcionamiento.

Por tanto, el objetivo será dotar a un vehículo tipo péndulo invertido del mecanismo de control que evite que este se caiga, manteniéndolo en movimiento y en su posición vertical de equilibrio. Para ello, contaremos con un sensor que nos proporcione en cada momento la información sobre la velocidad de caída e inclinación del vehículo con respecto a su vertical. Con los datos obtenidos por el sensor, y mediante dos motores paso a paso conectados a cada una de las ruedas, dotaremos a estas de la aceleración necesaria para mantener la posición estable. El objetivo de control será por tanto la aceleración a proporcionar a cada rueda por parte de los motores, dicha aceleración será implementada a través del microcontrolador Arduino, siendo la Raspberry Pi el cerebro de nuestro proyecto, y por ello, el que realice las tareas de control a alto nivel calculando la aceleración que se debe aplicar. Cada rueda deberá tener la posibilidad de girar a diferente velocidad para dotar al vehículo de capacidad de giro, por ello las aceleraciones deberán ser independientes para ambas ruedas.

Se pretende poder controlar remotamente el vehículo, para ello, se va a añadir un módulo bluetooth. De tal manera que, mediante una App y desde nuestro teléfono móvil, se pueda controlar un joystick que muevan el vehículo en ambos lados y hacia delante y atrás.

1.3 Punto de partida

El presente trabajo trata de presentar una variante de los continuos trabajos que el departamento de automática de la Escuela Técnica Superior de Ingeniería de Sevilla, en adelante ETSI, ha ido realizando en los últimos años. En trabajos anteriores se ha diseñado, simulado y fabricado un vehículo basado en el péndulo invertido de bajo coste mediante el microcontrolador Arduino.

Bajo las mismas premisas que en trabajos anteriores, se ha optimizado para mantener el coste económico de fabricación a sus niveles más bajos, utilizando para ello componentes electrónicos de bajo coste y amplio uso en el mundo electrónico, testados en multitud de experimentos.

En la figura 1.1 podemos apreciar el vehículo utilizado en los trabajos realizados por Antonio Croche Navarro [18] y Cecilia González González [8], siendo este uno de los primeros vehículos con los que se inició esta línea de experimentos. Vemos que presentan grandes diferencias con respecto a nuestro proyecto, tanto en el hardware como en software. En ambos trabajos se utilizó el microcontrolador Arduino Mega como cerebro del proyecto. En cuanto al chasis del vehículo, se utilizo una estructura de aluminio y madera y dos motores de corriente continua con reductora para el movimiento de las ruedas, lo cual hacía superior el volumen y peso del vehículo. Los inconvenientes de dichos materiales fueron explicados por Antonio Borja Conde en su trabajo [4]

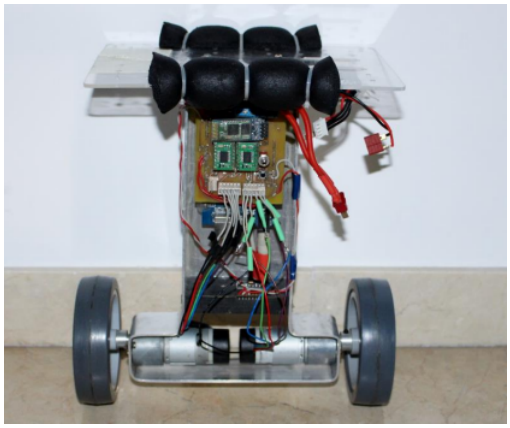


Figura 1.1 Vehículo autobalanceado Antonio Croche-Cecilia González.

Posteriormente, Jose Antonio Borja Conde [4] y Nicolás Cortés Fernández [6] realizaron una serie de cambios sustanciales, pasando de utilizar motores de corriente continua a motores paso a paso, los cuales son mucho más precisos al poder aplicar velocidades similares a la referencia, prescindiendo del uso de reductoras. Ambos trabajos continuaban con el uso del microcontrolador Arduino para las tareas de control, cambiando la versión del Arduino Mega al Arduino Uno. En cuanto al chasis del vehículo, este paso a ser de PLA obtenido mediante impresión 3D el primero de ellos, y de madera el segundo.

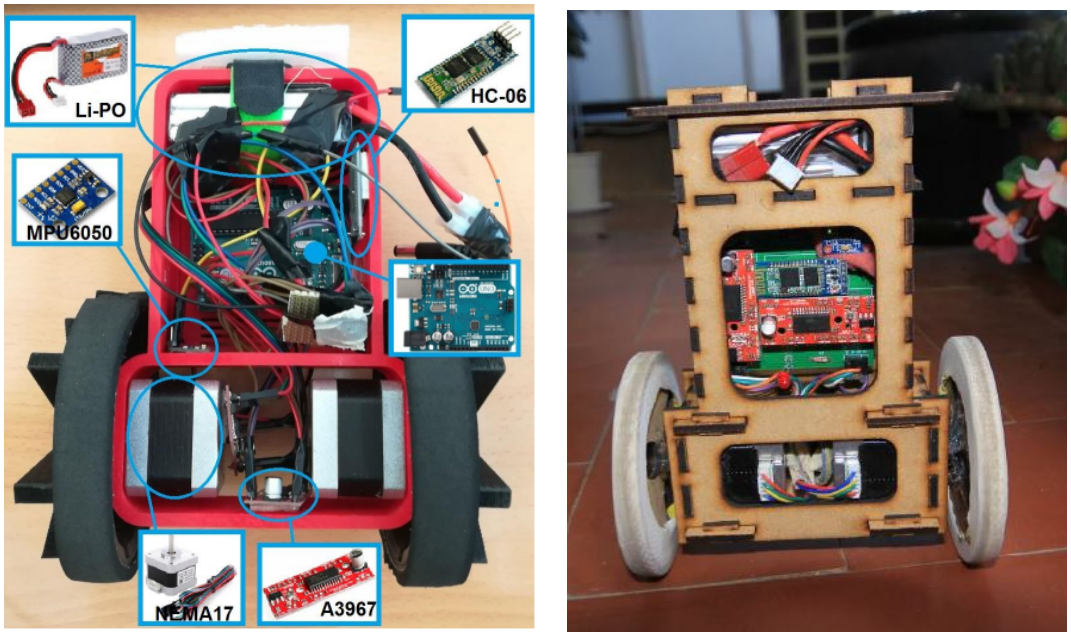


Figura 1.2 (a) Vehículo Jose A. Borja. (b) Vehículo Nicolás Cortés .

Como hemos indicado anteriormente, el presente trabajo trata de continuar con la línea iniciada, mejorando las características del vehículo. Todos los trabajos citados hacían uso del microcontrolador Arduino para implementar las tareas de control, aquí radica la principal diferencia, pues para el control de nuestro vehículo se ha optado por el uso de la microcomputadora Raspberry Pi modelo 3, más adelante citaremos las mejoras que aporta este cambio. El uso del microcontrolador Arduino seguirá presente para el control de los motores, usaremos la versión Nano, de características similares a la versión Uno pero de tamaño reducido. Al igual que el anterior vehículo, seguirá usándose dos motores paso a paso.

1.4 Requisitos

Todo proyecto robótico debe cumplir una serie de requisitos con el fin de que la tarea para la que ha sido diseñada sea funcional. Entre los diferentes requisitos del presente trabaja podemos destacar:

1. Autonomía: Permitiendo que el robot funcione sin intervención humana después de su calibrado.
2. Bajo Coste: Debe ser un proyecto que se abastezca de componentes de bajo coste para que pueda ser replicado en trabajos futuros. La RPi, así como los drivers y motores que utilizaremos cumplen este requisito.
3. Portabilidad: Debe ser un proyecto capaz de ser retomado y replicado por otros alumnos.

4. Ligero y de fácil transporte: No dotando al robot de elementos voluminosos ni pesados, facilitando con ello movimientos más veloces.
5. Funcional: Debemos asegurar su funcionalidad, estudiando en particular todas las posibles opciones que nos podrían dar fallos.

1.5 Estructura de la memoria

Podemos dividir el trabajo en tres grandes bloques bien diferenciados, pero dependiente entre ellos. El primero de ellos será el estudio de todos los componentes que forman parte de vehículo, así como el diseño y fabricación del chasis que los englobe. La segunda parte contendrá el estudio del modelo y de los diferentes métodos de control a aplicar, es decir, el diseño y simulación del sistema de control. La última parte consistirá en la implementación del sistema de control elegido, así como la discusión de los resultados obtenidos.

Dentro de los bloques comentados anteriormente, vamos a dividir la presente memoria en una serie de capítulos con el objetivo de dotarla de una mayor simplicidad y así facilitar la lectura de la misma.

- Capítulo 1: Se realiza una breve introducción del trabajo, dando una imagen general de lo que nos vamos a encontrar en los capítulos venideros. Explica la motivación, así como los objetivos y estructura del trabajo.
- Capítulo 2: Dedicaremos íntegramente este capítulo a exponer y explicar los diferentes elementos electrónicos que formarán parte del proyecto, desde el acelerómetro y giroscopio para conocer la posición y velocidad del vehículo, hasta los motores que dotarán de movilidad a nuestro robot.
- Capítulo 3: Realizaremos un estudio cinemático del modelo, obteniendo las ecuaciones del modelo físico-matemático, así como el modelo linealizado para poder aplicar controladores en variables de estado.
- Capítulo 4: Dedicaremos este capítulo a mostrar las diferentes estrategias de control que hemos aplicado al proyecto. Implementaremos un control LQR, LQR con efecto integral y MPC.
- Capítulo 5: Este capítulo estará dedica a explicar el software del vehículo. Se explicará las estrategias de programación en ambos dispositivos, mostrando el código y detallando su funcionamiento.
- Capítulo 6: En este capítulo se mostrarán y analizarán detenidamente los resultados obtenidos mediante gráficas que muestran su funcionamiento. Se expondrá el vehículo a diferentes experimentos como pueden ser perturbaciones de su posición de equilibrio, variaciones en la referencia de velocidad de las ruedas o ascenso y descenso de un rampa.
- Capítulo 7: Para finalizar la memoria se incluirá este capítulo donde se mostrarán las conclusiones del trabajo y se expondrán futuras lineas de investigación y mejora.

1.6 Herramietas Sofware

A lo largo del trabajo necesitaremos el uso de diferentes herramientas software que nos conduzcan a la resolución de todos los apartados. A continuación recopilaremos los diferentes programas utilizados.

- PuTTy: Es un cliente SSH, sirve para la conexión remota a servidores. En resumidas cuentas, PuTTy nos permite tener acceso al terminal de comandos de un servidor remoto.

- VNC Viewer: Con este programa podremos controlar remotamente la Raspberry Pi desde nuestro ordenador, teniendo acceso a todos los documentos, mostrándonos el escritorio, actuando como si conectásemos directamente la Raspberry a un monitor.
- Autpdesk Fusion 360: Programa de diseño CAD con el que se ha diseñado el cuerpo del vehículo en 3D.
- EasyEda. Programa web destinado al diseño del esquema de conexiones y diseño de la PCB
- Arduino IDE. Entorno de desarrollo que permite programar y compilar programas para el microcontrolador Arduino.
- Matlab. Programa de cómputo numérico con lenguaje de programación propio. Su uso en este trabajo ha sido para el cálculo del vector de realimentación del control LQR y para la representación gráfica de los resultados obtenidos ante los experimentos desarrollados.

2 Componentes

En el presente capítulo mostraremos los diferentes componentes que forman nuestro vehículo. Haremos una clasificación de los mismos, entrando en detalle sobre su funcionamiento y sus características técnicas consideradas más importantes.

Indagaremos con mayor detalle sobre la Raspberry Pi, ya que es el componente más novedoso que hemos añadido con respecto a las versiones anteriores del proyecto, y por ello el más desconocido.

2.1 Raspberry Pi

La Raspberry Pi, en adelante RPi, es un pequeño computador del tamaño aproximado de una tarjeta de crédito, cuya idea surgió en la universidad de Cambridge (Reino Unido) en el año 2006 de la mano del físico e ingeniero Eben Upton. El principal objetivo era el fomento de la enseñanza de las ciencias de la computación entre los más jóvenes, vista las cadencias con las que llegaban los alumnos a la universidad. Así, se proporcionaba un dispositivo barato con el que poder experimentar libremente sobre programación.

Con la incorporación de 26 pines GPIO en los primeros modelos, y posteriormente 40, hace que cada día tenga mayor aceptación por parte del público en general para desarrollar proyectos robóticos, ya que podemos conectar dispositivos como motores, sensores, controladores, cámaras…

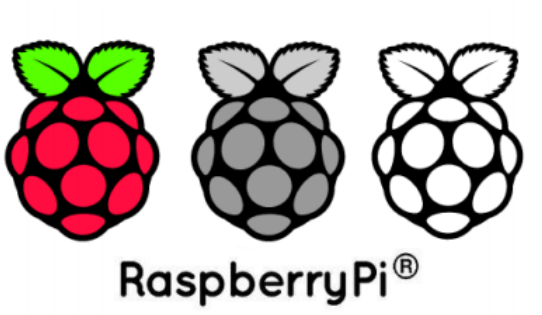


Figura 2.1 Logo de la fundación Raspberry Pi.

2.1.1 Historia

Como hemos indicado, la idea de fabricar este pequeño computador surgió en 2006, aunque no fue hasta febrero del 2012 después de numerosas pruebas y prototipos, cuando salieron al mercado las primeras unidades de la primera generación en dos modelos claramente diferenciados, el modelo A y el modelo B.

El modelo A presentaba 26 pines GPIO, salida de video HDMI, RCA y DSI, un conector Jack de 3.5 milímetros, un puerto USB y otro micro USB destinado a la alimentación, con un procesador Broadcom BCM2835 Single-Core a 700MHz y 256 MB de RAM. Mientras que el modelo B presentaba las mismas características en cuanto al procesador y la parte gráfica, mejorando la memoria RAM a 512 MB, aumentando a 2 los puertos USB, añadiendo un conector Ethernet. Poco después se lanzó el modelo B+ y A+ presentando mejoras tales como el aumento del número de puertos USB, mejorando el consumo de potencia y sustituyendo el lector SDHC por un micro SDHC.

Tabla 2.1 Características primeros modelos RPi.

Modelo	**A**	**A+**	**B**	**B+**
SoC	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2835
CPU	ARM 1176JZF-S A 700 MHz	ARM 1176JZF-S A 700 MHz	ARM 1176JZF-S A 700 MHz	ARM 1176JZF-S A 700 MHz
GPU	Broadcom VideoCore IV	Broadcom VideoCore IV	Broadcom VideoCore IV	Broadcom VideoCore IV
RAM	256 Mb	256 Mb	512 Mb	512 Mb
USB	1	1	2	4
Almacenamiento	SD	MicroSD	SD	MicroSD
Tamaño	85.6 X 56.5 mm	85.6 X 56.5 mm	85.6 X 56.5 mm	85.6 X 56.5 mm

Debido a la gran acogida inicial, superando con creces las expectativas de sus creadores, continuaron las mejoras, lanzando tan solo dos años más tarde, en 2014, la segunda generación. La Raspberry Pi 2 modelo B, mejoró sustancialmente el motor de cálculo acoplando un procesador Broadcom BCM2836 pasando de 1 a 4 el número de núcleos, aumentando la velocidad de procesamiento a 900MHz, duplicando la memoria RAM hasta llegar a un 1 GB y aumentando el número de pines GPIO de 26 a 40.

En el año 2015 salió al mercado la Raspberry Pi Zero, a grandes rasgos similar a la Raspberry Pi 1 modelo B, con la diferencia de que su tamaño se redujo notablemente, debido a ello se tuvo que prescindir del puerto Ethernet y del conector DSI. Posee el procesador Broadcom BCM2858 con 512 MB de SDRAM y un procesador a 1 Ghz. Poco después se empezó a comercializar la Raspberry Pi ZERO W, similar a la mencionada anteriormente añadiéndole un módulo Bluetooth 4.1 y Wifi.

En el año 2016 salió al mercado la tercera generación. La Raspberry Pi 3 modelo B cuyas diferencias con el modelo anterior aparentemente no eras sustanciales, renovando únicamente el procesador pasando de una velocidad de procesamiento de 900MHz a 1200MHz, pero con ello se aumentó el rendimiento un 30 %. Se añadió un módulo WIFI y un módulo Bluetooth 4.1, mejorando así la conectividad del terminal.

Dos años más tarde, en 2018 salieron al mercado los modelos B+ y A+, en el primero de ellos se mejoró el procesador hasta llegar a los 1.4 GHz, mejorando la conectividad inalámbrica y aumentando la velocidad del puerto Ethernet. En el modelo A+ se dio un paso atrás en cuanto a las prestaciones, a costa de reducir notablemente su precio, el cual bajo a casi la mitad, la memoria RAM disminuyó a 512 MB, se redujo el número de puertos USB a 1 y se elimina el puerto Ethernet.

En 2019 salió al mercado la cuarta generación, la Raspberry Pi 4 que cuenta con un procesador Broadcom BCM2711 de 4 núcleos a 1.5 GHz, una memoria RAM de 1, 2 o 4 GB dependiendo del modelo, eliminado el puerto HDMI sustituido por dos puertos micro HDMI dando la posibilidad de conectar dos monitores, continúa contando con cuatro conectores USB mejorando la velocidad de dos de ellos a 3.0.

Tabla 2.2 Características RPi 3 generación.

Modelo	3 A+	3 B+	3 B
SoC	Broadcom BCM2837B0	Broadcom BCM2837B0	Broadcom BCM2837
CPU	1.4GHz QUAD ARM Cortex-A53	1.4GHz QUAD ARM Cortex-A53	1.2GHz QUAD ARM Cortex-A53
GPU	Broadcom VideoCore IV	Broadcom VideoCore IV	Broadcom VideoCore IV
RAM	512 Mb	1 Gb	1 Gb
USB	1	4	4
Almacenamiento	SD	MicroSD	MicroSD
Tamaño	85.6 X 56.5 mm	85.6 X 56.5 mm	85.6 X 56.5 mm

2.1.2 Especificaciones Raspberry Pi modelo 3

Para el presente trabajo nos hemos decantado por utilizar el modelo B de la Raspberry Pi 3 que, como hemos visto antes, presenta las siguientes características:

Tabla 2.3 Características RPi 3 modelo B.

Procesador	Broadcom BCM2837
CPU	1.2GHz QUAD ARM Cortex-A53
GPU	Broadcom VideoCore IV a 400MHz
RED	Ethernet, wifi y bluetooth 4.1
Alimentación	2.5-3A/5V a través de microUSB
Almacenamiento	MicroSD
Pines	40 GPIO en dos hileras de 20
RAM	1GB SRAM
Puertos USB	4 a 2.0
V/A	Jack y HDMI
Tamaño	85x56mm

2.1.3 Sistema operativo

Como ya sabemos, la RPi es un microcomputador y por ello será necesario la instalación de un sistema operativo para poder operar con ella.

Existen multitud de sistemas operativos que podemos instalar a nuestra RPi. La fundación Raspberry Pi proporciona oficialmente el sistema operativo Raspbian, basado en Debian, un popular sistema operativo Linux que se encuentra optimizado para sacar el mayor rendimiento a nuestro dispositivo y es por el que nos hemos decantado en este proyecto. Podemos destacar su sencillez, estabilidad y versatilidad. Dispone de más de 40000 paquetes precompilados para su instalación.

En la web de la fundación Raspberry Pi podemos encontrar diferentes sistemas operativos que podemos instalar fácilmente en nuestra RPi a través de la tarjeta micro SD. Entre los sistemas operativos que podemos encontrar, citaremos los siguientes: LibreELEC, Raspbian Lite, Lakka, Data Partition, OSMC, Recalbox o Windows 10 Iot.

Como hemos indicado, nos hemos decantado por la instalación del sistema operativo Raspbian, sin embargo dicho sistema al ser una distribución linux no esta optimizado para aplicaciones en tiempo real, lo cual puede llegar a ser un grave impedimento al disminuir las prestaciones de la RPi. Para solventar dicho problema, se ha instalado un parche (Xenomai) que convierte nuestro sistema operativo en un sistema en tiempo real.

Podemos definir un sistema en tiempo real como aquel en el que el tiempo en que se produce la salida es significativo. Esto es debido a que la entrada corresponde a algún movimiento en el mundo

físico y la salida esta relacionada con dicho movimiento. En los sistemas en tiempo real no solo es importante el resultado de la computación, sino también el tiempo que tarda en producirse dicho resultado.

Xenomai

Xenomai es un proyecto de software libre para construir un marco versátil en tiempo real para sistemas operativos basados en Linux. Xenomai pretende que varias API de sistemas operativos en tiempo real se encuentren disponibles en Linux, en especial cuando el núcleo de Linux no puede cumplir con los requisitos de tiempo de respuesta.

Xenomai está basado en un enfoque de doble núcleo o dual kernel. Esta configuración consiste en situar un pequeño núcleo de tiempo real debajo del sistema operativo. Este pequeño núcleo programa las tareas en tiempo real situadas en su entorno y ejecuta el sistema operativo sobre el que se asienta como una tarea de prioridad mínima. Gracias a ello, las tareas que se ejecute en este núcleo adicional, dispondrán de alta prioridad consiguiendo con ello un alto rendimiento y predictibilidad.

2.1.4 Configuración inicial

Una vez instalado el sistema operativo, tenemos que proceder a su configuración inicial. A simple vista lo que tenemos es un ordenador normal con su escritorio y sus diferentes programas y archivos.

Para la configuración inicial de la RPi, necesitaremos además de lo mencionado anteriormente, un teclado y un ratón que conectaremos vía USB y una pantalla que conectaremos mediante un cable HDMI para poder movernos por los diferentes menús. Posteriormente, podemos trabajar con la RPi directamente manteniendo conectado el teclado, el ratón y la pantalla, o podemos también trabajar remotamente en ella a través de nuestro ordenador, mediante conexión remota o SSH, lo cual ahorra costes al prescindir de los periféricos mencionados.

En mi caso y para mayor comodidad he preferido trabajar remotamente, por ello tenemos que habilitar la interfaz SSH. Simplemente accedemos al menú *preferencias>configuración de Raspberry PI>interfaces*, donde nos saldrá la siguiente pantalla:

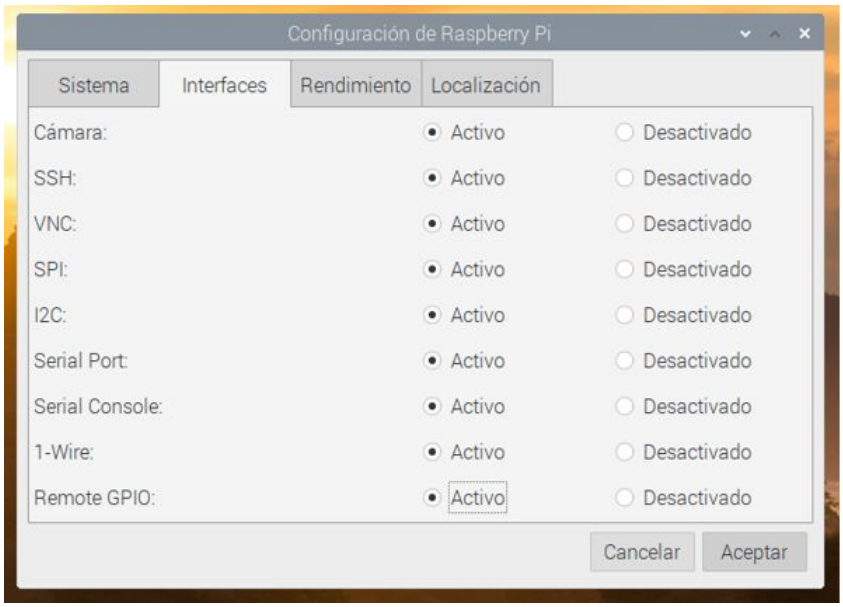


Figura 2.2 Pantalla de configuración de interfaces.

Por defecto se encuentran desactivadas todas las interfaces. Simplemente las activaremos y reiniciaremos la RPi para que los cambios se hagan efectivos.

Raspbian, como hemos indicado anteriormente, no es más que una distribución libre del sistema operativo GNU/Linux basada en Debian que se ha adaptado y optimizado para la RPi. Para mantener constantemente actualizado el software, tenemos que teclear y ejecutar los siguientes comandos en el terminal o consola de comandos de la RPi:

- *sudo apt-get update*
- *sudo apt-get upgrade*

El primero de ellos se encargará de comprobar si existen actualizaciones del software instalado, mientras que el segundo instala las actualizaciones disponibles. Es recomendable teclear ambos comandos con asiduidad para tener siempre actualizada nuestra RPi. Para aprovechar todo el tamaño de la tarjeta microSD, vamos a proceder a expandir el sistema operativo para que ocupe toda la capacidad de la tarjeta y así contar con mayor memoria. El paso anterior podemos hacerlo tecleando desde el terminal de comandos *sudo raspi-config* donde nos aparecerá la siguiente pantalla y elegir la primera opción *ExpandFilesSystem*.

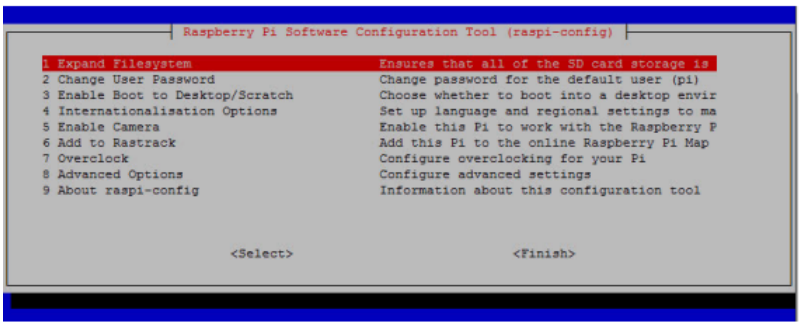


Figura 2.3 Pantalla de configuración RPi.

2.1.5 Conexión remota

Como hemos comentado, podemos trabajar con la RPi conectándole una pantalla mediante el puerto HDMI y un teclado y ratón mediante los puertos USB, ya que nuestro modelo consta de 4 puertos USB 2.0 o podemos conectarle un teclado y ratón inalámbricos mediante Bluetooth, de esta manera lo que tendríamos sería un ordenador convencional con su sistema operativo y sus periféricos de entrada/salida.

Sin embargo, la manera más cómoda para trabajar con Raspberry en aplicaciones robóticas, es mediante la conexión remota desde otro ordenador. Para ello, la RPi debe estar conectada a la misma red Wifi estableciendo así una conexión inalámbrica o mediante un cable Ethernet entre el computador y la RPi, está sería una conexión cableada más engorrosa y con mayor limitación.

Trabajar remotamente con la RPi proporciona una serie de ventajas y también de inconvenientes, que detallaremos a continuación.

Ventajas:

- Eliminamos la necesidad de tener que conectarnos con un teclado, ratón y pantalla, abaratando los costes al prescindir de estos dispositivos.
- Eliminación de cableado conectado a los puertos HDMI y USB, evitando tener excesivas conexiones en nuestra placa.

Inconvenientes:

- Necesidad de estar conectados a la misma red mediante otro ordenador, bien por Wifi o cable Ethernet.

Para realizar una conexión remota desde nuestro PC con Windows instalado, necesitamos descargar previamente una serie de programas en el pc:

- PuTTy: Es un cliente SSH, sirve para la conexión remota a servidores. En resumidas cuentas, PuTTy nos permite tener acceso al terminal de comandos de un servidor remoto.
- VNC Viewer: Con este programa podremos controlar remotamente la Raspberry Pi desde nuestro ordenador, teniendo acceso a todos los documentos, mostrándonos el escritorio, actuando como si conectásemos directamente la Raspberry a un monitor.

A continuación vamos a explicar cómo conectarnos remotamente mediante un cable Ethernet. Lo primero que tenemos que hacer es conectar el cable entre los puertos Ethernet del PC y de la RPi. Nuestro PC cuenta con el sistema operativo Windows 10, tenemos que seguir la ruta: *panel de control> Red e internet>Centro de redes y recursos compartidos> Cambiar configuración del adaptador*.

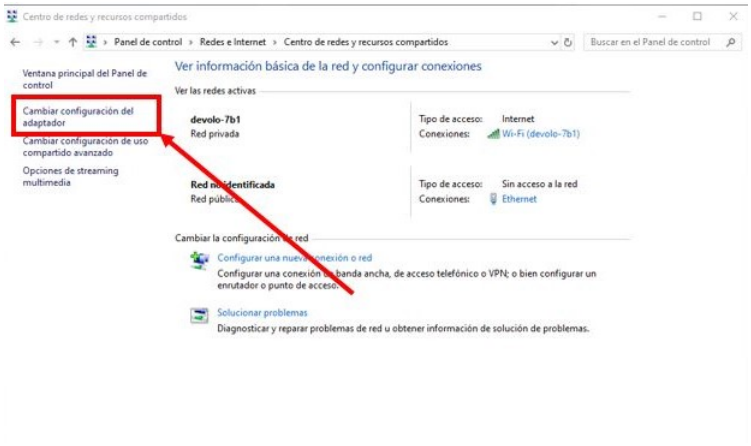


Figura 2.4 Pantalla de configuración Windows 10.

Hacemos click derecho en el icono Ethernet y accedemos a sus propiedades,

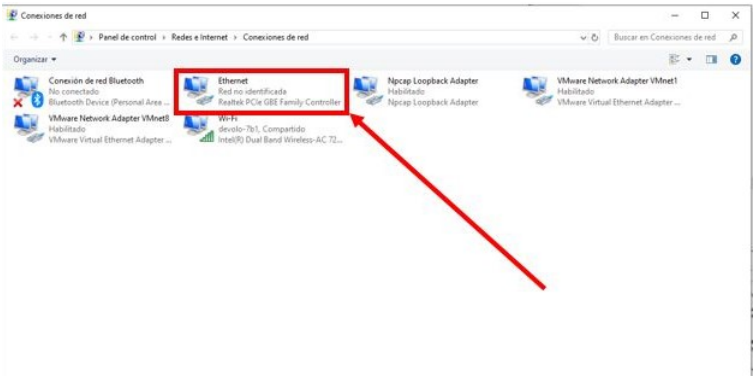


Figura 2.5 Pantalla de configuración Windows 10.

Cliqueamos en Protocolo de internet versión 4 (TCP/IPv4)

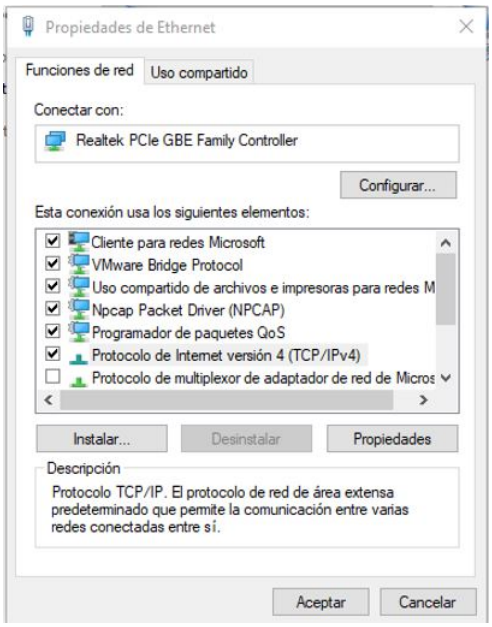


Figura 2.6 Propiedades de Ethernet.

Aparecerá la siguiente pantalla, donde pulsaremos *Usar la siguiente dirección IP* y le proporcionaremos una, en nuestro caso 192.168.148.1 y la máscara de red 255.255.255.0.

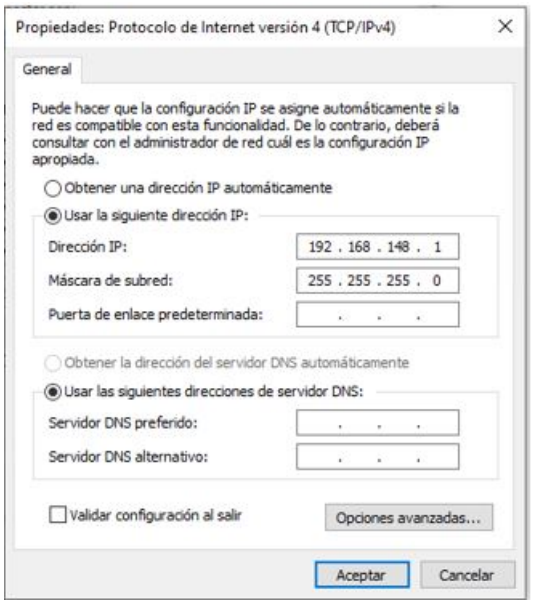


Figura 2.7 Propiedades Protocolo de Internet versión 4.

Realizada la configuración en el PC, tenemos que proceder a configurar la IP estática en la RPi. Para ello, en la tarjeta micro SD buscaremos el archivo *cmdline.txt*. Al final del texto del archivo tenemos que añadir la siguiente información:

Tabla 2.4 Configuración IP estática RPi.

IP cliente	192.168.148.2
IP servidor	
Puerta de enlace predeterminada	192.168.148.1
Mascara de Red	255.255.255.0
Nombre del cliente	rpi
Dispositivo al que nos conectamos	eth0
Configuración automática	off

Todos los datos anteriores deberán estar separado con dos puntos, quedando en nuestro caso la siguiente línea:

ip=192.168.148.2::192.168.1:255.255.255.0:rpi:eth0:off

Modificado el archivo en la tarjeta micro SD, volvemos a introducirla en la RPi y la conectamos por cable al PC.

Ahora vamos a comprobar que la instalación ha tenido éxito. Desde el terminal de comandos de Windows tecleamos el comando *ipconfig*, obteniendo la siguiente lista, donde vemos que el puerto Ethernet está conectado a la dirección IP definida estáticamente.

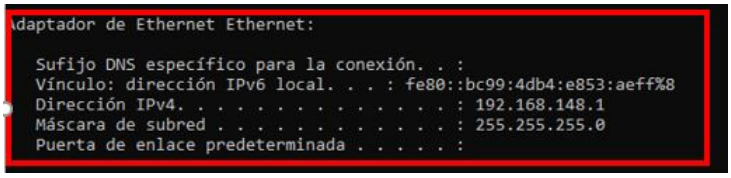


Figura 2.8 Terminal de comandos Windows 10.

Realizada la comprobación, podemos proceder a conectarnos a la Raspberry. Lo primero que tenemos que hacer es abrir el programa PuTTy e introducir la IP de la Raspberry que hemos definido estáticamente

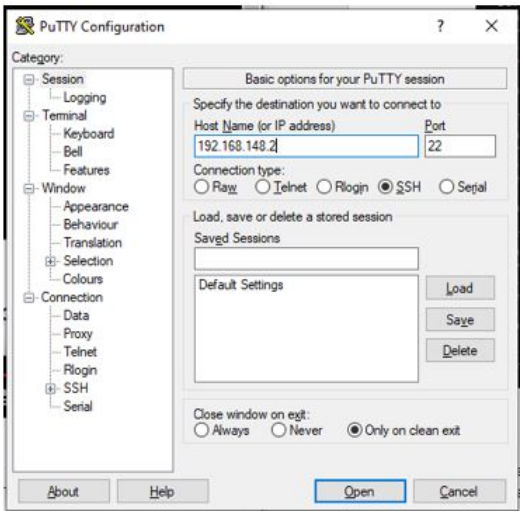


Figura 2.9 Servidor Putty.

Acto seguido se abrirá el terminal de comandos de la RPi, donde se nos pedirá el usuario, por defecto **pi** y la contraseña que le pusimos cuando instalamos el software.

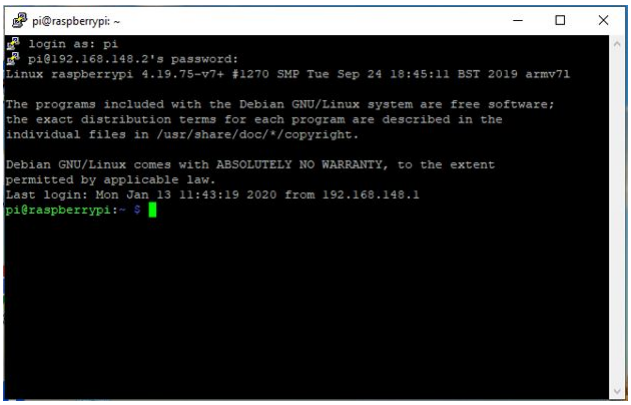


Figura 2.10 Terminal de comandos RPi.

2.1.6 Comunicación

Una de las grandes ventajas del uso de la Raspberry en robótica es la comunicación y control que podemos realizar sobre los circuitos y módulos electrónicos, que pueden ser de diversa manera:

- Mediante el uso de buses, como I2C o SPI que permiten la comunicación con dispositivos complejos como sensores.
- Mediante el uso de los pines GPIO que proporcionan versatilidad y su uso es similar a la ya más que conocida placa Arduino.
- Mediante el uso de los módulos USB (nuestro modelo cuenta con 4), al que le podemos conectar un monitor, un teclado, un ratón…
- Mediante el uso del módulo Wifi, Ethernet o Bluetooth, con los que podemos establecer contacto en tiempo real con la red.

2.1.7 I2C

Por su importancia en este proyecto, se va hacer especial mención al protocolo I2C. A través de dicho protocolo se va conectar la RPi con el microcontrolador Arduino y con el sensor MPU6050.

El protocolo I2C (Inter-Integrate Circuit) es un protocolo de transferencia síncrona de datos en serie maestro-esclavo, utilizado para comunicar datos entre dos dispositivos. Dicho protocolo apareció al principio de la década de los ochenta de la mano del fabricante Philips, destinado a periféricos de baja velocidad. Hoy en día sigue gozando de una gran popularidad y aceptación debido a su relativa simplicidad, así como a la capacidad de establecer un gran número de conexiones.

El funcionamiento para la comunicación es simple, se requieren dos líneas de señal. Por un lado, se encuentra la línea Serial Data o SDA y por otro la línea Serial Clock o SCL. La línea SDA se utiliza para la transmisión de datos bidireccional, es decir, la misma línea SDA se utiliza para enviar y recibir datos en paquetes de 8 bits y la línea SCL se utiliza para la sincronización de la transmisión de datos. Dado que se requiere de una señal de reloj para la sincronización, la transferencia será síncrona. En la figura 2.11 podemos apreciar un esquema del bus I2C.

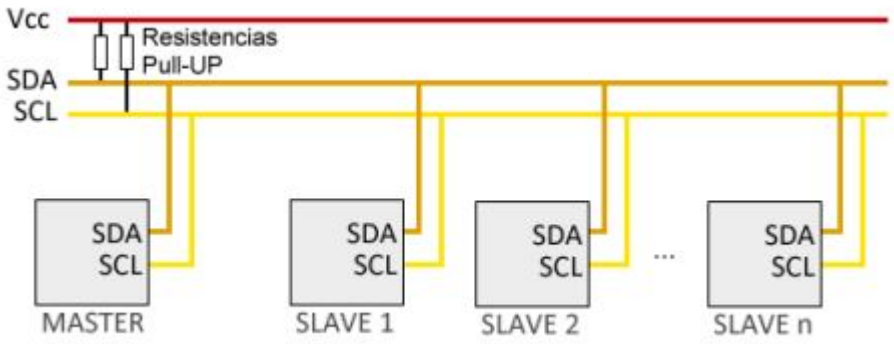


Figura 2.11 Esquema comunicación I2C.

Los dispositivos conectados al bus I2C disponen de una dirección única para cada dispositivo y pueden ser maestros o esclavos. Cada uno de los esclavos recibe de forma predeterminada una dirección en formato de 7 bits, por ejemplo en el caso que se verá más adelante, el módulo GY-521 que integra el sensor MPU-6050, tendrá asignada la dirección 0x68 o 0x69.

Como hemos indicado anteriormente, el proceso de transferencia de datos es del tipo maestro-esclavo. El maestro será el encargado de comenzar la comunicación con el resto de esclavos, pudiendo mandar o recibir datos de estos. Por otro lado, los dispositivos que se han definido como esclavos no pueden iniciar comunicación con el dispositivo maestro, aunque si pueden transferir datos al maestro.

En la RPi, el bus I2C admite direccionamiento de 7 y 10 bits y frecuencias como máximo hasta 400 kHz en modo rápido y 100 kHz en modo estándar.

Protocolo de transferencia

Para la transferencia de datos, el protocolo I2C seguirá los siguientes pasos:

1. El maestro será quien inicie la comunicación. Mientras la señal de reloj (SCL) se encuentre en alto, el maestro enviará un flanco descendente en la linea SDA para el comienzo de la comunicación. Se define así la secuencia de arranque, donde el bus pasa a considerarse ocupado.

2. Como hemos indicado, el bus envía paquetes de 8 bits. Primero enviará los 7 bits que corresponden a la dirección del dispositivo esclavo junto con el bit que indica si queremos realizar una operación de lectura o escritura. Los bits en la linea SDA estarán colocados de manera que comenzará enviándose el bit más significativo.

3. Si existe un dispositivo esclavo ocupando la dirección de los primeros 7 bits que envía el maestro, el esclavo contestará con un bit en bajo denominado bit de reconocimiento (ACK) que indicará que reconoce la solicitud y está listo para la comunicación.

4. Se envía el paquete de datos de 8 bits. En cada envío, la linea SCL produce un flanco de subida y posteriormente de bajada.

5. Una vez enviado el paquete de 8 bits, el dispositivo que recibe los datos envía de vuelta al un bit de reconocimiento que indica que se ha recibido los datos. Por lo tanto, en cada transferencia de 8 bits de datos se producen 9 pulsos en la señal de reloj(SCL), en concreto 8 pulsos para el envío del dato y 1 pulso para el envío del bit de recepción.

6. La trasmisión finalizará cuando el maestro envíe un flanco ascendente en la linea SDA, mientras la línea SCL se encuentre en alto. Se define así la secuencia de parada, el bus pasa a considerarse libre.

En el presente trabajo actuará como maestro la RPi, siendo la que inicia la comunicación con los dispositivos esclavos y el que lleve la iniciativa y genere la señal de reloj. Uno de los esclavos será el sensor MPU-6050 que ocupará la dirección 0x68 y del que únicamente se leerán datos. Por otro lado, tendremos el dispositivo esclavo Arduino al que hemos asignado la dirección 0x05. La interacción con este último esclavo será doble, pues el maestro le enviará dato y también leerá datos de él. Aunque se explicará más adelante, tenemos que recalcar que el maestro no puede comunicarse con ambos esclavos en el mismo instante, pues se produciría una interferencia y daría datos erróneos. Es decir, mientras se esta produciendo una transferencia de datos, no podemos iniciar otra sin que esta última haya finalizado.

Instalación en Raspberry

Por defecto, cuando instalamos el sistema operativo Raspbian no se encuentra habilitada la comunicación I2C, por ello tendremos que seguir una serie de pasos para habilitar dicha comunicación.

Lo primero que tenemos que hacer es entrar en el menú de configuración de la RPi y activar la interfaz I2C como vemos en la figura 2.2.

Raspbian, al ser una distribución de Linux, nos proporciona una serie de herramientas que facilitan la comunicación con los dispositivos del bus I2C. Para instalar dichas herramientas tenemos que teclear desde el terminal de comandos la siguiente línea:

sudo apt-get install i2c-tools.

Entre las herramientas que nos proporcionan, destacamos *i2cdetect*, que tecleareremos en el terminal de comandos como *sudo i2cdetect –y 1*. Dicho comando se encarga de comprobar la dirección del dispositivo que se encuentra conectado a nuestro terminal por el bus I2C, si no se encuentra conectado ningún dispositivo a la dirección en cuestión, aparcera el símbolo "–" que significará que se ha sondeado dicha dirección pero no se ha obtenido respuesta de ningún dispositivo. En el sondeo de direcciones puede aparecer UU en el lugar de la dirección que debería ocupar un dispositivo, esto indica que se ha obviado el sondeo de la dirección, debido a que dicha dirección está siendo utilizada por un controlador.

2.1.8 GPIO

Los pines GPIO (General Purpose Input/Output) permiten conectar dispositivos externos a nuestra Raspberry y pueden funcionar como de entrada o salida. Cuenta con 40 pines en dos hileras de 20, y la distribución es la que se muestra en la figura 2.12:

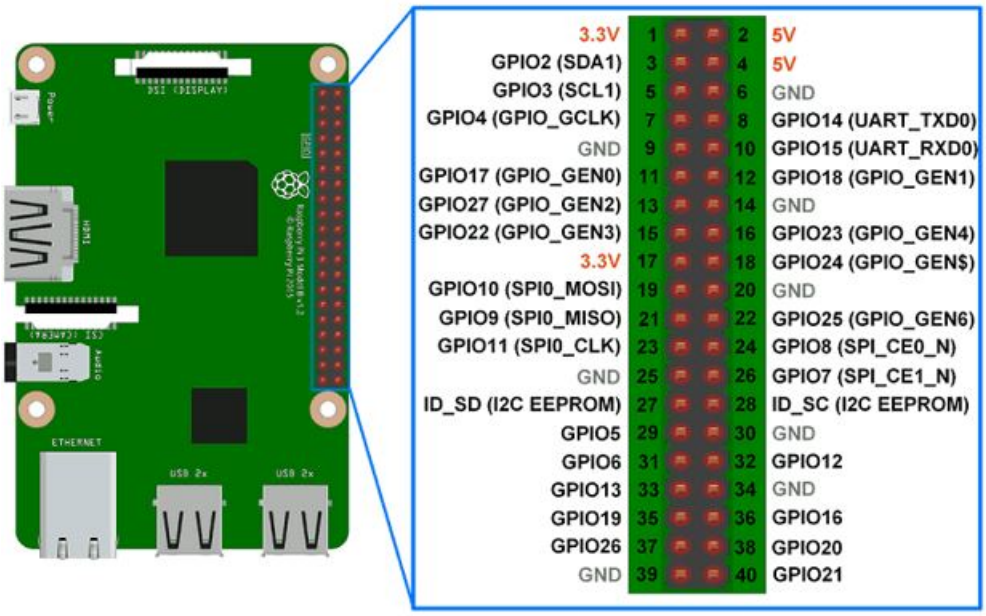


Figura 2.12 Distribución GPIO.

- La mayoría de los pines, un total de 26, están configurados como de entrada/salida de propósito general.
- Los pines 8 y 10 se pueden configurar como interfaz UART.
- Los pines 3 y 5 se utilizan para la interfaz I2C, permitiendo interactuar con dispositivos que se rigen según este protocolo.
- El pin 12, 13, 18 y 19 se configuran como salida PWM. Constará de dos canales PWM independientes que operan a la misma frecuencia, pudiendo alterar su ciclo de trabajo.
- Los pines 6, 9 ,14, 20, 25, 30, 34 y 39 se configuran como GND.
- Los pines 27 y 28 están configurados inicialmente como salida y se utilizan para conectar una memoria serie en las placas de expansión.
- Los pines 1 y 17 se utilizan para alimentar a 3.3 V, pueden suministrar juntos un máximo entorno a los 50 mA.
- Los pines 2 y 4 se utilizan para alimentar a 5V, pueden suministrar juntos un máximo entorno a los 200-300 mA.

Estos pines soportan una tensión de entrada comprendida entre 0-3.3V. Por lo tanto, tenemos que prestar sumo cuidado a la hora de conectar directamente a ellos dispositivos que trabajen a 5V, como el caso de Arduino. Sin embargo, la conexión de la RPi con Arduino mediante los pines I2C no presentan este inconveniente. Esto es debido a que los pines SCL y SDA están dotados de una resistencia pull-up.

2.2 Arduino Nano

Para el control de los motores se va hacer uso de la placa Arduino en su versión Nano.

Arduino Nano es una placa que contiene el microcontrolador Atmel ATmega328p. De características similares a Arduino Uno, cuenta con 3 Timers para la gestión de interrupciones, como se

explicará en el apartado de programación, se usará uno de ellos para generar el tren de pulsos que dote a los motores de movimiento.

A continuación se detallan las características consideradas más importantes. No se entra en gran detalle, para más información se puede acudir al foro oficial de Arduino donde se resuelven todo tipo de dudas.

Tabla 2.5 Características Arduino Nano.

Microcontrolador	Atmel ATmega328p
Frecuencia de operación	16 MHz
Voltaje de operación	5 V
Tensión de entrada	5-12 V
Entradas/Salidas Analógicas	8
Entradas/Salidas Digitales	14
PWM	6
SRAM	2kB
Flash	32kB
Puerto de programación	Mini USB
UART	1
EEPROM	1 kB
Dimensiones	45x18 mm

En el apartado anterior se indica que la RPi cuenta con dos canales PWM a la misma frecuencia. Al comienzo del proyecto se intentó controlar los motores paso a paso mediante ambos canales. Sin embargo, un requerimiento de nuestro proyecto es poder dotar a las ruedas de velocidades diferentes, es decir dotar a cada rueda de un tren de pulsos independientes a diferente frecuencia. Otra opción era generar la señal PWM por software pero este caso era lento y no se da garantía que los pulsos se produzcan cuando se desean. En esta explicación radica la necesidad de incorporar Arduino Nano a nuestro proyecto para el control de los motores paso a paso.

2.3 Motores paso a paso

Como actuadores del sistema se harán uso de dos motores paso a paso. En concreto, usaremos dos motores Nema 17.

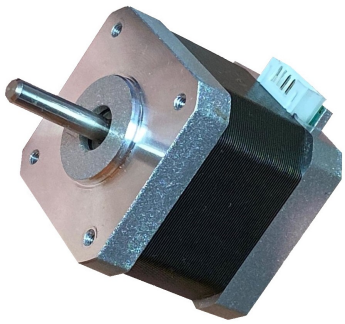


Figura 2.13 Motor paso a paso Nema 17.

El motor nema 17 es un dispositivo electrónico que se encarga de realizar movimientos precisos, potentes y controlados a partir de impulsos eléctricos. La velocidad de rotación será proporcional a la frecuencia con la que recibe los impulsos eléctricos. Recibe su nombre debido a las dimensiones de su base, 1.7 x 1.7 pulgadas.

Mejora las prestaciones de precisión y control en comparación con los motores de corriente continua o los servomotores.

Los motores paso a paso están compuestos por dos elementos, por un lado se encuentra una parte fija denominada estator compuestas por dos bobinas agrupadas y por otra parte se compone de unos imanes dentados fijados a un eje giratorio denominado rotor. Las bobinas del estator son imanes que cuando se excitan energéticamente, atraen a los ejes del rotor en sentido horario o anti horario.

Un motor paso a paso puede definirse como un motor de corriente continua en los que los desplazamientos angulares en el rotor son discretos. El funcionamiento del motor se produce en las dos bobinas que lo componen debido al cambio en la dirección del flujo de corriente que las atraviesa, es decir, al proporcionar energía en ambas bobinas de forma secuencial, se generan movimientos angulares en el rotor. El mínimo movimiento angular que se produce en el rotor se denomina paso, el movimiento de un paso se produce en cada pulso que se le envía al motor.

La elección de este motor es debido, entre otros aspectos, a su bajo coste, alta precisión sin errores acumulados, alta potencia , rápido posicionamiento, así como facilidad de control desde un microcontrolador a través del envió de trenes de pulsos.

Podemos encontrar dos tipos de motores paso a paso, los motores unipolares y los motores bipolares. En nuestro caso, usaremos este segundo tipo. La elección del motor bipolar se ha debido a que ese tipo de motores tiene mayor par motor que los unipolares, a costa de una mayor complejidad en el control. Este escollo es salvable con la adición del driver de control que se explica en el siguiente apartado.

Nuestro motor bipolar cuenta con cuatro cables, dos por cada bobina, por ello para energizarlas necesitamos del uso de un controlador o driver. Con dicho controlador, generaremos de forma sencilla los pulsos de señal para el funcionamiento del motor.

Para alimentar el motor, necesitaremos una fuente de alimentación externa que proporcione 12 V, ya que la RPi no es capaz de proporcionarnos dicha tensión por sus pines, como máximo proporciona 5V.

2.3.1 Características específicas

- Cuenta con 200 pasos completos por vuelta lo que equivale a 1.8 grados cada paso ($\frac{360^o}{200 pasos} = 1.8^o/paso$). Cada paso puede dividirse en hasta 8 micropasos, con lo que tendríamos 1600 micropasos por vuelta, aumentando con ello la precisión y la suavidad del giro. Para poder producir esta división del paso, necesitaremos el uso de un microstepping motor drive, tal y como veremos en el apartado siguiente.

  A continuación podemos apreciar los pasos necesarios para barrer una vuelta de 360 grados.

Tabla 2.6 División de pasos.

	Micropasos	**Pulsos por vuelta**	**Ángulo de avance**
Paso	1	200	1.8º
Medio paso	2	400	0.9º
Cuarto de paso	4	800	0.45º
Octavo de paso	8	1600	0.225º

- Presenta un eje circular de 5 mm de diámetro y 21 mm de largo.

- Presenta una base de 42x42 mm y una altura de 34 mm con un peso que ronda los 280 g.
- Cuenta con dos cables por cada bobina que se conectará con el controlador como indicaremos más adelante.
- Corriente nominal de 1.5 A DC.
- Tensión de alimentación de 12-36 V.
- Resistencia en fase de 8 ohm (+-10 %).
- Fase de inductancia de 10 mH (+-20 %).
- Inercia del rotor de 54 $G.cm^2$.
- Par de frenado: 1.6 $N.cm$.
- Par de fricción: 12 $mN.m$.
- Máxima frecuencia de arranque sin carga >= 1500pps.
- Máxima frecuencia de ejecución sin carga >=8000pps.

2.4 Driver de control Easydriver HW135 v4.4

Para el control de los motores paso a paso necesitaremos indispensablemente dos controladores, uno para cada motor. Hemos elegido el Easydriver HW135 en su versión 4.4. Consiste en un circuito integrado con un chip A3967SLB de Allegro. En el mercado encontramos diferentes tipos de driver, la elección de este en concreto se debe a su utilización en proyectos anteriores del departamento donde se ha comprobado su alta fiabilidad, gozando de una gran popularidad debido, entre otros aspectos, a su bajo coste y tamaño reducido, ideal para sistemas embebidos.

Como hemos indicado, el Easydriver HW135 v4.4 es un controlador para un motor paso a paso bipolar con consumos entre 150mA y 750mA por fase y una salida entre 7 y 30V. Capaz de proporcionar modos de paso completo, medio, cuarto u octavo de paso.

El dispositivo cuenta con un potenciómetro que permite ajustar la corriente máxima que se suministra al motor por fase. Como se ha indicado, este ajuste se encuentra comprendido entre 150mA y 750 mA. Nuestro motor tiene un consumo de 1.5 A, por lo tanto tendremos que ajustar dicho pontenciómetro a su valor máximo.

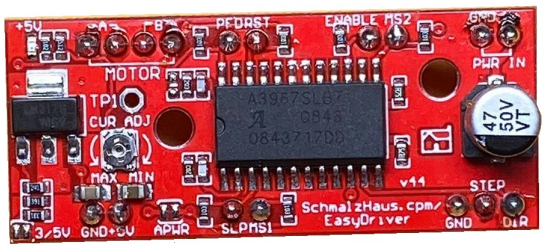


Figura 2.14 Easydriver HW135 v4.4.

Entre las funciones del driver de control podemos destacar:

- Proporcionar, a través de una fuente de alimentación externa, la energía necesaria para el funcionamiento del motor que será de 12 V. Tanto la RPi como Arduino poseen como máximo salidas de 5 V, la cual no harían funcionar los motores.

- Proporciona flexibilidad para cambiar el sentido de giro del motor.
- Posee la capacidad para regular o limitar la velocidad o el par.
- Proporciona protección ante sobrecargas que puedan producirse en el motor al limitar la corriente que circula por él.

2.4.1 Descripción de los pines

Pasaremos a describir brevemente el funcionamiento de los diferentes pines de entrada y salida que presenta el controlador. Indicar que los pines pueden soporta una tensión comprendida entre 0 y 5V o 0 y 3.3V, por defecto estará configurado a 0-5V pero cuenta con un puente de soldadura que permite al usuario cambiar esta configuración. Como la versión Arduino que se esta usando proporciona una tensión de 5V, dejaremos la configuración por defecto.

- GND: Es el pin de tierra o masa, el dispositivo cuenta con 3 de ellos.
- M+: Este pin es con el que alimentamos el controlador, acepta una entrada de alimentación de 6 V a 30V y una corriente de 2 A. Nosotros alimentaremos con una batería externa LIPO de 3 celdas que posee una tensión de 11.1 V nominales, energía que será más que suficiente para el funcionamiento de los motores.
- A (dos pines) y B (dos pines): Estos pines son las conexiones del motor con el controlador, son salidas analógicas. Tenemos un polo positivo y otro negativo de la conexión para cada bobina del motor, proporcionando la energía necesaria para su funcionamiento. Se conectará con el motor Nema 17 de la siguiente forma:

Tabla 2.7 Conexión motor.

Motor paso a paso	Easydriver HW135
cable azul	Pin B-
cable verde	Pin A-
cable Amarillo	Pin B+
cable Rojo	Pin A+

- STEP (PASO): Este pin es el encargado de producir el movimiento de los motores cuando se le aplica un tren de pulsos. Cuando se produce un flanco de subida, es decir un paso del nivel bajo al nivel alto, se activa el motor para que avance un paso o micropaso según hallamos configurado los pines MS1 y MS2. El movimiento se realizará en el sentido que proporciona el pin DIR. Es una señal digital de 0 a 5 V.
- DIR (DIRECCIÓN): Este pin activado a nivel alto o bajo determina la dirección de rotación del eje del motor a un lado o hacia el otro. Es una señal digital de 0 a 5 V.
- MS1 / MS2: Pin de entrada digital para controlar la resolución, es decir, el modo de micropaso que puede ser paso completo, medio paso, cuarto de paso u octavo de paso dependiendo si se encuentran en valor alto o bajo.
- RTS (RESET): pin de reseteo o inicio, desactiva las salidas cuando está a nivel BAJO, es decir, las señales que se envíen por el pin STEP no se tendrán en cuenta. Para habilitar el control de los pasos debe estar a nivel ALTO, es decir, desactivado. Por defecto, se encuentra en este último estado. Irrelevante en nuestro proyecto.
- SLP: modo suspensión, minimiza el consumo de energía. Irrelevante en nuestro proyecto

Tabla 2.8 Resolución del paso.

MS1	MS2	Resolución del paso
BAJO	BAJO	Paso completo
ALTO	BAJO	Medio paso
BAJO	ALTO	Cuarto de paso
ALTO	ALTO	Octavo de paso

- +5V: Es un pin de salida que proporciona 5 V que se pude utilizar para alimentar otro dispositivo. Irrelevante en nuestro proyecto.
- ENABLE: Pin de entrada lógica que permite el funcionamiento de los transistores de salida que maneja el motor. En ALTO desactiva los transistores, mientras que, en BAJO, habilita los transistores y permite controlar el motor. Irrelevante en nuestro proyecto.
- PFD: Voltaje de entrada que selecciona el modo de descenso de la corriente de salida. Para activar el modo de descenso de corriente lento, PFD debe ser mayo de VCC, por el contrario, PFD es menor de VCC se activará el modo de descenso rápido. Si PFD se encuentra entre ambos valores obtendremos un descenso intermedio de corriente. Irrelevante en nuestro proyecto
- SLP: Pin de entrada lógica que cuando está en BAJO, se encontraran desactivadas las salidas y el consumo será mínimo. Irrelevante en nuestro proyecto

2.4.2 Características

El controlador EasyDriver presenta las siguientes características:

- Tensión de alimentación de carga hasta 30V.
- Corriente de salida continua +- 750 mA.
- Corriente de salida de pico +- 850mA.
- Temperatura de operación -20 +85ºC.
- Rango de tensión de alimentación de la lógica 3.3 o 5 V.
- Frecuencia máxima en el pin STEP 500kHz.

2.5 MPU6050

Nuestro proyecto necesitará la adición de un dispositivo que le proporcione información del mundo exterior, en concreto, necesita que se le proporcione el ángulo de inclinación del vehículo con respecto a su vertical, así como su velocidad angular de caída del vehículo.

Para ello, se ha elegido el dispositivo GY-521 que incorpora el sensor MPU6050 del fabricante InvenSense. La motivación para la elección de este dispositivo radica en su bajo coste y alta popularidad, además este dispositivo se ha utilizado en proyectos anteriores similares donde se ha comprobado su alta fiabilidad.

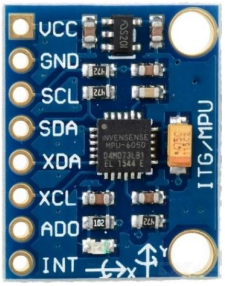


Figura 2.15 Controlador MPU.

Los sensores son elementos que nos proporcionan información sobre magnitudes físicas del entorno que lo rodea o de su estado interno, detecta estímulos y actúa en consecuencia. Detectan magnitudes físicas y las transformas en magnitudes eléctricas que serán interpretadas por el controlador. Usaremos un sensor interno que nos proporciona información de la posición y velocidad del dispositivo.

El sensor MPU-6050 es conocido como unidad de medición inercial o por sus siglas en inglés, IMU (Inertial Measurment Units). Consta de 6 grados de libertad, combinando un MENS (Micro-ElectroMechanical Systems) acelerómetro y un MENS giroscopio ambos de 3 ejes. Está dotado de 8 pines y opera a 3.3 voltios.

Con el giroscopio de 3 ejes medimos la velocidad angular en grados por segundo (dps), y con el acelerómetro podemos medir las componentes X, Y y Z de la aceleración en g.

En nuestro trabajo, al tratarse de un péndulo 2D, solamente usaremos tres grados de libertad, dos ejes del acelerómetro (ejes x e y) y uno del giroscopio.

Tenemos que tener que cuenta que tanto el giroscopio como el acelerómetro proporcionan datos brutos en enteros de 16 bits en forma de complemento a 2, lo que da un rango de lecturas entre -32768 y 32768. Para convertir estos datos en valores tangibles, tenemos que realizar una conversión sabiendo que el factor de escala del acelerómetro es 16384 LSB/g y del giroscopio de 131 LSB/º/s para los rangos por defecto:

- Acelerómetro -2g a +2g.
- Giroscopio -250/s a 250 º/s.

Los datos brutos obtenidos tendrán que ser divididos entre los factores indicados anteriormente.

El dispositivo estará conectado con la RPi que procesará los datos que recibe de él mediante el bus I2C.

2.5.1 Descripción del fabricante

La unidad de procesamiento de movimiento MPU-60X0 es la primera solución del mundo en procesamiento de movimientos, con la fusión integrada de sensores con 9-ejes, la fusión de sensores utiliza su motor propiedad MotionFusion™ probado en el campo de teléfonos móviles, tabletas, aplicaciones, controladores de juegos, mandos a distancia, puntero de movimiento y otros dispositivos de consumo. El MPU-60X0 tiene integrado un giroscopio MEMS de 3 ejes, un acelerómetro de 3 ejes MEMS, y un procesador digital de movimiento (DMP™) motor acelerador de hardware con un puerto I2C auxiliar que se conecta a las interfaces de sensores digitales de terceras partes tales como magnetómetros. Cuando se conecta a un magnetómetro de 3 ejes, el MPU-60X0 entrega una salida completa de 9 ejes MotionFusion para su primario I2C o puerto SPI (SPI está disponible sólo en MPU-6000)". El MPU-60X0 combina la aceleración y el movimiento de rotación más la información de rumbo en un único flujo

de datos para la aplicación. Esta integración de la tecnología MotionProcessing™ presenta un diseño más compacto y tiene ventajas de costos inherentes en comparación con soluciones discretas de giroscopio más acelerómetro. El MPU-60X0 también está diseñado para interactuar con múltiples sensores digitales no inerciales, tales como sensores de presión, en su bus I2C auxiliar maestro. El MPU-60X0 es un procesador de movimiento de segunda generación y es la huella compatible con la familia MPU-30X0.

2.5.2 Descripción de los pines

El MPU6050, presenta los siguientes pines, aunque serán de utilidad para nuestro proyecto únicamente los 4 primeros:

- Vcc: Pin de alimentación que dota a nuestro dispositivo de la tensión necesaria para su funcionamiento. La tensión con la que opera el dispositivo son 3.3V. Dispone de un regulador de tensión interno que limita la tensión que se le proporciona a 3.3V, lo que hace posible poder alimentarlo con el pin de 5V de la RPi.
- GND: Pin de tierra o masa.
- SDA: Pin de datos en serie para la comunicación I2C.
- SCL: Pin de entrada de reloj para la comunicación I2C.
- XDA: Esta es la línea de datos del sensor I2C SDA para configurar y leer desde sensores externos. No será de utilidad en nuestro proyecto.
- XCL: Esta es la línea de reloj del sensor I2C SCL para configurar y leer desde sensores externos. No será de utilidad en nuestro proyecto.
- ADO: Conectando este pin a una alimentación de 5V cambiará la dirección I2C. Por defecto ocupara la dirección 0x68, conectando este pin, cambiará la dirección a 0x69.No será de utilidad en nuestro proyecto.
- INT: Pin de interrupción para indicación de datos listos. No será de utilidad en nuestro proyecto.

2.5.3 Características del dispositivo

- Comunicación I2C con una velocidad máxima de 400kHz.
- Salida digital del giroscopio con rango programable (±250, ±500, ±1000 y ±2000 °/s).
- Salida digital del acelerómetro con rango programable (±2g, ±4g, ±8g y ±16g).
- 6 convertidores analógicos digitales de 16 bits para las salidas del giroscopio y acelerómetro.
- Buffer de 1024 bytes para la recogida de datos de forma conjunta en el caso de un modo de bajo consumo del sistema

2.5.4 Conexión con la Raspberry

La conexión del MPU-6050 con la Raspberry tiene lugar a través del protocolo de comunicación I2C. Los pines SCL y SDA están dotados de una resistencia pull-up para conectarlo directamente a nuestra placa sin que sufran daños los dispositivos.

El IMU dispone de dos direcciones I2C con las que podemos trabajar, una será la dirección 0x68 que será la proporcionada por defecto y la otra la dirección 0x69.

Tecleando desde el terminal de comando *sudo –y 1*, podemos comprobar que la Raspberry detecta correctamente el dispositivo en la dirección indicada:

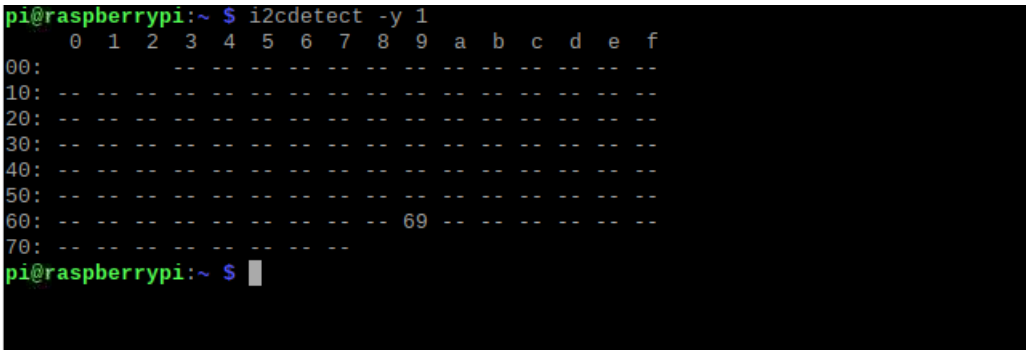


Figura 2.16 Comprobación dirección MPU en la RPi.

La conexión del módulo MPU-6050 sigue el siguiente esquema:

Tabla 2.9 Conexión MPU-RPi.

MPU	RPi
VCC	PIN 1 (5V)
GND	PIN 9 (GND)
SCL	PIN 5 (SCL)
SDA	PIN 3 (SDA)

Una vez realizada la conexión, tenemos que modificar el fichero ***/etc/modules***, para ello tecleamos desde el terminal de comandos *sudo nano /etc/modules* y añadidos las siguientes líneas

- I2c-bcm2708
- I2c-dev

Realizado este paso, nuestro dispositivo ya estará conectado a la RPi y listo para su funcionamiento.

2.5.5 Acelerómetro

Podemos definir la aceleración como una fuerza por unidad de masa.

El acelerómetro se encarga de medir la aceleración en las tres dimensiones del espacio, los ejes X, Y y Z. En el eje vertical, el dispositivo detecta la proyección de la fuerza de la gravedad (aceleración aproximada de 9.8 m/s^2). Gracias a dicho dato, el IMU es capaz de calcular el ángulo de inclinación con respecto a los otros dos ejes aplicando simples reglas trigonométricas.

Si colocamos el dispositivo alineando el eje Z con la perpendicular del suelo, se obtiene como medida en dicho eje el valor de la gravedad que se ha indicado anteriormente, obteniendo un valor cercano a cero en los otros dos ejes.

En la figura 2.17, a la izquierda se puede apreciar el dispositivo donde el eje Z se encuentra alineado con la perpendicular al suelo. En la imagen de la derecha, se puede apreciar el dispositivo girado 90 grados de forma que el eje Z se encuentra paralelo al suelo, adquiriendo el eje vertical, en este caso el eje X, el valor de la aceleración gravitatoria.

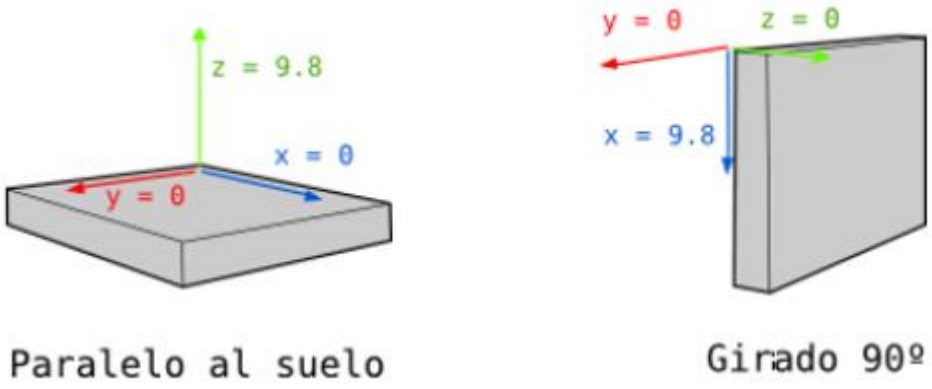


Figura 2.17 Disposición sensor MPU.

Aplicando las siguientes ecuaciones podemos calcular, mediante simples reglas trigonométricas el ángulo X e Y, siempre y cuando el eje sobre la vertical sea el Z.

$$AnguloX = atan\left(\frac{x}{\sqrt{y^2+z^2}}\right) \tag{2.1}$$

$$AnguloY = atan\left(\frac{y}{\sqrt{x^2+z^2}}\right) \tag{2.2}$$

Las medidas que obtenemos de las ecuaciones anteriores presentan un grave inconveniente como es el ruido y los grandes márgenes de error, lo que inhabilita su uso para la implementación en algoritmos de control. Para solucionar este problema necesitaremos hacer un filtrado de los datos obtenidos, para ello usaremos el filtro complementario como veremos más adelante. Las características principales del acelerómetro serán:

Tabla 2.10 Características Acelerómetro.

Características	Medidas
3 convertidores ADC	16 bits
Rango de salida programable	+-2,+-4,+-8 y +-16 g
Intensidad nominal	500μA

2.5.6 Giroscopio

El giroscopio se encarga de medir la velocidad angular o de rotación con respecto a un sistema de referencia estático. Podemos definir la velocidad angular como el número de grados que gira el dispositivo en un segundo, sus medidas en el S.I. serán rad/s (1 rad = 180/Π grados). Podemos conocer el ángulo en cada instante solamente sumándole al valor inicial del IMU, que será conocido, el valor del ángulo calculado por el giroscopio en un periodo determinado de tiempo. Como nota, indicar que la el dispositivo mide la velocidad angular en (º/s), por tanto tendremos que aplicar un factor de conversión para obtener los datos en el S.I.

La siguiente ecuación muestra la forma de obtener el ángulo mediante el uso del giroscopio.

$$Angulo = Angulo_{anterior} + (Giroscopio * \triangle t) \tag{2.3}$$

Por lo general, las medidas obtenidas por el giroscopio suelen ser más precisas que las calculadas por el acelerómetro, en un corto periodo de tiempo. Sin embargo, en un largo periodo de tiempo, al calcularse el dato como una suma de incrementos, hará que se aleje de la realidad debido a una suma de los errores acumulados en la medida.

Las características principales del giroscopio serán:

Tabla 2.11 Características Giroscopio.

Características	Medidas
3 convertidores ADC	16 bits
Rango de salida programable	+-250,+-500,+-1000 y +-2000 º/s
Intensidad nominal	3.6 mA

La finalidad de giroscopio en nuestro proyecto no solo radica en la posibilidad de calcular el ángulo de inclinación del vehículo. También nos proporciona la velocidad de caída del vehículo, que como se verá en el capítulo de control, será una variable de estado para el desarrollo de controladores lineales.

2.5.7 Filtro complementario

Como se ha indicado, no podemos obtener el ángulo de inclinación de nuestro vehículo directamente haciendo uso de las medidas del acelerómetro o del giroscopio. Dichos datos presentarán un gran ruido en el caso del acelerómetro y una gran deriva en el caso del giroscopio. Para salvar estas dificultades, vamos aplicar un filtrado de las medidas con el objetivo de obtener unos datos limpios, carentes de ruido y deriva que puedan ser interpretados por nuestro controlador.

Para el filtrado de los datos obtenidos directamente del IMU podemos contar con varias estrategias. Entre ellas se encuentran aplicar un filtro de Kalman o un filtro complementario. Por su sencillez vamos a hacer uso del filtro complementario.

El filtro complementario puede entenderse como un modelo simplificado del filtro de Kalman, eliminando el análisis estadístico lo cual reduce el número de cálculos y dota de mayor sencillez su implementación, igual de válida para el caso que se nos presenta.

La ecuación del filtro complementario adquiere la siguiente expresión, donde vemos que aparece el término τ, el cual pondera el peso de las medidas del acelerómetro y del giroscopio. Normalmente adquiere un valor cercano a 0.99 dando una mayor importancia a la medida del giroscopio, que como hemos indicado son más precisas. La medida del acelerómetro será la encarga de corregir la deriva que va produciendo el giroscopio.

$$Angulo = \tau(Angulo_{previo} + Angulo_{giroscopio}) + (1-\tau)Angulo_{acelerometro} \tag{2.4}$$

2.6 Batería

Para dotar al vehículo de la autonomía necesaria se prescindirá de toda alimentación por cable conectados a la corriente. Nos centraremos en el uso de una batería que alimente a todos los dispositivos involucrados en el funcionamiento del sistema.

La batería elegida será de polímero de litio, popularmente conocida como batería Li-Po. En el mercado podemos encontrar baterías Li-Po con diferentes especificaciones. Se ha optado por el uso de la batería X-COPTER POWER que consta de 3 celdas. Nos proporciona una capacidad de 1.500 mAh y una tensión nominal de 11.1 V.

Figura 2.18 Batería LIPO.

En el esquema del sistema electrónico, podemos apreciar más detalladamente la forma de alimentar cada dispositivo dependiente de la batería, aquí se citará brevemente.

Arduino y la RPi se alimentarán a una tensión constante de 5V, tensión que se obtendrá a partir del regulador de tensión explicado en el apartado siguiente. Ambos motores se alimentarán a través de los controladores Easydriver con la tensión que proporciona directamente la batería. El resto de componentes se alimentarán a partir de la tensión que proporciona tanto RPi como Arduino en las salida de sus pines.

2.7 Regulador de tensión KA78T05

En el apartado anterior se indica el uso de una batería LIPO para la alimentación de la RPi, de Arduino y de ambos motores. La alimentación de ambos motores será a 12 V, no teniendo problemas para alimentarlo directamente desde la batería a través del controlador EasyDriver. Sin embargo, la Rpi no tolera valores de 12 V por su pin de alimentación, siendo la tensión máxima soportada de 5V. Arduino se podría alimentar por el pin Vin que tolera voltajes comprendidos entre 6-12 V o por el pin de 5V. Se opta por esta segunda opción.

Para solventar el problema anterior con respecto al voltaje de alimentación de la Rpi, se va hacer uso del regulador de tensión lineal KA78T05.

Figura 2.19 Regulador de tensión KA78T05.

El regulador anterior nos permite obtener una tensión de salida continua de 5V y 3A a partir de

la entrada de 12 V. Dicha salida será estable y suficiente para la alimentación de la Rpi y Arduino. Ambas placas se alimentarán por su pin correspondiente de 5V. En el esquema eléctrico podemos apreciar las conexiones.

Si analizamos con detalle las conexiones del esquema eléctrico, se puede apreciar la incorporación de dos condensadores. Uno de 0.33μF a la entrada del regulador y otro de 0.1μF a la salida. Ambos condensadores se emplean, según detalla el Datasheet del regulador, para evitar picos de tensión que dañe nuestros dispositivos.

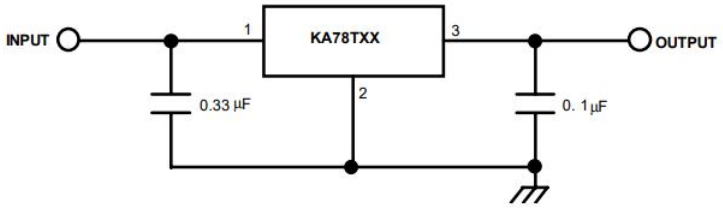


Figura 2.20 Esquema de conexión del Regulador de Tensión.

Ha sido necesario la adición de un disipador de calor debido a las altas temperatura que alcanza el regulador. Sin dicho disipador, el regulador corre riesgo de quemar la PCB. A parte del disipador, se ha incorporado un ventilador axial que ayuda a disipar el calor que genera tanto el regulador como la RPi y el resto de componentes en su funcionamiento.

2.8 Chasis

El chasis del vehículo es la estructura que engloba en su interior todos los componentes electrónicos.

Para su diseño, hemos hecho uso del programa de diseño CAD 3D Autodesk Fusion 360 y el formato utilizado ha sido STL, el cual nos permite obtener el vehículo mediante impresión 3D. El material utilizado para la impresión ha sido PLA (ácido poliláctico).

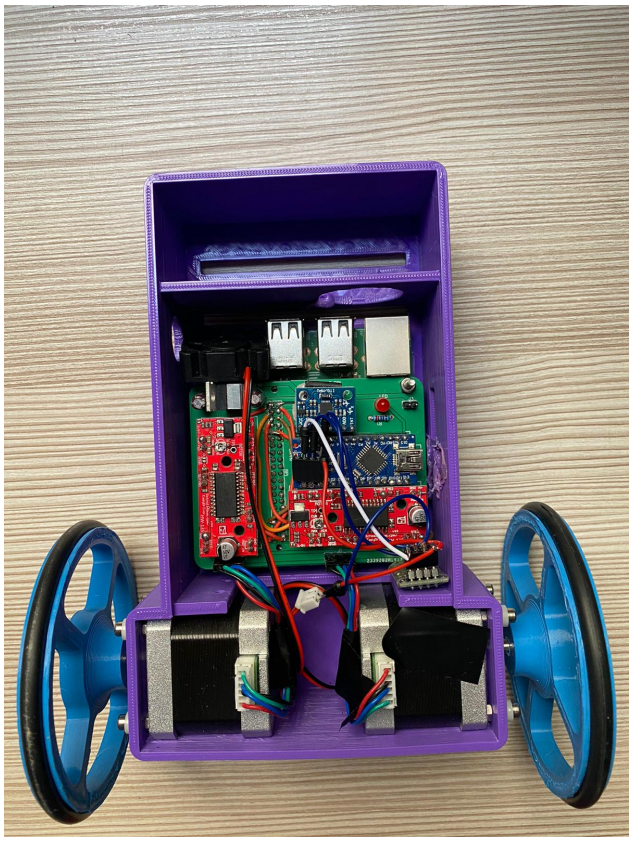
El vehículo consta de una estructura lisa, que será la base, dividida en 3 compartimentos conectados donde alojaremos la electrónica, la batería y los motores.

La parte exterior de la base se ha diseñado con paredes de 3 mm de grosor, en su parte trasera se ha dotado de respiradores con los que se pretende evacuar el calor que generan los circuitos electrónicos.

La distribución de los compartimentos será la siguiente:

- El compartimento superior está destinado a alojar la batería.
- El compartimento medio se ha destinado a alojar los componentes electrónicos. Se ha perforado la parte trasera para atornillar la RPi. En uno de sus laterales se han realizado una serie de perforaciones para tener acceso a la alimentación por cable y al puerto HDMI de la RPi, así como al puerto microUSB de Arduino Nano.
- El compartimento inferior alojará los motores atornillados a su pared lateral. Se ha realizado un orificio que permite la salida del eje del motor para poder acoplarle las ruedas.

Las ruedas utilizadas también se han impreso en 3D. Su diseño no se ha realizado en el presente trabajo, ya se disponía de él y ha sido proporcionado por el tutor del mismo. Para dotar de adherencia a la rueda, se ha recubierto de una junta tórica.



(a)

(b)

Figura 2.21 (a) Vista frontal del vehículo. (b) Vista trasera del vehículo. .

2.9 PCB

Para las conexiones de los diferentes componentes electrónicos, se ha optado por el diseño y fabricación de una placa de circuito impreso de tipo shield.

La PCB (Printed Circuit Board) se ha diseñado como escudo para la RPi. En ella agrupamos todos los componente prescindiendo así del uso de cables que puedan darnos problemas en un futuro, consiguiendo así mejores contactos y más fiables. Las conexiones se realizan por caminos o pistas de cobre. El grosor de las pistas dependerá de la corriente que circule por ellas, se ha establecido en 1 mm excepto las pistas de alimentación que, debido a la intensidad que circula por ellas, se ha diseñado de 1.34 mm.

Para la elección del grosor anterior hacemos uso de la siguiente gráfica donde se muestra el aumento de temperatura que se produce en una pista a partir de la corriente que circula por ella y de la anchura de la misma, todo ello para placas con un recubrimiento estándar de 35 micras. Se puede apreciar que para una intensidad de 3.1 A, el incremento mínimo de temperatura que soporta la placa ($10°C$). exige que la pista tenga un grosor de 1.35 mm.

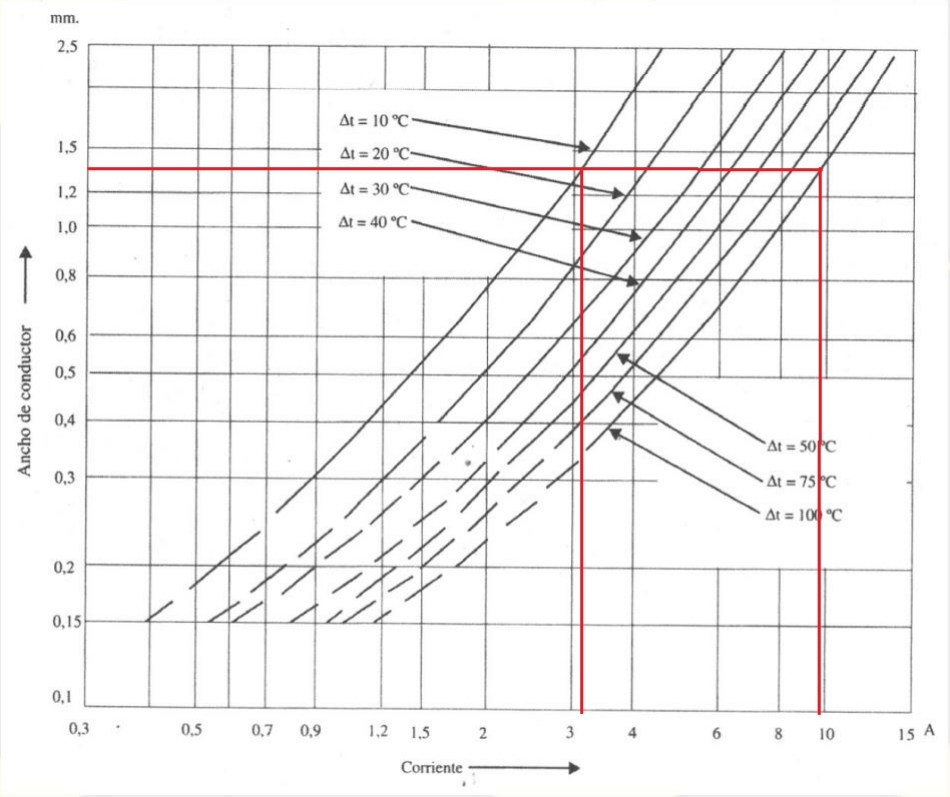


Figura 2.22 Elección del ancho de pista.

Para el diseño se ha hecho uso del programa web EasyEda que nos permite crear un esquema de conexiones para posteriormente crear la placa.

En la sección conexiones se muestra el esquema que compone la placa. Como hemos indicado, se ha diseñado como escudo para la RPi, sobre ella irán los siguientes elementos:

- Arduino nano.
- 2 controladores de motores paso a paso EasyDriver.
- MPU-6050.

- LED y resistencia que se encenderá cuando Arduino reciba alimentación.
- Convertidor de voltaje lineal KA78T05 y 2 condensadores.

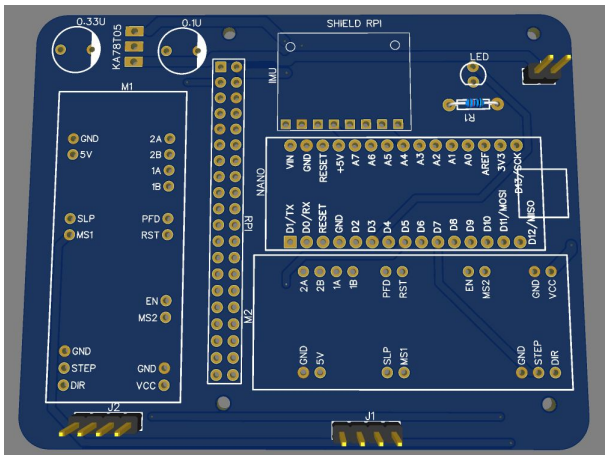


Figura 2.23 Vista 3D de la PCB.

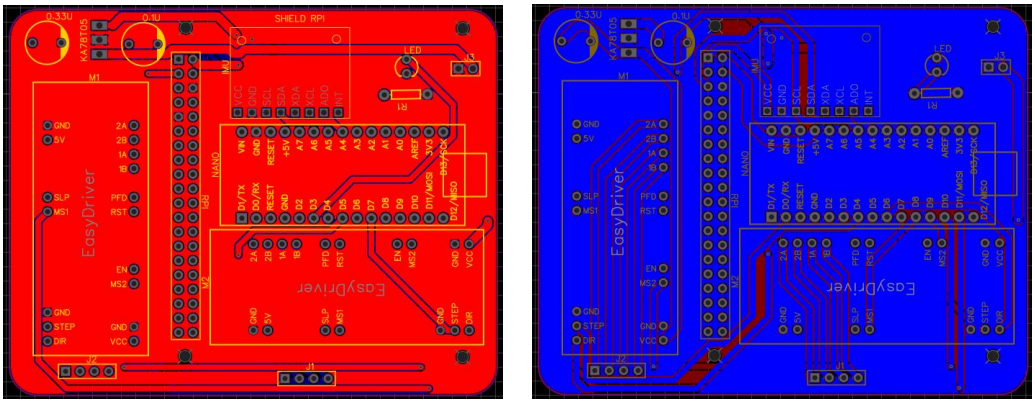


Figura 2.24 (a) Vista capa superior PCB. (b) Vista capa inferior PCB .

2.10 Módulo bluetooth

Con el fin de poder llevar a cabo un control remoto del vehículo, se ha optado por la incorporación de un módulo Bluetooth directamente conectado a Arduino, que nos permita el movimiento del vehículo por medio de una aplicación en nuestros teléfono móvil.

Se ha hecho uso del módulo Bluetooth HC-06, en el apartado de programación se detallará su configuración y funcionamiento.

Para la comunicación vía Bluetooth se hace uso de la App para Android Joystick BT commander, la cual nos permite controlar fácilmente el vehículo mediante el envío de comandos que serán proporciones a las coordenadas del joystick. Pudiendo realizar movimientos hacia delante y atrás y a ambos lados.

El joystick disponible en la aplicación sigue una circunferencia de valores enteros que comprenden desde el 0 al 100. La conversión dependerá de los límites de nuestro sistema. Para el movimiento

hacia atrás o delante, el dato enviado será la referencia en velocidad, mientras que para los giros, el dato enviado será el de la variable que aumente/disminuya la velocidad de las ruedas, recordemos que ambas velocidades pueden ser diferentes.

2.11 Esquema de conexiones

Una vez explicado todos los componentes que forman parte de nuestro proyecto, vamos a proceder a mostrar su esquema de conexiones.

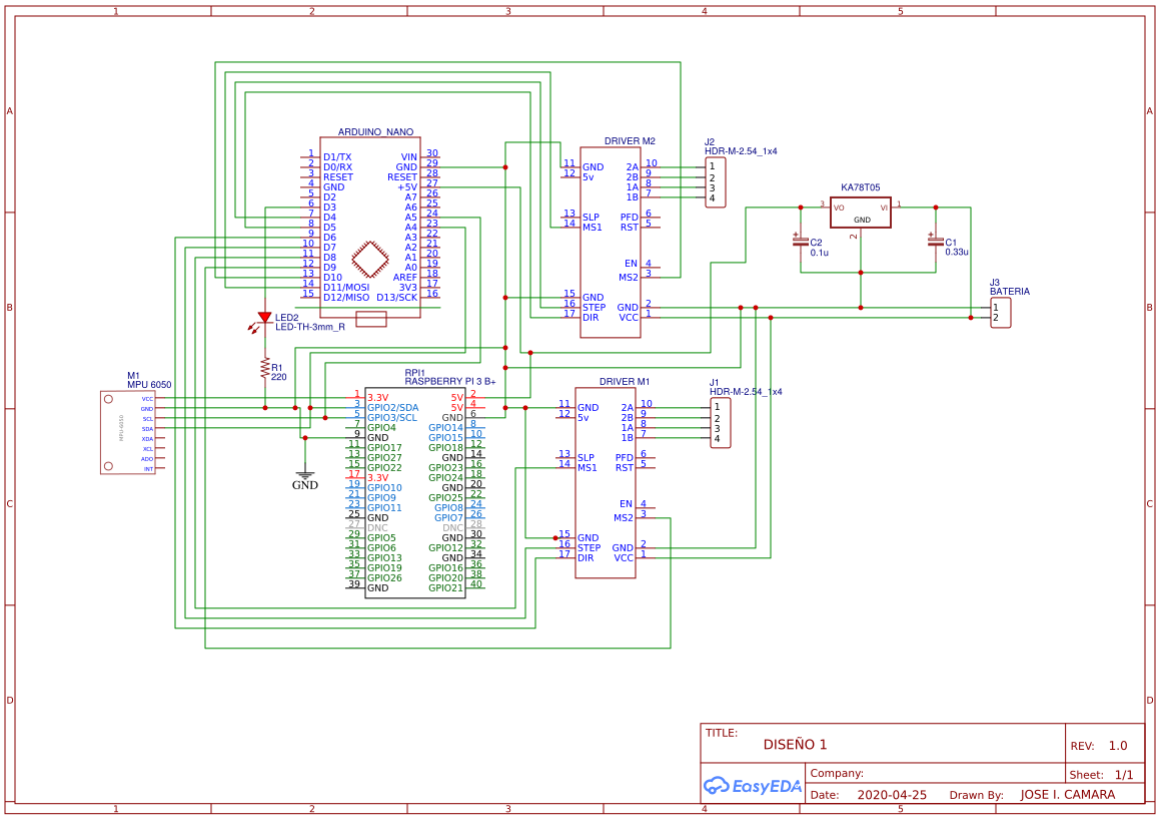


Figura 2.25 Esquema de conexiones.

Los dispositivos que podemos apreciar en el esquema anterior, son:

- Microcontrolador Arduino Uno.
- Raspberry Pi 3 modelo B.
- Dos EasyDriver v4.4 basado en el chip A3967.
- Un módulo GY-521 (unidad de medición inercial MPU-6050).
- Batería Li-Po de 11.1V
- Dos Motores paso a paso NEMA-17.
- Un regulador de tensión lineal KA75T05.
- Un led.
- Una resistencia de 220 ohmios.

- Un condensador de 0.33 μF y otro de 0.1 μF.

Nótese en el anterior esquema que no aparece el módulo bluetooth, ni el ventilador axial que ya se han comentado, esto es debido a que se añadieron posteriormente al diseño de la PCB.

2.12 Presupuesto

Tabla 2.12 Presupuesto.

Descripción	Cantidad	Precio unitario (€)	Precio (€)
Raspberry Pi 3B	1	37.43	37.43
Arduino Nano	1	4.66	4.66
Motor Nema 17	2	8.60	17.2
Controlador EasyDriver	2	3.90	7.80
GY-521 (MPU 6050)	1	2	2
PCB	1	1	1
Batería LIPO	1	14.50	14.50
Bluetooth HC 06	1	8	8

El precio total de fabricación del vehículo alcanza los 92.59 €. El resto de componentes mencionados en la memoria y que no aparecen en el presupuesto anterior, es debido a que su precio es irrelevante en comparación con el resto de componentes o bien porque se disponía de ellos antes del comienzo del proyecto.

3 Modelo

La clave para obtener un control eficiente de nuestro sistema, radica en la la obtención de un modelo que refleje el comportamiento real del mismo. Reflejar el comportamiento del sistema real es una tarea ardua y complicada que puede llevar a fallos, nuestro objetivo no será ese. Buscaremos la obtención de un modelo simplificado pero igualmente válido.

Por ello, en el presenta capítulo vamos a proceder al estudio del modelo físico-matemático que rige el comportamiento del vehículo. Para realizar el estudio vamos hacer uso de las ecuaciones energéticas, es decir, de la mecánica Lagrangiana. Para el estudio del modelo se ha tomado como referencia el trabajo de Fin de Grado de Dº Nicolás Cortés Fernández, titulado *Diseño, fabricación, montaje, estudio dinámico, control y teleoperación de un vehículo tipo péndulo invertido sobre dos ruedas* [6]. En el citado trabajo, se puede encontrar una deducción de las ecuaciones utilizando la dinámica vectorial de Newton, llegando a la misma conclusión que la aquí presente.

3.1 Modelado del sistema

Antes de establecer el modelo de control, tenemos que realizar un estudio del sistema. Como ya hemos mencionado, el sistema estará compuesto por un vehículo de dos ruedas basado en la configuración del péndulo invertido, cuyo eje de giro es coincidente con el eje de giro de ambas ruedas. Cada rueda se ha dotado de un motor independiente para dotar de capacidad de giro al vehículo. El sistema de control tratará de mantener en todo momento el vehículo en su posición vertical, que será su posición de equilibrio.

Notese en el montaje final del vehículo, como su posición de equilibrio no estará completamente en su vertical, sino unos pocos grados desplazado, en concreto 3.5°. Esto es debido a la distribución de los componentes y su peso dispar en el interior del vehículo. Por todo ello, cuando hagamos referencia al ángulo de inclinación con respecto a la vertical (ϕ), estaremos haciendo referencia al ángulo de inclinación corregido(ϕ_c).

$$\phi_c = \phi + \phi_0 \tag{3.1}$$

Siendo ϕ_0 el ángulo de desviación con respecto a la vertical.

3.2 Suposiciones y parámetros del sistema

A continuación mostraremos los parámetros del sistema que determinan su modelo.

- m_r : masa de cada rueda.
- R: Radio de cada rueda.

- M: Masa del vehículo excluyendo las ruedas.
- L: Distancia del eje de las ruedas al CDG del vehículo.
- g: Aceleración gravitatoria (se usará la aproximación 9.81 m/s^2).
- θ: Ángulo de giro de las ruedas.
- ϕ: Ángulo de inclinación del vehículo. Recuérdese que será el ángulo de giro corregido.

A modo de recuerdo, se citana continuación las hipótesis realizadas:

- Las ruedas presentan rodadura sin deslizamiento.
- La fricción en los balances de energía es despreciable.
- Las ruedas estarán en todo momento en contacto con la superficie.
- El vehículo se considera un cuerpo rígido y sólido, sin partes flexibles.

A continuación, se muestra el esquema del vehículo para obtener las ecuaciones que modelan su sistema. Dicho esquema puede encontrarse en [3]

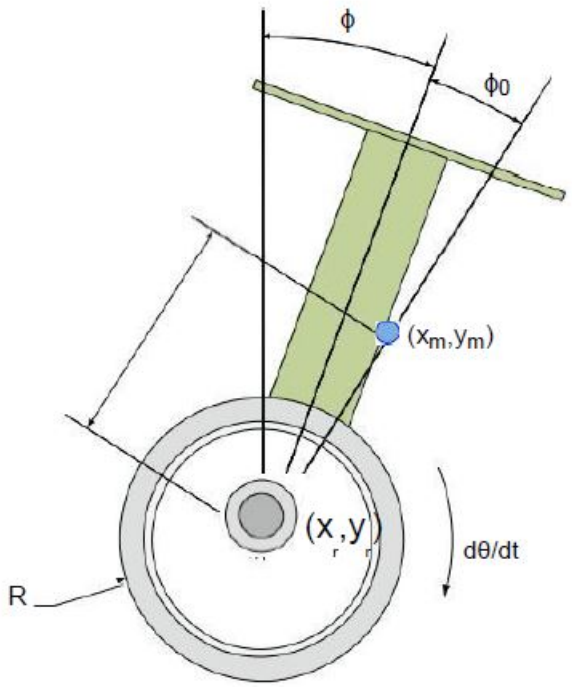


Figura 3.1 Esquema del modelo.

3.3 Aplicación de las ecuaciones energéticas. Mecánica Lagrangiana

Mediante las siguientes ecuaciones, seremos capaces de conocer el comportamiento dinámico del sistema.

Antes de aplicar las ecuacuines Lagrangianas, a modo de recordatorio, vamos a proceder a explicarlas en su forma general.

3.3.1 Ecuaciones Lagrangianas

La mecánica de Lagrange proporciona **n** ecuaciones diferenciales que corresponden a **n** coordenadas generalizadas. La ecuación de Lagrange en su forma general adquiere la expresión que apreciamos en la ecuación (3.2). En el miembro de la izquierda aparece el término L que será la Lagrangiana y q_i que representa una coordenada generalizada, siendo $\dot{q}_i$ la derivada con respecto al tiempo de la coordenada generalizada. En el miembro de la derecha aparece las fuerzas y momentos externos que se aplican al sistema.

$$\frac{d}{dt}\left(\frac{\partial L}{d\dot{q}_i}\right) - \frac{\partial L}{dq_i} = \tau_i \qquad i = 1,2,3... \tag{3.2}$$

$$L = T - V \tag{3.3}$$

La ecuación (3.3) muestra el balance energético. Como hemos indicado, L es el término Lagrangiano, siendo T la energía cinética y V la energía potencial (3.4). La energía cinética podemos descomponerla en energía cinética de rotación T_{rot} (3.5) y energía cinética de traslación T_{tras} (3.6).

$$V = Mgh \tag{3.4}$$

$$T_{rot} = \frac{1}{2} I w^2 \tag{3.5}$$

$$T_{tras} = \frac{1}{2} M v^2 \tag{3.6}$$

En la ecuación (3.5), el término I hace referencia al momento de inercia del sólido con respecto al eje de rotación, mientras que el término w^2 muestra la velocidad de rotación.

En la ecuación (3.6), el término M muestra la masa del sólido, y el término v^2 es la velocidad en el centro de masas.

3.3.2 Aplicación de las ecuaciones de Lagrange al sistema de estudio

Una vez introducidas las ecuaciones necesarias para realizar el balance energético, podemos proceder a aplicarlas en nuestro sistema. Lo primero que tenemos que hacer es definir las coordenadas generalizas del sistema, que serán:

- θ: Ángulo que forman las ruedas al girar con respecto a la vertical.
- ϕ : Ángulo de inclinación del vehículo.

Conocida las coordenadas generalizas, podemos proceder a particularizar la ecuación (3.2), donde obtendremos:

$$\frac{d}{dt}\left(\frac{\partial L}{d\dot{\theta}_i}\right) - \frac{\partial L}{d\theta_i} = \tau \tag{3.7}$$

$$\frac{d}{dt}\left(\frac{\partial L}{d\dot{\phi}_i}\right) - \frac{\partial L}{d\phi_i} = -\tau \tag{3.8}$$

En las ecuaciones anteriores, τ hace referencia al par motor externo que se aplica. Procedemos ahora a conocer los términos de dichas ecuaciones, para ello, necesitaremos calcular el Lagrangiano L que adquirirá la siguiente expresión:

$$L = T - V = T_{rot} + T_{tras} - V \tag{3.9}$$

Para aplicar las ecuaciones (3.4),(3.5)y (3.6), necesitamos conocer previamente las posiciones con respecto al centro de gravedad para, mediante su derivación con respecto al tiempo, obtener las velocidades lineales de las ruedas y del vehículo. Para obtenerlas, atenderemos a la figura 3.1.

Posición y velocidad del centro de gravedad de ambas ruedas en su plano horizontal:

$$x_r = R\theta; \dot{x}_r = R\dot{\theta} \tag{3.10}$$

Posición y velocidad del centro de gravedad de ambas ruedas en su plano vertical:

$$y_r = R; \dot{y}_r = 0 \tag{3.11}$$

Posición y velocidad del centro de gravedad del vehículo en su plano horizontal:

$$x_m = R\theta + L\sin\phi; \dot{x}_m = R\dot{\theta} + L\dot{\phi}\cos\phi \tag{3.12}$$

Posición y velocidad del centro de gravedad del vehículo en su plano vertical:

$$y_m = R + Lt\cos\phi; \dot{y}_m = -L\dot{\phi}\sin\phi \tag{3.13}$$

Con las ecuaciones (3.10) y (3.11), podemos obtener la v^2 de una rueda:

$$v^2 = (\dot{x_r}^2 + \dot{y_r}^2) = R^2\dot{\theta}^2 \tag{3.14}$$

Sustituyendo la ecuación (3.14) en (3.5) obtenemos la energía cinética de traslación de una rueda:

$$T^r_{tras} = \frac{1}{2}m_r R^2\dot{\theta}^2 \tag{3.15}$$

Con las ecuaciones (3.12) y (3.13), podemos obtener la v^2 del vehículo:

$$\begin{aligned} v^2 &= (\dot{x_m}^2 + \dot{y_m}^2) = (R\dot{\theta} + L\dot{\phi}\cos\phi)^2 + L^2\dot{\phi}^2\sin^2\phi = \\ &= R^2\dot{\theta}^2 + ^2 L\dot{\phi}^2\cos^2\phi + 2R\dot{\theta}L\dot{\phi}\cos\phi + L^2\dot{\phi}^2\sin^2\phi = \\ &= R^2\dot{\theta}^2 + L^2\dot{\phi}^2 + 2R\dot{\theta}L\dot{\phi}\cos\phi \end{aligned} \tag{3.16}$$

Sustituyendo la ecuación (3.16) en (3.6) obtenemos la energía cinética de traslación del vehículo:

$$T^v_{tras} = \frac{1}{2}M(R^2\dot{\theta}^2 + L^2\dot{\phi}^2 + 2R\dot{\theta}L\dot{\phi}\cos\phi) \tag{3.17}$$

La energía cinética total de traslación del sistema quedaría como:

$$T_{tras} = 2T^r_{tras} + T^v_{tras} = 2\frac{1}{2}m_r R^2\dot{\theta}^2 + \frac{1}{2}M(R^2\dot{\theta}^2 + L^2\dot{\phi}^2 + 2R\dot{\theta}L\dot{\phi}\cos\phi) \tag{3.18}$$

Para la obtención de la energía cinética de rotación, nos hará falta conocer los momentos de inercia. Usaremos su aproximación lineal:

- Momento de inercia del vehículo sin ruedas $I_{yy} = ML^2$
- Momento de inercia de una rueda $I_r = \frac{1}{2}m_r R^2$

Conocido los momentos de inercia y sustituyendo las expresiones en la ecuación (3.5) obtenemos los términos:

$$T^r_{rot} = \frac{1}{4}m_r R^2\theta^2 \tag{3.19}$$

$$T_{rot}^{v} = \frac{1}{2} M L^2 \dot{\phi}^2 \tag{3.20}$$

Quedando la energía total de rotación:

$$T_{rot} = 2T_{rot}^{r} + T_{rot}^{v} = 2\frac{1}{4} m_r R^2 \dot{\theta}^2 + \frac{1}{2} M L^2 \dot{\phi}^2 \tag{3.21}$$

La energía cinética total del sistema se calculará como la suma de las ecuaciones (3.18) y (3.21):

$$T = T_{rot} + T_{tras} = \left(\frac{3}{2} m_r + \frac{1}{2} M\right) R^2 \dot{\theta}^2 + M L^2 \dot{\phi}^2 + MRL\dot{\theta}\dot{\phi}\cos\phi \tag{3.22}$$

Para la obtención de la energía potencial usaremos la ecuación (3.4) sabiendo que la distancia entre ambos centro de masas es $L\cos\phi$, quedando:

$$V = -MgL\cos\phi \tag{3.23}$$

Sustituyendo las ecuaciones (3.22) y (3.23) en la ecuación (3.9), obtendremos el Lagrangiano del sistema:

$$L = \left(\frac{3}{2} m_r + \frac{1}{2} M\right) R^2 \dot{\theta}^2 + M L^2 \dot{\phi}^2 + MRL\dot{\theta}\dot{\phi}\cos\phi - MgL\cos\phi \tag{3.24}$$

Del cálculo de los términos de la ecuación (3.7) obtenemos los siguientes resultados.

$$\begin{aligned} \frac{\partial L}{\partial \dot{\theta}} &= 2\dot{\theta}\left(\frac{3}{2} m_r + \frac{1}{2} M\right) R^2 + MRL\dot{\phi}\cos(\phi) \\ \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} &= (3m_r + M) R^2 \ddot{\theta} + MRL\ddot{\phi}\cos\phi - MRL\dot{\phi}^2\cos\phi \\ \frac{\partial L}{\partial \theta} &= 0 \end{aligned} \tag{3.25}$$

Del cálculo de los términos de la ecuación (3.8) obtenemos los siguientes resultados.

$$\begin{aligned} \frac{\partial L}{\partial \dot{\phi}} &= 2ML^2\dot{\phi} + \dot{\theta}MRL\cos\phi \\ \frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} &= 2ML^2\ddot{\phi} + \ddot{\theta}MRL\cos\phi - \dot{\theta}\dot{\phi}MRL\sin\phi \\ \frac{\partial L}{\partial \phi} &= -MRL\dot{\phi}\dot{\theta}\sin\phi + MgL\sin\phi \end{aligned} \tag{3.26}$$

Combinando las ecuaciones (3.7) y (3.8) y sustituyendo en ella los términos de las ecuaciones (3.25) y (3.26) obtenemos la ecuación que modela el sistema.

$$(2ML^2 + MRL\cos\phi)\ddot{\phi} + [R^2(3m_r + M) + MRL\cos\phi]\ddot{\theta} - MRL\dot{\phi}\sin\phi - MgL\sin\phi = 0 \tag{3.27}$$

3.4 Obtención del modelo linealizado

A continuación, vamos a linealizar la ecuación (3.27) en torno a su punto de equilibrio inestable, el cual puede definirse como aquel punto donde el sistema pierde su estabilidad al ser sometida por una perturbación. Como hemos comentado en apartados anteriores, el punto de equilibrio inestable de nuestro sistema se encuentra en su vertical, es decir, cuando el ángulo que forma el vehículo con su vertical se anula.

Sabemos que el equilibrio inestable se produce cuando el centro de masas del vehículo se encuentra por encima de su eje de giro. Analizando la ecuación (3.27), obtenemos que este caso se produce cuando el ángulo ϕ adquiere un valor nulo. En dicha posición, la velocidad angular de las ruedas será una constante y por ello, la aceleración angular en el punto de equilibrio, será nula.

$$\phi_{eq} = 0 \tag{3.28}$$

$$\ddot{\theta}_{eq} = 0 \tag{3.29}$$

La función a linealizar en torno al punto de equilibrio, adquiere la siguiente expresión:

$$F(\phi,\dot{\phi},\ddot{\phi},\ddot{\theta}) = (2ML^2 + MRL\cos\phi)\ddot{\phi} + [R^2(3m_r + M) + MRL\cos\phi]\ddot{\theta} - MRL\dot{\phi}^2\sin\phi - MgL\sin\phi = 0 \tag{3.30}$$

Las derivadas parciales serán:

$$\frac{\partial F}{\partial \phi} = -MRL\ddot{\phi}\sin\phi - MRL\ddot{\theta}\sin\phi - MRL\dot{\phi}^2\cos\phi - MgL\cos\phi \tag{3.31}$$

$$\frac{\partial F}{\partial \dot{\phi}} = -2MRL\dot{\phi}\sin\phi \tag{3.32}$$

$$\frac{\partial F}{\partial \ddot{\phi}} = 2ML^2 + MRL\cos\phi \tag{3.33}$$

$$\frac{\partial F}{\partial \ddot{\theta}} = (3m_r + M)R^2 + MRL\cos\phi \tag{3.34}$$

Con las ecuaciones anteriores extrapoladas en su punto de equilibrio, obtenemos el modelo linealizado:

$$(2ML^2 + MRL)\ddot{\phi} + [(3m_r + M)R^2 + MRL]\ddot{\theta} - MgL\phi = 0 \tag{3.35}$$

3.5 Obtención de los parámetros del sistema

En el presente apartado se pretende calcular los parámetros que modelan el sistema. De la ecuación (3.35), observamos los parámetros necesarios, que serán:

- Masa del vehículo sin ruedas.
- Masa y radio de las ruedas.
- Valor de la gravedad.
- Distancia entre el centro de las ruedas y el CDG del vehículo.

Los valores anteriores son fácilmente obtenibles. Para el calculo de la masa del vehículo, se utilizará una báscula. Las ruedas al ser fabricadas por impresión 3D, el programa de laminado nos proporciona la cantidad de material utilizado y su peso, aunque se comprobará con una báscula que ambos coinciden. El radio de las ruedas será el indicado en el diseño 3D. La obtención de la distancia existente entre el centro de gravedad del vehículo y el centro de las ruedas, será algo más complicado, a continuación vamos a indicar dos posibles métodos y compararemos los resultados obtenidos.

El primer método consiste en sujetar el vehículo, con todos los componentes incorporadas a excepción de las ruedas, de varios extremos. Para ello, usaremos una cuerda de mayor longitud que el vehículo, a la que añadiremos un peso a su extremo inferior y esperaremos a que el vehículo no se

balanceé. Repetiremos esta medición desde diferentes ángulos, siendo el centro de masas el punto coincidente de todas las mediciones.

El segundo método para la obtención de L consistirá en colocar el vehículo con todos los componentes incorporadas, a excepción de las ruedas, horizontalmente sobre una superficie en canto, e ir variando su posición hasta conseguir que permanezca en equilibrio. Esta posición será donde se encuentre el CDG del vehículo.

Para el cálculo de los momentos de inercia de cada rueda y del vehículo sin ruedas se ha seguido su aproximación.

Con todas las consideraciones anteriores, el valor de los parámetros que conforman el sistema serán los siguientes:

Tabla 3.1 Parámetros del sistema.

Parámetro	Medición	Unidad
Masa del vehículo sin las ruedas	0.975	Kg
Masa de una rueda	0.064	Kg
Radio de una rueda	0.05	m
Longitud eje-cdg	0.05	m
Gravedad	9.81	m/s^2

3.6 Límites del sistema

A continuación se mostrarán los límites físicos y operativos del sistema que serán necesarios para establecer las restricciones en el cálculo del control MPC.

Tendremos que establecer los valores máximos y mínimos del estado del sistema que serán el ángulo de inclinación , la velocidad de caída y la velocidad de las ruedas. Así como los límites de la acción de control, que será la aceleración de las ruedas.

Para el ángulo de inclinación se sabe que su límite físico se sitúa en $\pm 90^{\circ}$, cuando el vehículo esta paralelo al suelo. El límite anterior no es operativo, pues el vehículo no es capaz de alcanzar tal inclinación y volver a su posición de equilibrio. Sin embargo, se deja como limitación dichos valores a los que no se llegará en ningún caso.

La velocidad de caída se estable entre unos límites de $\pm 4 rad/s$. Valores que son irrelevantes ya que no se llegarán en ningún caso.

El cálculo de los límites de la velocidad de las ruedas es algo más complejo, pues la velocidad máxima alcanzable vendrá determinada por dos factores, el motor que se utiliza y el controlador de dicho motor. En las especificaciones del motor se establece que tiene una frecuencia máxima de arranque en 1500 pps y una frecuencia máxima de funcionamiento de 8000 pps que podemos traducir en rad/s como se puede apreciar en la siguiente tabla:

Tabla 3.2 Velocidad máxima teórica.

nº pasos por rotación	200	400	800	1600
Velocidad de rotación (rad/s)	251.32	125.66	62.83	31.41

Estos serían los valores máximos para cada modo de micropaso. Sin embargo, estos valores no son alcanzable al incluir los motores en el vehículo y tener en cuenta las limitaciones del controlador EasyDriver. Para el cálculo de la velocidad máxima alcanzable se ha comprobado su valor experimentalmente. El procedimiento a seguir a sido ir dando poco a poco una aceleración

constante viendo hasta donde llega el motor, el valor alcanzado ronda los 60 rad/s, estableciendo este dato como limite de velocidad.

$$\begin{bmatrix} -90\frac{2\pi}{360} \\ -4 \\ -60 \end{bmatrix} \leq \begin{bmatrix} \phi \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} \leq \begin{bmatrix} 90\frac{2\pi}{360} \\ 4 \\ 60 \end{bmatrix} \begin{pmatrix} rad \\ rad/s \\ rad/s \end{pmatrix} \tag{3.36}$$

El cálculo de la aceleración máxima que puede proporcionarse a los motores, se obtendrá experimentalmente, siendo un buen candidato$\pm 90 rad/s^2$. La elección de este valor puede apreciarse en las gráficas de los controladores LQR y LQRI, donde se vio que estos valores eran suficientes y alcanzables.

$$-90 \leq u \leq 90 \left(\frac{rad}{s^2} \right) \tag{3.37}$$

4 Control

El sistema basado en el péndulo invertido es un sistema aparentemente inestable lo cual supone un gran reto para el control automático del mismo. A lo largo de los años, muchos han sido los métodos y variantes de control que se han utilizado para dotar de estabilidad a este tipo de experimentos.

En el presente capítulo, se calcularán y explicarán las diferentes estrategias de control que, posteriormente, van a aplicarse al modelo real. Empezaremos el estudio obteniendo el modelo matemático en el espacio de estados, para seguir con un control LQR y un control LQR con adición de un término integral así como la implementación de un control MPC. Finalizaremos explicando el marco teórico de los sistema a implementar.

Por lo tanto, los objetivos de este apartado serán:

- Verificación de las ecuaciones del modelo calculadas en el apartado 3.
- Análisis de las diferentes estrategias de control, evaluando la validez de cada una de ellas.
- Implementación del control de estabilidad, resolviendo los problemas de orientación del mismo.

4.1 Variables de estado

Una vez obtenida la ecuación que modela el sistema y su linealizado, podemos proceder al desarrollo de controladores lineales en variables de estado.

Podemos definir las variables de estado como el conjunto de variables de un sistema cuyo conocimiento, en cualquier momento, permite conocer la evolución del sistema en un futuro.

Las variables que determinan el estado en el que se encuentra nuestro sistema en todo momento, serán las siguientes:

- ϕ : Ángulo de inclinación del vehículo en [rad].
- $\dot{\phi}$: Velocidad angular del vehículo [rad/s].
- $\dot{\theta}$: Velocidad angular de las ruedas[rad/s].

Las dos primeras variables serán fácilmente obtenibles, pues su dato puede tomarse directamente del dispositivo IMU, en el capítulo 5 "Software" veremos la forma de calcular la tercera variable.

4.1.1 Espacio de estados

Conocidas las variables de estado y el modelo matemático del sistema, se va a proceder a obtener el modelo en espacio de estados.

El espacio de estados puede definirse como la representación, en ecuaciones diferenciales de primer orden, del modelo matemático mediante el conjunto de entradas, salidas y variables internas. Estas ecuaciones diferenciales de primer orden se combinan en una ecuación diferencial matricial.

A tenor de la definición anterior, el objetivo será convertir la ecuación (3.35) en una ecuación diferencial en términos matriciales.

El sistema ante el que estamos presente, es un sistema dinámico y continuo, en el que las variables evolucionan con el tiempo. Por ello, será necesario la adición de una serie de mecanismos que nos den información, en cada momento, del estado y la evolución del sistema.

A modo de ejemplo, si en un determinado instante el vehículo se encuentra inclinado, para enderezarlo necesitaremos el ángulo de inclinación con la vertical medido por el sensor. Este dato será insuficiente para el correcto control del vehículo y aplicación de la acción de control, que como es sabido será la aceleración de las ruedas. Necesitaremos pues, conocer otros parámetros como la velocidad de las ruedas. En la necesidad de conocer este dato radica el concepto de "estado del sistema". Es decir, necesitamos conocer en que condiciones se encuentra el vehículo en el instante de estudio.

Es esencial, representar la salida del sistema en función del estado y del valor de las entradas.

$$\ddot{\theta} = f(x(t), u(t)) \tag{4.1}$$

La ecuación matricial que representa el modelo en espacio de estados en forma continua adquiere la siguiente forma:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{4.2}$$

$$y(t) = Cx(t) + Du(t) \tag{4.3}$$

Donde:

- x: Vector de estados (Vector de dimensión n).
- u: Vector de entradas (Vector de dimensión m).
- y: Vector de salidas (Vector de dimensión n).
- A: Matriz del sistema (Dimensión nxn).
- B: Matriz de entrada (Dimensión nxm).
- C: Matriz de salida.

La ecuación (4.2) es la encargada de obtener el estado siguiente a partir del estado actual y de la variable de control. La ecuación (4.3) obtiene la salida deseada. En dicha ecuación, el término matricial D se puede obviar ya que si asumimos que la entrada no afecta a la salida al mismo tiempo, su valor se hace nulo.

4.1.2 Espacio de estados particularizado a nuestro sistema

Particularizando la ecuación anterior para nuestro sistema, se tiene:

$$\begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} = A \begin{bmatrix} \phi \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} + B\ddot{\theta} \tag{4.4}$$

El objetivo radica ahora en el cálculo de las matrices A y B, para ello vamos a agrupar la ecuación (3.35) para que adquiera la siguiente forma:

$$a\ddot{\phi} + b\ddot{\theta} - c\dot{\phi} = 0 \tag{4.5}$$

Siendo los valores de las constantes anteriores a,b y c:

$$\begin{aligned} a &= 2ML^2 + MRL \\ b &= (3m_r + M)R^2 + MRL \\ c &= MgL \end{aligned} \tag{4.6}$$

Con los valores anteriores, obtenemos las matrices A y B que serán sustituidas en la ecuación (4.4) que junto a la ecuación (4.3) obtendremos la descripción del sistema en el espacio de estados, cabe recordar que la expresión siguiente se encuentra en forma continua:

$$\begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ c/a & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ -b/a \\ 1 \end{bmatrix} \ddot{\theta} \tag{4.7}$$

Como necesitamos trabajar en tiempo discreto, vamos a hacer uso del siguiente código en MATLAB. El código nos proporciona las matrices A y B del sistema en tiempo discreto, partiendo de los parámetros del sistema y sus matrices en modo continuo.

Para el cálculo de los valores siguientes, se tiene que hacer un cambio del sistema en tiempo continuo al sistema en tiempo discreto.

$$\dot{x}(t) = Ax(t) + Bu(t) \rightarrow x_{k+1} = A_d x_k + B_d u_k \tag{4.8}$$

$$t = T_m \cdot k \tag{4.9}$$

```
clear ;
clc;

%Parámetros del modelo en el SI
M = 0.975 ;
L = 0.05;
R = 0.05;
mr = 0.064;
Iyy = 2*M*L^2;
g = 9.81;

A = [0 1 0; (M*g*L/(Iyy + M*R*L)) 0 0; 0 0 0];
B = [0; -(R^2*(3*mr+M)+M*R*L)/(Iyy + M*R*L); 1];
C = eye(3);
D = zeros(3, 1);

%Creación del modelo continuo en el espacio de estados
Modelo_continuo=ss(A,B,C,D);

%Discretización del modelo anterior con un Tm de 20 ms
Modelo_discreto=c2d(Modelo_continuo,0.02);

%Representación de las matrices discretas
[Ad,Bd,Cd,Dd]=ssdata(Modelo_discreto);
```

Los valores obtenidos, en tiempo continuo, serán:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 65.4 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$B = \begin{bmatrix} 0 \\ -0.7283 \\ 1 \end{bmatrix} \tag{4.10}$$

Mientras que en tiempo discreto adquieren:

$$A = \begin{bmatrix} 1.0131 & 0.0201 & 0 \\ 1.3137 & 1.0131 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$B = \begin{bmatrix} -0.0001 \\ -0.0146 \\ 0.02 \end{bmatrix} \tag{4.11}$$

4.2 Formulación general del control LQR

El controlador *Linear Quadratic Regulator* (por sus siglas en inglés, LQR) pretende conseguir un control óptimo a un mínimo coste. Por ello, esta estrategia de control busca la obtención de una matriz de realimentación K que minimice la función energética o de coste, la cual adquiere la siguiente forma:

$$J = \frac{1}{2}x^T(t_f)P_x^T + \frac{1}{2}\int_{t_0}^{t_f}(x^TQx + u^TRu)dt \tag{4.12}$$

La ecuación anterior, puede simplificarse si asumimos un tiempo final infinito ($t_f \mapsto \infty$), lo cual hace que el estado $x(t_f)$ se anule y la función de coste J quede como tal:

$$J = \int_0^{\infty}(x^TQx + u^TRu)dt \tag{4.13}$$

La función J de la ecuación (4.12) está expresada en su término general y está asociada con la energía del sistema.

El término Q de la ecuación anterior es la matriz de ponderación sobre el error en el estado, mientras que R es la ponderación sobre el error en la señal de control. Tanto Q como R son matrices semidefinidas positivas, de dimensión nxn la primera y mxm la segunda, donde n es el número de variables de estado y m el número de entradas.

La ley de control seleccionada consistirá en una realimentación negativa del vector de estados.

$$u(t) = -Kx(t) \tag{4.14}$$

El vector K es el vector de ganancias de ralimentación de estados. Sustituyendo el termino anterior en la ecuación (4.2)

$$\dot{x}(t) = Ax(t) - BKx(t) = (A - BK)x(t) \tag{4.15}$$

En el resultado obtenido, observamos que el estado futuro del sistema depende tanto de la entrada como del estado del sistema en el momento actual.

Como se ha indicado, el objetivo del control LQR será obtener la matriz de realimentación K que minimice el funcional J. Para ello, tenemos que resolver la ecuación:

$$K = R^{-1}B'P \tag{4.16}$$

Para el cálculo de la matriz P, que debe ser definida positiva, se tiene que hacer uso de la ecuación de Riccati.

$$A'P + PA - PBR^{-1}B'P + Q = 0 \tag{4.17}$$

.

Una vez obtenida la matriz P, se debe sustituir en la ecuación (4.16) , consiguiendo así la obtención de la matriz de realimentación. Observando la ecuación de Riccatti, se aprecia que la obtención de la matriz P es sumamente complicada, dando lugar a cálculos largos y tediosos, propenso a fallos, donde aparecen matrices inversas.

Para evitar los cálculos anteriores, se puede hacer uso de la función de Matlab "lqr" en tiempo continuo o "dlqr" si la representación del sistema es en tiempo discreto .

La ley de control LQR debe aplicarse sobre modelos linealizados. Recordemos pues la ecuación (4.7), la cual mostraba el modelo del sistema linealizado entorno al punto de equilibrio en forma de espacio de estados. El vector x de dimensiones $3x1$ hace referencia a las variables de estado. Por ello, la matriz K será de dimensiones 1x3, siendo el objetivo ahora, el cálculo de dichas 3 componentes. Se debe cumplir que la matriz de realimentación K minimice el valor del funcional J, es decir, se asegure que la energía del sistema se minimice al máximo haciendo que el sistema sea estable. Obtenida la matriz K por Matlab, y sustituida en la ecuación (4.14) obtenemos el vector de entrada que realimenta el sistema.

$$u = -Kx = -\begin{bmatrix} K_1 & K_2 & K_3 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} \tag{4.18}$$

4.2.1 Formulación LQR en sistemas discretos

Dado que tenemos que trabajar en un espacio discreto, tendremos que partir de la ecuación (4.13), la cual se encuentra representada en tiempo continuo y horizonte infinito, para obtener funcional J en tiempo discreto y horizonte finito:

$$J = \sum_{k=0}^{N} [X_n^T Q X_n + U_n^T R U_n] \tag{4.19}$$

siendo,

- $X_n = x_k - x_e$
- $U_n = u_k - u_e$

Como es sabido, el control optimo busca minimizar la energía aportada por cada estado ($X_n^T Q X_n$). Siendo el objetivo, el cálculo de la ley de control discreta,

$$u_k = -K(X_n) + u_e \tag{4.20}$$

El término x_e hace referencia al punto de equilibrio siendo,

$$x_e = Ax_e + Bu_e \tag{4.21}$$

Aplicando el cambio de variable $Z_{k+1} = x_{k+1} - x_e$ llegamos a las ecuaciones,

$$Z_{k+1} = (A - BK)Z_k \tag{4.22}$$

$$J = \sum_{k=0}^{N} Z_k^T (A - BK)Z_k \tag{4.23}$$

4.2.2 Calculo del Vector de realimentación K

Para el cálculo del vector de realimentación K se usará la función *dlqr* implementada en el programa Matlab. La función *dlqr*, recibe como argumento de entrada las matrices A, B y C, de la descripción del sistema en modo discreto, así como las matrices de ponderación Q y R.

Para el cálculo de las matrices A, B y C se usarán los parámetros del modelo que se han obtenido en apartados anteriores. Posteriormente, se usará la función de matlab *ss* para crear el modelo en el espacio de estados, finalmente con la función *c2d* se discretizará el modelo de estado anterior.

Q y R son matrices que han de definirse heurísticamente de acuerdo a una ponderación. Como ya sabemos, la matriz Q busca minimizar la energía empleada por las variables de estado, mientras que la matriz R busca minimizar la energía empleada por su entrada.

```
clear ;
clc;

%Parámetros del modelo en el SI
M2 = 0.975 + 0.064;
L = 0.05;
R = 0.05;
mr = 0.064;
Iyy = 2*M2*L^2;
g = 9.81;

%Matrices del sistema en modo continuo
A = [0 1 0; (M2*g*L/(Iyy + M2*R*L)) 0 0; 0 0 0];
B = [0; -(R^2*(3*mr+M2)+M2*R*L)/(Iyy + M2*R*L); 1];
C = eye(3);
D = zeros(3, 1);

%Creación del modelo continuo en el espacio de estados
Modelo_continuo=ss(A,B,C,D);

%Discretización del modelo anterior con un Tm de 20 ms
Modelo_discreto=c2d(Modelo_continuo,0.02);

%Representación de las matrices discretas
[Ad,Bd,Cd,Dd]=ssdata(Modelo_discreto);

%%obtención del vector de realimentación K
Q=[10 0 0 ; 0 1 0 ; 0 0 10 ];
R=0.1;
[k]=dlqr(Ad,Bd,Q,R)
```

Una vez calculado el vector K, podemos proceder a calcular los autovalores del sistema en bucle cerrado (A-BK), obteniendo:

$$K = \begin{bmatrix} -337.7394 & -41.8230 & -7.6894 \end{bmatrix} \tag{4.24}$$

$$Autovalores = \begin{bmatrix} -8.4018 \\ -7.1035 + 2.8725i \\ -7.1035 - 2.8725i \end{bmatrix} \tag{4.25}$$

De los autovalores anteriores, obtenemos que el sistema cuenta con un polo rápido en,

$$\tau = \frac{1}{8.4018} = 0.1190 \tag{4.26}$$

y con dos polos lentos en ,

$$\tau = \frac{1}{7.1835} = 0.1392 \tag{4.27}$$

Siendo menor que el tiempo de muestreo que hemos establecido en 20 ms, por lo que deducimos que el planteamiento es correcto.

4.2.3 Formulación LQR con efecto integral

El control LQRI consiste en adicionar un término integral que corrija la perturbación causada cuando el centro de gravedad del robot no se encuentre en su centro geométrico. La explicación del porqué se adiciona este término, podemos encontrarla en [3]

Para evitar la influencia de la perturbación, se introduce la siguiente ecuación que se encuentra representada en variables incrementales en tiempo discreto

$$\Delta x_{k+1} = A\Delta x_k + B\Delta u_k + E\Delta w_k \tag{4.28}$$

Al ser la perturbación constante, su variación es cero y por ello se anula el último término de la ecuación anterior, quedando:

$$\Delta x_{k+1} = A\Delta x_k + B\Delta u_k \tag{4.29}$$

El error en tiempo discreto puede representarse como:

$$\Delta e_{k+1} = A\Delta r_k - \Delta x_{k+1} \tag{4.30}$$

Si consideramos una referencia constante, el primer término de la ecuación anterior puede considerarse nulo y por tanto, el sistema adquiere la siguiente forma:

$$\begin{bmatrix} \Delta x_{k+1} \\ e_k \end{bmatrix} = \begin{bmatrix} A & 0 \\ -I & I \end{bmatrix} \begin{bmatrix} \Delta x_k \\ e_{k-1} \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} \Delta u_k \tag{4.31}$$

Suponiendo que la acción de control es una ganancia, se tiene:

$$\Delta u_k = -K \begin{bmatrix} \Delta x_k \\ e_{k-1} \end{bmatrix} = K_x \Delta x_k + K_e \Delta e_{k-1} \tag{4.32}$$

Y sustituyendo en (4.31):

$$\begin{bmatrix} \Delta x_{k+1} \\ e_k \end{bmatrix} = \left(\begin{bmatrix} A & 0 \\ -I & I \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} K \right) \begin{bmatrix} \Delta x_k \\ e_{k-1} \end{bmatrix} \tag{4.33}$$

Entonces obtenemos que:

$$u_n = u_0 + K_x(x_n - x_0) + K_e \sum_{i=0}^{n-1} e_i \tag{4.34}$$

El nuevo vector K estará compuesto por 4 componentes, las tres primeras serán las calculas en el apartado anterior, la cuarta se obtendrá experimentalmente. La componente primera penaliza el

error de inclinación, la segunda la velocidad de inclinación, la tercera la velocidad de las ruedas y la cuarta el error acumulado en la velocidad de las ruedas.

4.3 Control Predictivo basado en el Modelo MPC

El control predictivo basado en el modelo (MPC), es una técnica de control avanzado, en el que se establece el estado del sistema a partir de las referencias que se desean alcanzar en el instante actual más las referencias futuras. Es decir, es necesario disponer del conocimiento de la situación actual para la predicción. Se establece un intervalo de tiempo que abarca desde el tiempo actual hasta el tiempo actual más la suma de un horizonte de predicción.

4.3.1 Estructura general del modelo

La formulación general del modelo asume que el sistema cuenta con m entradas, q salidas y n estados. Las ecuaciones del modelo en espacio de estados discreto es lineal, finito e invariable en el tiempo y adquieren la siguiente forma.

$$x(k+1) = Ax(k+1) + Bu_k \tag{4.35}$$

$$y(k) = Cx(k) + Du(k) \tag{4.36}$$

En la ecuación anterior, se asume que la entrada no afecta a la salida al mismo tiempo y por ello, el término $D = 0$. Por ello, la ecuación 4.36 queda:

$$y(k) = Cx(k) \tag{4.37}$$

En las ecuaciones (4.35) y (4.36), A, B y C representan las matrices del espacio de estados en tiempo discreto, siendo x el vector de estados de dimensión n, y representa la salida del sistema y u la variable de entrada.

Para tener información tanto del estado actual como del estado pasado, debemos trabajar con variables incrementales, de tal manera:

$$\Delta x = x(k) - x(k-1) \tag{4.38}$$

$$\Delta u = u(k) - u(k-1) \tag{4.39}$$

Si relacionamos la salida $y(k)$ con el vector de estados $\Delta x_m(k)$ se tiene.

$$\Delta y(k+1) = C\Delta x(k+1) = CA\Delta x(k+1) + CB\Delta u(k+1) \tag{4.40}$$

Donde:

$$\Delta y(k+1) = y(k+1) - y(k) \tag{4.41}$$

Con todo ello, la nueva representación del vector de estado será;

$$\begin{bmatrix} \Delta x(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A & 0^T \\ CA & I \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ y(k) \end{bmatrix} + \begin{bmatrix} B \\ CB \end{bmatrix} \Delta u(k) \tag{4.42}$$

$$y(k) = \begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ y(k) \end{bmatrix} \tag{4.43}$$

Donde I es la matriz identidad de orden q, O es la matriz de ceros de dimensione qxn, A es una matriz cuadrada de orden n, B es de dimensiones nxm y C de q x n.

4.3.2 Predicción de las variables de estado y las salidas

La trayectoria futura vendrá determinada por la ecuación

$$\Delta U = [\Delta u(k_i), \Delta u(k_i+1), ..., \Delta u(k_i+N_c-1)]^T \tag{4.44}$$

Siendo las variables de estado futuras :

$$X = [\Delta x(k_i+1|k_i), \Delta x(k_i+2|k_i), ..., \Delta x(k_i+N_p|k_i)]^T \tag{4.45}$$

En las ecuaciones anteriores aparece el término N_c que hace referencia al horizonte de control, mientras que N_p es el horizonte de predicción. Debe cumplirse siempre que el horizonte de control sea menor que el horizonte de predicción. Las variables de salida predichas serán:

$$Y = [y(k_i+1|k_i), y(k_i+2|k_i), ..., y(k_i+N_p|k_i)]^T \tag{4.46}$$

La variable de salida anterior estará relacionada con la trayectoria futura por la siguiente ecuación:

$$Y = Fx(k_i) + \Phi \Delta U \tag{4.47}$$

Donde F es la matriz de observalidad:

$$F = \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{N_P} \end{bmatrix}; \Phi = \begin{bmatrix} CB & 0 & 0 & \cdots & 0 \\ CAB & CB & 0 & \cdots & 0 \\ CA^2B & CAB & CB & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ CA^{N_p-1}B & CA^{N_p-2}B & CA^{N_p-3}B & \cdots & CA^{N_p-N_C}B \end{bmatrix} \tag{4.48}$$

4.3.3 Breve descripción del MPC para formulación de seguimiento

En los apartados anteriores se presenta la formulación general del MPC para tener una idea de dicho control. En el presente apartado se va a describir el control MPC para seguimiento que se usará en el trabajo. Para más información se puede acudir a [7]

Consideramos el sistema descrito por el modelo de espacio de estado lineal, discreto e invariante en el tiempo que se muestra en la ecuación (4.35). En dicha ecuación, $x_k \in \mathbb{R}^n$ y $u_k \in \mathbb{R}^m$ son el estado y la entrada del sistema de control en el tiempo de muestreo k respectivamente. Además consideramos que el sistema está sujeto a las restricciones:

$$\underline{x} \leq x_k \leq \overline{x} \tag{4.49a}$$

$$\underline{u} \leq u_k \leq \overline{u}. \tag{4.49b}$$

El objetivo del control es dirigir el sistema a la referencia dada (x_r, u_r) mientras se satisfacen las restricciones del sistema. Esto solo es posible si la referencia dada es una referencia admisible del sistema, que definimos formalmente en la siguiente definición.

Definición: La pareja $(x_a, u_a) \in \mathbb{R}^n \in \mathbb{R}^m$ se dice que es admisible para el sistema (4.35) sujeto a (4.49) si *(i)* $x_a = Ax_a + Bu_a$, es un estado estable del sistema (4.35), *(ii)* $\underline{x} \leq x_a \leq \overline{x}$,y *(iii)* $\underline{u} \leq u_a \leq \overline{u}$.

Si la referencia dada no es un estado admisible del sistema (4.35) sujeto a (4.49), entonces se dirigirá el sistema al estado estable admisible más próximo.

Modelo predictivo para el control de seguimiento

Esta sección describe brevemente la formulación del MPCT [7]. Para un horizonte de predicción dado N, la ley de control MPCT para un estado dado x y la referencia (x_r, u_r) se deriva de la solución del siguiente problema de optimización convexa dada por $\mathbb{T}(x; x_r, u_r)$.

$$\mathbb{T}(x; x_r, u_r) : \min_{\mathbf{x},\mathbf{u},x_s,us} \sum_{i=0}^{N-1} \|x_i - x_s\|_Q^2 + \sum_{i=0}^{N-1} \|u_i - u_s\|_R^2 + \|x_s - x_r\|_T^2 + \|u_s - u_r\|_S^2 \tag{4.50a}$$

$$s.t.\ x_0 = x \tag{4.50b}$$

$$x_{i+1} = Ax_i + Bu_i,\ i = 0, \ldots, N-1 \tag{4.50c}$$

$$\underline{x} \leq x_i \leq \overline{x},\ i = 0, \ldots, N-1 \tag{4.50d}$$

$$\underline{u} \leq u_i \leq \overline{u},\ i = 0, \ldots, N-1 \tag{4.50e}$$

$$x_s = Ax_s + Bu_s \tag{4.50f}$$

$$\underline{x} \leq x_s \leq \overline{x} \tag{4.50g}$$

$$\underline{u} \leq u_s \leq \overline{u} \tag{4.50h}$$

$$x_N = x_s, \tag{4.50i}$$

donde las variables de decisión son los estados predichos y las entradas $\mathbf{x} = (x_0, \ldots, x_N)$, $\mathbf{u} = (u_0, \ldots, u_N)$ y la referencia artificial(x_s, u_s); y Q, R, T y S son matrices diagonales definidas positivas.

5 Software

Una vez llegado a este punto de la memoria, ya tenemos montado el vehículo mini Segway y explicado cada uno de los componentes que lo forman. También tenemos el modelo matemático que rige su funcionamiento, así como la explicación teórica de los diferentes controladores que vamos a proceder a implementar, para posteriormente analizar sus resultados y decidir que técnica es la que mejor se adapta a los resultados requeridos.

Este capítulo se puede dividir en tres apartados. Por un lado se explicará el código a implementar en Arduino, el cuál se encargará del control de la velocidad e implementación de la aceleración en ambos motores. Dicho código parte del trabajo realizado por Jose Antonio Borja Conde [4], por ello, no se explicará en detalle, simplemente se dará una serie de pinceladas para su compresión, para mayor indagación se puede acudir a su trabajo titulado *"Desarrollo de un robot autoequilibrado basado en Arduino con motores paso a paso"*, el cual se encuentra citado en las referencias.

En segundo lugar, se explicará la forma de obtener el ángulo de inclinación del vehículo partiendo de los datos obtenidos por la IMU. Se explicará y demostrará la necesidad de programar un filtro complementario para el cálculo del ángulo, mediante la combinación de los datos que proporciona el acelerómetro y el giroscopio.

Por último, se procederá a explicar el código de las diferentes técnicas de control calculadas en el capítulo anterior. Esta implementación tendrá lugar en la RPi, siendo la encargada de la obtención de la señal de control, que como ya se ha indicado es la aceleración de las ruedas. El dato anterior deberá ser enviado a Arduino, por ello se explicará también en este capítulo la forma de conexión entre ambas placas.

En el siguiente esquema podemos apreciar la interconexión entre los códigos de programación que se han mencionado. Muy resumidamente el funcionamiento en su conjunto sería el siguiente: Por un lado, se encuentra la Rpi que será la encargada de implementar el control a alto nivel. La RPi se encargará pues, de procesar el ángulo de inclinación y la velocidad de caída del vehículo que recibe del dispositivo GY-521 (MPU-6050), posteriormente preguntará a Arduino cuál es la velocidad de las ruedas en ese instante y con dichos datos calculará la señal de control. La señal de control, que será la aceleración angular de las ruedas, será enviada a Arduino que mediante un tren de pulsos aplicado a los controladores de los motores paso a paso producirá el movimiento de las ruedas.

Se ha incorporado un módulo Bluetooth conectado a Arduino que nos permitirá controlar remotamente el vehículo. El Bluetooth recibirá dos valores desde una App Android. El primer valor será el encargado de hacer girar el vehículo y será procesado en Arduino, el segundo valor que recibe será la referencia de velocidad que tendrá que ser enviada a la RPi para que el control lo tenga en cuenta a la hora de calcular la acción de control.

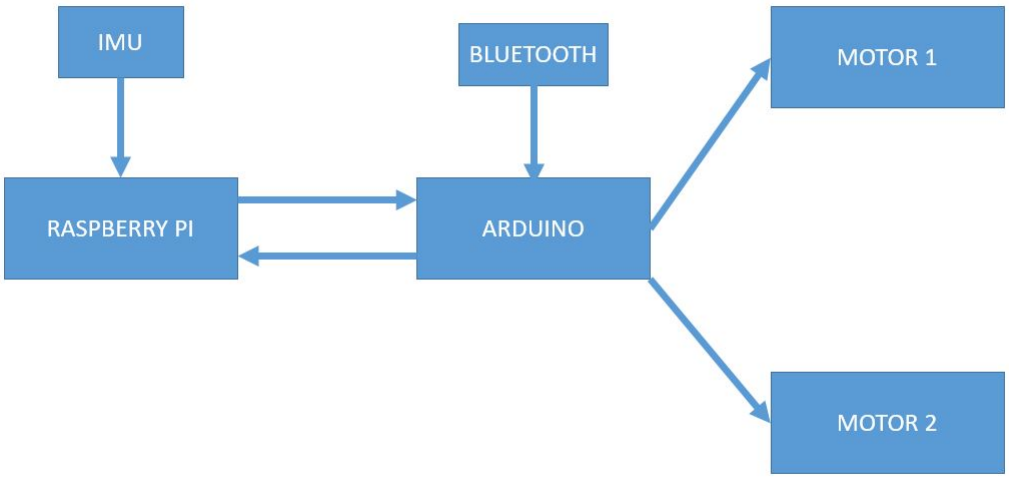


Figura 5.1 Esquema de comunicación entre los dispositivos.

5.1 Programación Arduino

El microcontrolador Arduino será el encargado del control de los motores paso a paso, en concreto, será el encargado de dotar a las ruedas de la velocidad necesaria para su funcionamiento. Para conseguir velocidad en los motores es necesario aplicar un tren de pulsos con una frecuencia determinada sobre el pin STEP de cada controlador Easydriver.

Como es sabido, Arduino es una plataforma de desarrollo de código abierto con la filosofía de proporcionar un software y hadware libre, sencillo y flexible. Para que se cumpla el principio de software libre, Arduino proporciona un entorno de desarrollo integrado (IDE por sus siglas en inglés) que nos permite crear diferentes programas para interactuar con diferentes dispositivos conectados a ella. Arduino cuenta con una gran comunidad que crea y comparte conocimiento, creando librerías que son puestas a disposición de toda la comunidad. En este punto, y centrándonos en nuestro proyecto, podemos encontrar una serie de librerías para controlar los motores paso a paso.

El IDE de Arduino cuenta con las librerías Stepper.h y AccelStepper.h. Ambas son capaces de dotar a los motores paso a paso del tren de pulsos necesario para que se produzca movimiento, la diferencia entre ellas es que la segunda permite llegar a una velocidad determinada mediante una aceleración establecida. Inicialmente se pretendió controlar la aceleración de los motores haciendo uso de esta última librería, sin embargo, los resultados no fueron satisfactorios.

Para el funcionamiento de la librería AcelSepper, es necesario definir una velocidad máxima en pasos por segundo y una aceleración máxima en pasos por segundo al cuadrado. Una vez definidas dicha velocidad máxima y aceleración máxima, hay que establecer una serie de pasos totales que serán los que queramos que se mueva el motor. Con todos los datos anteriores, la librería calcula en cada iteración el número de pasos que a de desplazarse con una aceleración constante para mover el motor hasta que se llega a su máxima velocidad y, en ese momento se comienza a desacelerar el motor, todo ello dentro del rango de pasos totales que se le indica que se mueva.

El cálculo de la aceleración es mediante una ecuación cuadrada, y los términos computacionales superaban las restricciones temporales que requiere nuestro proyecto, tras numerosas pruebas e intentos se descarto esta opción.

Otro problema que presentaba la librería era la gestión de pasos, pues nosotros necesitamos precisión y por lo tanto, trabajar con los diferentes modos de pasos que nos proporciona el controlador Easydriver. Sin embargo, mientras se llega a la velocidad deseada en el número de pasos que se establece, no se puede cambiar el modo de paso. El programa queda bloqueado hasta que realiza la acción, es decir, hasta que termina de mover el número de pasos indicados.

El uso de la librería AccelStepper podría haber simplificado mucho el desarrollo del trabajo, pero vista la imposibilidad de usar esta librería, se procedió al estudio del código desarrollado por Jose Antonio Borja [4]. Dicho código es óptimo y satisfactorio, dando unos resultados magníficos, basándose en la generación de un tren de pulsos a cada motor mediante el uso de interrupciones temporales. Las interrupciones temporales se llevarán a cabo por los Timers. Arduino Nano cuenta con 3 Timers.

5.1.1 Introducción a los interrupciones Arduino

Podemos entender como interrupción el mecanismo por el cual salta a ejecución una función cuando se activa un evento determinado, esta función recibe el nombre de ISR (Interruption Service Rutine).

El funcionamiento de una interrupción es simple, cuando ocurre un determinado evento al que se asocia dicha interrupción, el procesador abandona la tarea que este realizando previo salvado del estado del procesador y del valor del contador de programa y ejecuta la función asociada a la interrupción al ser considera más prioritaria, ignorando el resto de tareas. Cuando finaliza la función que ha saltado con la interrupción, el procesador vuelve al código que estaba ejecutando en el momento de la interrupción, restaurando el estado que había guardado en el procesador y en la pila.

Arduino dispone de varios tipos de eventos capaces de producir interrupciones. Podemos citar las más utilizadas, como son las interrupciones de hardware que tiene lugar en eventos ocurridos en los pines físicos, y por otro lado, se encuentran las interrupciones asociadas a los timers que son las que nos interesan en este trabajo.

En cuanto a los pines que generan interrupciones, estos varían dependiendo de la versión utilizada de Arduino. En nuestro caso se disponen de dos interrupciones asociadas a los pines 2 y 3.

Entramos ahora a resumir el funcionamiento de las interrupciones asociadas a los timers de Arduino. Como se ha comentado, podemos completar el resumen aquí mostrado acudiendo a [4] aunque la versión Arduino sobre la que se basa dicho trabajo es Arduino Uno, nuestra versión Arduino Nano puede considerarse idéntica en los términos que aquí se refieren.

Para mayor indagación en cuanto a interrupciones en Arduino, se puede acudir a [17]

Interrupciones internas de Arduino

Este tipo de interrupciones en Arduino son denominadas interrupciones de eventos programados y están íntimamente relacionas con los timers. Un timer puede considerarse como un temporizador periódico que salta cada cierto tiempo establecido previamente.

Arduino Nano cuenta con 3 timers. El timer 0 y el timer 2 de 8 bits y el timer 1 de 16 bits.

Las interrupciones que podemos llevar a cabo serán de 3 tipos.

- Interrupciones por desbordamiento del temporizador (Timer Overflow).
- Interrupciones por comparación de salida (Output Compare Match Interrupt)
- Interrupciones por captura de entrada (Timer Imput Capture Interrupt)

Modos de operación de los Timers

Acudiendo al Datasheet del microcontrolador Atmega328p incrustado en nuestra placa Arduino nano, podemos apreciar que se describen 4 modos de funcionamiento asociados a los Timers. Cabe recordar en este punto que la frecuencia de operación de nuestra placa es de 16 MHz.

- Modo normal: catalogado como el modo más simple y versátil. Consiste en incrementar el contador hasta que llega a su valor máximo, reiniciándose y volviendo al conteo. Se va modificando el valor de OCRxA/B, de forma que cuando el valor de TCNTx alcanza el valor de OCRxA/B salta la interrupción correspondiente. Será el método elegido en este trabajo. Podemos destacar 3 funciones asociadas a este modo:
  - $ISR(TIMERx_COMPA_vet)$: Función que salta cuando el contador alcanza el valor del registro A, es decir, TCNTx = OCRxA.
  - $ISR(TIMERx_COMPB_vet)$: Función que salta cuando el contador alcanza el valor del registro B, es decir, TCNTx = OCRxB.
  - $ISR(TIMERx_COMPA_vet)$: Función que salta cuando el contador alcanza el valor máximo, se produce un desvordamiento. El valor máximo será 255 para los timer 0 y 2 y 65535 para el timer 1.

  En este punto, es conveniente indicar que el contador TCNTx se incrementa en una unidad cada vez que ocurre un numero fijo de pulsos de reloj. Este número de pulsos viene determinado por el preescalador elegido. A modo de ejemplo, si se usa el reloj del sistema sin divisor, es decir a preescalado 1, el registro TCNTx se incrementa en uno cada 0.0625 us (1/16MHz), llegando a desbordarse en 16 us para el timer 0 y 2, y en 4 ms para el timer 1.
- Modo PWM rápido: El contador se incrementa hasta que se llega al valor superior OCRnA o OCRnB.
- Modo PWM en fase correcta: El contador se incrementa hasta que llega al valor superior y posteriormente se decrementa.
- Modo CTC: El contador se resetea cuando el valor de TCNTn se iguala con OCRnA o OCRnB .

Como vemos, aparecen dos términos desconocidos hasta ahora como son OCRnA y TCNTn. Para comprender dichos términos, acudimos a [4] y citamos textualmente:

Una vez se tiene claro qué es y cómo modificar un registro de desplazamiento, se presentan los registros de los timers que se usarán en este proyecto, con una breve explicación que se puede complementar con los apartados posteriores, los cuales se indican. El carácter "x" se refiere al timer, que puede ser 0,1 o 2:

- *TCNTx: Valor del contador asociado al timer, Es un entero comprendido entre 0 y 255 para el timer 0 y 2, y entre 0 y 65535 para el timer 1. Cada flanco de reloj lo incrementan una unidad, de forma que este valor sirve de referencia para referencias temporales. Su valor vuelve a ser 0 cuando alcanza el máximo o cuando se resetea. Más informacioón en el apartado "4.2.2 Características principales".*
- *OCRxA y OCRxB: Valores para que salten las interrupciones A o B, respectivamente. Cuando TCNTx se iguale a alguna de estas variables, se activará la interrupción correspondiente. En modo normal, si están activadas, se ejecutará la función de interrupción. En modo CTC sirven para resetear el valor del contador cuando se iguala a cualquiera de las dos, en el PWM para activar y desactivar la señal de salida, etc. Más información en los apartados "4.2.3 Modos de temporización" y "4.2.4 Interrupciones Timers".*
- *TIMSKx: Se denomina máscara del timer. Sirve para activar o desactivar las interrupciones asociadas. Sus posiciones OCIExA y OCIExB activan y desactivan las interrupciones A y B del timer, las cuales se producen cuando TCNTx = OCIExA/B. La posición TOIEx activa y desactiva la interrupción por OverFlow del timer. Más información en el apartado "4.2.4 Interrupciones Timers".*

- *TCCRxA y TCCRxB: Permiten variar las características de los timers. Según las posiciones a las que se le varía el valor se modificará una característica u otra:*
  - *CSx0, CSx1 y CSx2: Posiciones de los registros TCCRxA/B para el prescalado del timer, de forma que TCNTx se incrementa cada tantos flancos de reloj como prescalado haya especificado. Más información en el apartado "4.2.2.1 Prescalado Timers".*
  - *WGMx3, WGMx2, WGMx1 y WGMx0: Posiciones de los registros TCCRxA/B para cambiar su modo de funcionamiento. Más información en el apartado "4.2.3 Modos de temporización".*

ISR

Como ya se ha mencionado, la función ISR estará asociada al cumplimento de un temporizador. Dicha función presenta una serie de requisitos que la diferencian del resto de funciones conocidas.

Las funciones ISR presentan una serie de limitaciones, pues no puede contener ningún parámetro de llamada ni devolver ninguna función. En caso de llamar a una ISR mientras se esta ejecutando otra, quedará bloqueada hasta que finalice la que se esta ejecutando, es decir, no pueden ejecutarse dos ISR de forma simultánea. Además, cada ISR debe ser lo más concisa posible, pues cuando se esta ejecutando se bloquea el resto del código.

5.1.2 Preescalado

Con el preescalado de los Timers se consigue reducir la frecuencia y por ello aumentar el periodo de salto de una interrupción.

Con Arduino Nano, podemos aplicar diferentes escalados como pueden ser 1,8,64,256 y 1024. En este punto, cabe recordar que la frecuencia de trabajo de nuestro procesador es de 16MHz lo que quiere decir que sin aplicar el preescalado, el timer 0 y 2 produciría interrupciones por Timer Overflow (Por desbordamiento) cada 15.94 μs, mientras que el timer 1, produciría cada 4.1 ms. Los datos anteriores pueden deducirse partiendo de los siguientes cálculos:

$$T = T = \frac{1}{f} = \frac{1}{16Mhz} = 62.5ns$$

$$Timer\ 0,\ 2\ = 62.5ns * (2^8 - 1) = 62.5ns * 255 = 15.94\mu s$$

$$Timer\ 1\ = 62.5ns * (2^{16} - 1) = 62.5ns * 65535 = 4.1ms$$

El código Arduino que se encargará de modificar el Preescalado será el siguiente, en él puede apreciarse los valores de preescalado del Timer 1 y del Timer 2, que se obtienen por multiplicación de los cálculos anteriores con el valor del preescalado:

```
1 // //CONTINUA////
2 //Tiempo de pulso de cada timer
3 float T_PULS_1=0.0000000625*PREESCALA_1; //T=(1/16000000)*65535*Escala
4 float T_PULS_2=0.0000159375*PREESCALA_2; //T=(1/16000000)*255*Escala
5 //Funciones que modifican los registros
6 #define CLR(x, y) (x&=(~(1<<y)))
7 #define CLRR(x, y, z) (x&=(~((1<<y)|(1<<z))))
8 #define SET(x, y) (x|=(1<<y))
9 #define SETT(x, y, z) (x|=(1<<y)|(1<<z))
10 #define SETTT(x, y, z, v) (x|=(1<<y)|(1<<z)|(1<<v))
11 // //CONTINUA////
12 void setup() {
13 // //CONTINUA////
14 // Configuración de las interrupciones del timer 1
```

```
15 TCCR1A=0; //Iniciamos en modo normal
16 TCCR1B=0; //Iniciamos en modo normal
17 switch (PREESCALA_1){
18   case 1:                    // 4.095ms
19   SET(TCCR1B,CS10);
20   break;
21   case 8:                    // 32.7675ms
22   SET(TCCR1B,CS11);
23   break;
24   case 64:                   // 0.2621s
25   SETT(TCCR1B,CS11,CS10);
26   break;
27   case 256:                  // 1.048s
28   SETT(TCCR1B,WGM12,CS12);
29   break;
30   case 1024:                 // 4.1942S
31   SETT(TCCR1B,CS12,CS10);
32   break;
33   dir_M1=0;
34   dir_M2=0;
35 };
36 OCR1A =65535; //Iniciamos en modo normal
37 OCR1B =65535; //Iniciamos en modo normal
38 TIMSK1 =0;     // Limpia la máscara del timer
39 SETT(TIMSK1,OCIE1A,OCIE1B); //Activas las interrupciones A y B
40 // Configuración de las interrupciones del timer 2
41 TCCR2A=0;    // Iniciamos en modo normal
42 TCCR2B=0;    // Iniciamos en modo normal
43 TCNT2=0;
44 switch (PREESCALA_2){
45   case 1:                   // 19.94us
46   SET(TCCR2B,CS20);
47   break;
48   case 8:                   // 127us
49   SET(TCCR2B,CS21);
50   break;
51    case 32:                 // 510us
52   SETT(TCCR2B,CS21,CS20);
53   break;
54   case 64:                  //  1.02ms
55   SET(TCCR2B,CS22);
56   break;
57   case 128:                 // 2.04ms
58   SETT(TCCR2B,CS20,CS22);
59   break;
60   case 256:                 // 4.08ms
61   SETT(TCCR2B,CS21,CS22);
62   break;
63   case 1024:                // 16.32ms
64   SETTT(TCCR2B,CS20,CS21,CS22);
65   break;
66 };
67 OCR2A=250;
68 SET(TCCR2A,WGM21);
69 SET(TIMSK2,OCIE2A); //Activa la interrupción A del timer
70 // //CONTINUA////
71 }
```

5.1.3 Obtención de la aceleración

Para la obtención de la aceleración se parte del código desarrollado en [4] con ligeras modificaciones.

Implementación de la velocidad

Para implementar la velocidad, se hará uso de las interrupciones del Timer 1:

```
// //CONTINUA////
//     ///////////////     INTERRUPCIÓN A DEL TIMER 1 PARA MOVER EL MOTOR 1//////////////////////
ISR(TIMER1_COMPA_vect)
{
// Activa la interrupci ón A del Timer para dar velocidad al motor 1
if (dir_M1==0) // si el motor no se encuentra girando en ningún sentido (condición inicial), el motor
      no se moverá
return ;
SET(PORTD,STEP_MOTOR_1); //activa el pin STEEP
OCR1A+=npuls_M1;              // mantiene el pin en alto durante el tiempo que le inidica npuls_M1
CLR(PORTD,STEP_MOTOR_1); //desactiva el pin STEEP
}
//     ///////////////     INTERRUPCION B DEL TIMER 1 PARA MOVER EL MOTOR 2//////////////////////
   ISR(TIMER1_COMPB_vect)
{
if (dir_M2==0)
return ;
SET(PORTD,STEP_MOTOR_2);
OCR1B+=npuls_M2;
CLR(PORTD,STEP_MOTOR_2);
}
// //CONTINUA////
```

Actualización de la aceleración

Para actualizar la velocidad e implementar la nueva aceleración se hará uso de las interrupciones del Timer2. El código siguiente actualiza la velocidad cada 4 ms, que es el tiempo con el que se ha programado las interrupciones para que salte el Timer 2, e implementa la nueva aceleración cada 20 ms. Arduino será el encargado de actualizar la velocidad en cada salto del timer 2, con el dato de la aceleración que vendrá determinado por la Rpi que actualizará dicho dato cada 20ms, es decir cada 5 saltos del Timer 2 se actualiza el valor de la aceleración.

Aparece una bandera *flag* que se pone a cero cuando se recibe una nueva aceleración de forma que este dato recibido se procese una única vez. Es necesario introducir esta bandera porque recordemos que la velocidad se implementa cada 4 ms y la aceleración se actualiza cada 20 ms. Además, en la práctica se comprobó que de vez en cuando el dato que recibía de la RPi se desbordaba, dando un valor demasiado alto o un número no real. Por ello, en el caso de que el dato de la aceleración llegue corrupto, se tomará el dato anterior evitando así que el vehículo se descontrole.

```
// //CONTINUA////
ISR(TIMER2_COMPA_vect)
{
 interrupts ();     //Función que evita que el programa se colapse

if (flag ==0)
{
flag =1;          //Se pone la bandera a 1 para trabajar con la aceleraci ón recibida y no volver a
      cambiarla hasta que se vuelve a recibir un nuevo dato.
comprueba=isnan(alfa); //Comprueba que el número que se recibe es real, en caso contrario devuelve 1.
if (alfa >150 || alfa <-150 || comprueba==1) // Filtro para obviar los valores de aceleraci ón recibidos
      muy grandes o no reales
{
   alfa_M1=alfa_M1; //En caso de recibir un dato corrupto, mantengo la aceleraci ón anterior.
}
else
```

```
15 {
16   alfa_M1=alfa;     // En caso de recibir un dato dentro del rango, actualizo con la aceleración recibida
                .
17 }
18 }
19
20 vel_M1=vel_M1+alfa_M1*T_PULS_2+giro_M1*0.05; //Actualizo la velocidad del motor 1 cada T_PULS_2 MS
21 vel_M2=vel_M2-alfa_M1*T_PULS_2+giro_M1*0.05; //Actualizo la velocidad del motor 1 cada T_PULS_2 MS
22 vel_M1=constrain(vel_M1 ,-60,60);    // Saturo la velocidad entre los límites alcanzables
23 vel_M2=constrain(vel_M2 ,-60,60);    // Saturo la velocidad entre los límites alcanzables
24 comprueba_vel_1(vel_M1); //Compruebo la dirección de giro del motor 1 con la nueva velocidad
25 comprueba_vel_2(vel_M2); //Compruebo la dirección de giro del motor 2 con la nueva velocidad
26 vel_M1-=giro_M1*0.05;
27 vel_M2-=giro_M1*0.05;
28 vec[0]=(vel_M1-vel_M2)/2.0; //Calculo el valor de la velocidad media de la rueda que se envía a la RPi
29 }
30 // //CONTINUA////
```

Comprobación de la dirección de giro

Se muestra a continuación la función que comprueba el giro del motor 1 y actualiza el valor que debe estar en alto el pin STEP para implementar la velocidad. La función que comprueba la dirección del motor 2 es análoga a la siguiente y por ello no se muestra.

```
1  // //CONTINUA////
2  void comprueba_vel_1(float vel)
3  {
4  if (vel >0)
5  {
6    if(dir_M1 !=1)
7    {
8    SET(PORTD,DIR_MOTOR_1);
9    dir_M1=1;
10   }
11   if (vel<=omega_motor_max[3] && vel>=omega_motor_min[3])
12   {
13     SETT(PORTB, MS1_MOTOR_1, MS2_MOTOR_1);
14     npuls_M1=omega2npuls[3]/vel;
15   }
16   else if (vel<=omega_motor_max[2]&& vel>=omega_motor_min[2])
17   {
18     CLR(PORTB,MS1_MOTOR_1);
19     SET(PORTB,MS2_MOTOR_1);
20     npuls_M1=omega2npuls[2]/vel;
21   }
22   else if (vel<=omega_motor_max[1]&& vel>=omega_motor_min[1])
23   {
24     CLR(PORTB,MS2_MOTOR_1);
25     SET(PORTB,MS1_MOTOR_1);
26     npuls_M1=omega2npuls[1]/vel;
27   }
28   else if (vel<=omega_motor_max[0]&& vel>=omega_motor_min[0])
29   {
30     CLRR(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
31     npuls_M1=omega2npuls[0]/vel;
32   }
33   else
34   {
35     SETT(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
36     npuls_M1=npuls_max[3];
37   }
```

```
38 }
39 if (vel <0)
40 {
41   if (dir_M1 !=-1)
42   {
43   CLR(PORTD,DIR_MOTOR_1);
44   dir_M1=-1;
45   }
46   if (-vel<=omega_motor_max[3] && -vel>=omega_motor_min[3])
47   {
48     SETT(PORTB, MS1_MOTOR_1, MS2_MOTOR_1);
49     npuls_M1=-omega2npuls[3]/vel;
50   }
51   else if (-vel<=omega_motor_max[2]&& -vel>=omega_motor_min[2])
52   {
53     CLR(PORTB,MS1_MOTOR_1);
54     SET(PORTB,MS2_MOTOR_1);
55     npuls_M1=-omega2npuls[2]/vel;
56   }
57   else if (-vel<=omega_motor_max[1]&& -vel>=omega_motor_min[1])
58   {
59     CLR(PORTB,MS2_MOTOR_1);
60     SET(PORTB,MS1_MOTOR_1);
61     npuls_M1=-omega2npuls[1]/vel;
62   }
63   else if (-vel<=omega_motor_max[0]&& -vel>=omega_motor_min[0])
64   {
65     CLRR(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
66     npuls_M1=-omega2npuls[0]/vel;
67   }
68   else
69   {
70     SETT(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
71     npuls_M1=npuls_max[3];
72   }
73 }
74 }
75 // //CONTINUA////
```

5.2 Obtención del Ángulo

Sin duda uno de los puntos claves del proyecto radica en la obtención del ángulo del vehículo con su vertical. Se pretende en todo momento que este ángulo de inclinación con respecto al eje vertical, que será su eje de equilibrio, sea cero. Ante cualquier perturbación el vehículo debe buscar esa posición.

Como vimos en el apartado *"2.5 MPU6050"*, el dispositivo encargado de controlar en todo momento el ángulo de inclinación será el dispositivo GY-521, el cual cuenta con un giroscopio y un acelerómetro ambos de 3 ejes.

Para el cálculo de la acción de control será necesario el conocimiento del vector de estados. Dos de los parámetros que conforman el vector de estados del sistema van a ser obtenidos con este dispositivo. Seremos capaces de conocer tanto el ángulo de inclinación (componente ϕ del vector de estados) como la velocidad de caída del vehículo (componente $\dot{\phi}$ del vector de estados).

El mencionado dispositivo, irá conectado a la RPi y su comunicación con esta será a través del protocolo de comunicación I2C. El dispositivo cuenta con dos direcciones, la 0x68 y la 0x69. Por defecto la comunicación será a través de la primera dirección, sin embargo, si queremos comunicarnos por la segunda de ellas será necesario conectar el pin ADO a 5V.

Del funcionamiento del dispositivo, que se detalló en su apartado correspondiente, resumiremos los conocimientos que debemos de aplicar al código para la obtención del ángulo.

Aunque el dispositivo cuenta con 6 grados de libertad, cabe recordar que en nuestro proyecto solamente será necesario el uso de 3 de ellos. Dos componentes del acelerómetro y una del giroscopio.

El dispositivo se encuentra acoplado a la PCB y girado 90º con respecto al eje x. Por tanto, la orientación que siguen sus ejes se ve en el siguiente esquema, siendo el eje z el que se encuentra paralelo al suelo.

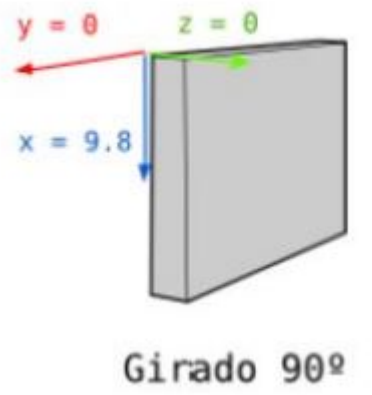


Figura 5.2 Orientación de la IMU en la PCB.

5.2.1 Programación

La RPi desde su lanzamiento en 2012 ha gozado de una gran popularidad y aceptación para el desarrollo de actividades robóticas. Presenta la posibilidad de programar en diferentes lenguajes como pueden ser, Python, C, C++, java, ADA…

La multitud de lenguajes de programación disponibles, así como el poco tiempo en el mercado, hacen difícil encontrar bibliotecas específicas para controlar dispositivos externos en un lenguaje de programación específico.

En el presente capítulo, se esta indagando en la necesidad de adicionar un dispositivo capaz de medir la inclinación del robot en cada momento. El lenguaje de programación por el que nos hemos decantado es C, ante la falta de una biblioteca en C para dicho dispositivo, estudiando su funcionamiento, se ha procedido a su creación.

A continuación, se va a proceder a la explicación de las partes más importantes del código, el cual se encuentra incluido en el archivo adjunto **MPU6050.h**.

- **void inicio (int addr)**: Función que recibe como argumento de entrada la dirección I2C del dispositivo, que por defecto será 0x68.

```
1 void inicio (int addr)
2 {
3 int status;
4
5 MPU6050_addr = addr; //guarda la dirección recibida en la
    variable MPU6050_addr
```

```
6
7   //abre la comuicacion i2c
8   f_dev = open("/dev/i2c-1", O_RDWR);
9     if (f_dev < 0) {
10  printf("error abriendo la conexión");
11    }
12  status = ioctl(f_dev, I2C_SLAVE, MPU6050_addr);
13  if (status < 0) {
14  printf("error");
15    }
16
17    i2c_smbus_write_byte_data(f_dev, 0x6b, 0b00000000); //
        desabilita el modo sleep del dispositivo
18
19    i2c_smbus_write_byte_data(f_dev, 0x1a, 0b00000011); //ajusta un
         filtro paso bajo a 44Hz
20
21    i2c_smbus_write_byte_data(f_dev, 0x19, 0b00000100); //Establece
         el divisor de frecuencia de muestreo
22
23    i2c_smbus_write_byte_data(f_dev, 0x1b, GYRO_CONFIG); //
        configura los agustes del giroscopio
24
25    i2c_smbus_write_byte_data(f_dev, 0x1c, ACCEL_CONFIG); //
        configura los ajustes del acelerómetro
26
27
28  i2c_smbus_write_byte_data(f_dev, 0x06, 0b00000000),
      i2c_smbus_write_byte_data(f_dev, 0x07, 0b00000000),
      i2c_smbus_write_byte_data(f_dev, 0x08, 0b00000000),
      i2c_smbus_write_byte_data(f_dev, 0x09, 0b00000000),
      i2c_smbus_write_byte_data(f_dev, 0x0A, 0b00000000),
      i2c_smbus_write_byte_data(f_dev, 0x0B, 0b00000000),
      i2c_smbus_write_byte_data(f_dev, 0x00, 0b10000001),
      i2c_smbus_write_byte_data(f_dev, 0x01, 0b00000001),
      i2c_smbus_write_byte_data(f_dev, 0x02, 0b10000001);
29
30  }
```

- **Void getAccelRaw(float *x, float y*, float *z)**: Función que devuelve los valores en bruto del acelerómetro. Recibe como argumento de entrada un puntero a los valores que devuelve. Tiene un rango de medición comprendido entre -32768 y +32767.

```
1   //obtiene los valores en bruto del acelerómetro
2      void getAccelRaw(float *x, float *y, float *z)
3   {
4      int16_t X = i2c_smbus_read_byte_data(f_dev, 0x3b) << 8 |
          i2c_smbus_read_byte_data(f_dev, 0x3c); //Lee el registro X
           de la aceleración
5     int16_t Y = i2c_smbus_read_byte_data(f_dev, 0x3d) << 8 |
          i2c_smbus_read_byte_data(f_dev, 0x3e); //Lee el registro Y
          de la aceleración
6     int16_t Z = i2c_smbus_read_byte_data(f_dev, 0x3f) << 8 |
          i2c_smbus_read_byte_data(f_dev, 0x40); //Lee el registro Z
          de la aceleración
7     *x = (float)X;
```

```
8     *y = (float)Y;
9     *z = (float)Z;
10  }
```

- **Void getGyroRaw (float *roll, float *pitch, float *yaw)**: Función que devuelve los valores en bruto del giroscopio. Recibe como argumento de entrada un puntero a los valores que devuelve. Tiene un rango de medición comprendido entre -32768 y +32767.

```
1   //obtiene los valores en bruto del giroscopio
2      void getGyroRaw(float *roll, float *pitch, float *yaw)
3   {
4     int16_t X = i2c_smbus_read_byte_data(f_dev, 0x43) << 8 |
         i2c_smbus_read_byte_data(f_dev, 0x44); //Lee el registro X
         del giroscopio
5     int16_t Y = i2c_smbus_read_byte_data(f_dev, 0x45) << 8 |
         i2c_smbus_read_byte_data(f_dev, 0x46); //Lee el registro Y
         del giroscopio
6     int16_t Z = i2c_smbus_read_byte_data(f_dev, 0x47) << 8 |
         i2c_smbus_read_byte_data(f_dev, 0x48); //Lee el registro Z
         del giroscopio
7     *roll = (float)X;
8     *pitch = (float)Y;
9     *yaw = (float)Z;
10
11  }
```

- **Void getAccel(float *x, float y*, float *z)**: Función que devuelve los valores en g del acelerómetro con 3 cifras decimales. Recibe como argumento de entrada un puntero a los valores que devuelve. Dicha función primero calcula los valores brutos con **getAccelRaw** y posteriormente aplica el factor de sensibilidad.

```
1   //obtiene los valores del acelerómetro en g con 3 cifras
       decimales
2      void getAccel(float *x, float *y, float *z)
3   {
4     getAccelRaw(x, y, z);
5     *x = round((*x - A_OFF_X) * 1000.0 / ACCEL_SENS) / 1000.0;
6     *y = round((*y - A_OFF_Y) * 1000.0 / ACCEL_SENS) / 1000.0;
7     *z = round((*z - A_OFF_Z) * 1000.0 / ACCEL_SENS) / 1000.0;
8   }
```

- **Void getGyro (float *roll, float *pitch, float *yaw)**: Función que devuelve los valores en g/s del giroscopio con 3 cifras decimales. Recibe como argumento de entrada un puntero a los valores que devuelve. Dicha función primero calcula los valores brutos con **getGyroRaw** y posteriormente aplica el factor de sensibilidad.

```
1   //obtiene los valores del giroscopio en g/s con 3 cifras
       decimales
2   void getGyro(float *roll, float *pitch, float *yaw)
3   {
4       getGyroRaw(roll, pitch, yaw);
5     *roll = round((*roll - G_OFF_X) * 1000.0 / GYRO_SENS) / 1000.0;
6     *pitch = round((*pitch - G_OFF_Y) * 1000.0 / GYRO_SENS) /
         1000.0;
```

```
7     *yaw = round((*yaw - G_OFF_Z) * 1000.0 / GYRO_SENS) / 1000.0;
8   }
```

- **void getOffsets(float *ax_off, float *ay_off, float *az_off, float *gr_off, float *gp_off, float *gy_off)**: Función que calcula los valores del offsets que posteriormente tendremos que añadir al mismo fichero "MPU6050.h". Para calcular el offset se realiza un bucle for de 10.000 iteraciones calculando los datos en bruto. Finalmente realiza la media de los datos leídos.

```
1   //calcula el offset
2      void getOffsets(float *ax_off, float *ay_off, float *az_off,
           float *gr_off, float *gp_off, float *gy_off)
3   {
4     float gyro_off[3];
5     float accel_off[3];
6   float suma_x=0, suma_y=0, suma_z=0, suma_gx=0, suma_gy=0,
        suma_gz=0;
7
8
9     for (int i = 0; i < 10000; i++) {
10      getGyroRaw(&gyro_off[0], &gyro_off[1], &gyro_off[2]); //
11      suma_gx=suma_gx+ gyro_off[0], suma_gy=suma_gy+ gyro_off[1],
            suma_gz=suma_gz+ gyro_off[2];
12
13      getAccelRaw(&accel_off[0], &accel_off[1], &accel_off[2]);
14      suma_x=suma_x+accel_off[0], suma_y=suma_y+accel_off[1],
            suma_z=suma_z+ accel_off[2];
15    }
16
17    *gr_off = suma_gx / 10000, *gp_off = suma_gy / 10000, *gy_off =
          suma_gz / 10000;
18    *ax_off = suma_x / 10000, *ay_off = suma_y / 10000, *az_off =
         suma_z / 10000;
19
20    *az_off = *az_off - ACCEL_SENS; /
21  }
```

Para el cálculo del offset, se programó una función a parte que, tras su ejecución, nos dio los siguientes valores.

- A_OFF_X =8977.75
- A_OFF_Y =1870.6
- A_OFF_Z =-16697.05
- G_OFF_X =-3.55
- G_OFF_Y =0.65
- G_OFF_Z =3.7

5.2.2 Cálculo del ángulo

Para la obtención del ángulo por medio del acelerómetro, tendremos que utilizar la siguiente ecuación trigonométrica, donde ax, ay y az son los valores de la aceleración medidos en dichos ejes:

$$Angulo = atan\left(\frac{az}{\sqrt{ay^2 + ax^2}}\right) \tag{5.1}$$

Como se vio en su apartado correspondiente, el valor que se obtiene no es del todo fiable pues introduce un exceso de ruido en la medida lo cual hace inviable la utilización de este dato para el cálculo del ángulo.

El cálculo del ángulo mediante el uso del giroscopio puede calcularse como:

$$Angulo = Angulo_{anterior} + (Giroscopio * \triangle t) \tag{5.2}$$

Por lo general, las medidas obtenidas por el giroscopio suelen ser más precisas que las calculadas por el acelerómetro, en un corto periodo de tiempo. Sin embargo, en un largo periodo de tiempo, al calcularse el dato como una suma de incrementos, hará que se aleje de la realidad debido a una suma de los errores acumulados en la medida.

Por todo ello, es necesario la adición de un filtro complementario, cuya explicación puede verse en el apartado "2.5.7 Filtro Complementario", que compense los errores citados y nos proporcione un valor fiable. Por tanto, la idea básica consiste en combinar ambas salidas ponderando el peso de cada una ellas. Se ha elegido un valor de $\tau = 0.99$. La función que se encarga de calcular dichos valores será:

```
1
2  void calculo_angulo(void)
3  {
4     getGyro(&gx_escalado, &gy_escalado, &gz_escalado);
5     getAccel(&ax_escalado, &ay_escalado, &az_escalado); //lee valores
          en el sistema internacional g
6     accel_ang = atan2(az_escalado , (sqrt((ax_escalado*ax_escalado) +
          (ay_escalado*ay_escalado))))*(180/3.1415); //Cálculo del á
          ngulo por el acelerómetro. Datos en grados.
7     giros_ang = filtro (accel_ang, gy_escalado);
8
9  }
```

En el código anterior se puede apreciar que aparece la función **filtro()**, la cuál se encarga de calcular el filtro complementario y el tiempo que transcurre entre cada medida, medido en ms. Como argumento de entrada recibirá el ángulo medido por el acelerómetro así como la velocidad angular escalada del eje y.

```
1
2  float filtro(float accel, float gxy)
3   {
4   dtt=(double)(clock()-iniciall)/CLOCKS_PER_SEC;
5  giros_ang_y_prev=TAU*(giros_ang_y_prev+dtt*gxy)+(1-TAU)*accel;
6  iniciall=clock();
7  return (giros_ang_y_prev);
8  }
```

En la siguiente gráficas podemos apreciar la necesidad del uso del filtro complementario. Se realiza una comparativa del ángulo obtenido con el acelerómetro (gráfica azul) donde se aprecia el excesivo ruido, y el ángulo calculado aplicando el filtro (gráfica roja) donde se aprecia que las medidas son más suaves y por lo tanto de mayor validez.

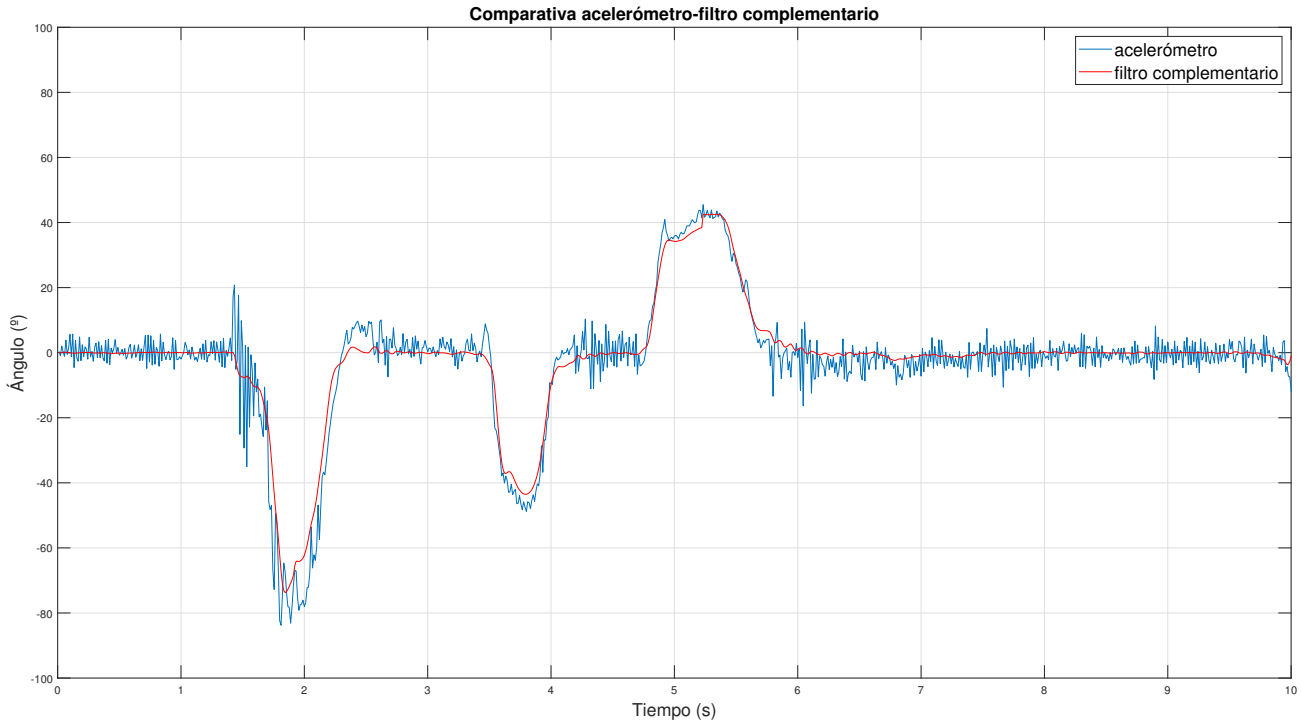


Figura 5.3 Comparativa del ángulo calculado mediante el acelerómetro y el filtro complementario.

5.3 Programación RPi

En este punto de capítulo, ya conocemos como obtener las tres componentes que forman el vector de estados. Mediante Arduino conocemos la velocidad de las ruedas ($\dot{\theta}$), y mediante la IMU conocemos el ángulo de inclinación (ϕ) y la velocidad del vehículo ($\dot{\phi}$). Es necesario conocer en este instante como calcular la señal de control, que será la aceleración de las ruedas, y como comunicarnos con Arduino para el envío de dicha señal de control.

La adición de la RPi a este proyecto abre un nuevo mundo de posibilidades, quedando atrás las limitaciones temporales de Arduino en cuanto a cálculos complejos. La frecuencia del procesador que incorpora la RPi es de 1.2 GHz, siendo sumamente superior a Arduino (16MHz), dando la posibilidad de programar todo tipo de controladores, abriendo numerosas líneas de investigación para trabajas futuros.

Dentro de la cantidad de lenguajes con los que podemos trabajar en la RPi, se ha optado por C. La RPi se encuentra especialmente diseñada para trabajar con python. Sin embargo, phyton es un lenguaje interpretado a diferencia de C que es compilado. Los costes computacionales de python son muy superiores a los alcanzados con C y como nuestro sistema requiere de una respuesta rápida, especialmente en el control MPC, no queda garantizado que se puedan llevar a cabo las acciones.

A continuación se va a proceder a explicar la parte común de código que comparten todos los controladores, para posteriormente centrarnos en la parte que difiere. La parte que difiere se

encuentra dentro de una función manejador que se encargará de calcular la acción de control.

Para que se produzca el salto del manejador se ha programado un temporizador cíclico aplicando las normas POSIX, al que se le ha asociado una señal, de tal manera que cuando se cumple el temporizador salta la señal y por consiguiente salta el manejador. La función manejador se encargará del envío y recepción de datos con Arduino, del cálculo de los valores de la IMU y del cálculo de la acción de control.

Con la aplicación del parche Xenomai que convierte nuestro sistema operativo en un sistema en tiempo real, queda garantizado el cumplimiento de los plazos, garantizando que el salto de la señal tenga lugar cuando se le especifica. En caso de utilizar python, no queda garantizado el cumplimiento de los plazos.

```
1  //DECLARACIÓN DE CABECERAS
2  #include "MPU6050.h" //dentro de dicha libreria se declaran en resto
       de librerías necesarias
3  #include <signal.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include "ARDUINO.h"
7  #include <stdbool.h>
8  //VARIABLES PARA EL MPU
9  float ax,ay,az, ax_escalado, ay_escalado, az_escalado;
10 float accel_ang_y;
11 float gx,gy,gz, gx_escalado, gy_escalado, gz_escalado;
12 float giros_ang_y, giros_ang_y_prev=3;
13 float TAU=0.99;
14
15 //VARIABLES PARA COMUNICAR CON ARDUINO
16 float omega, omega_r;
17 float alpha;
18 //VARIABLES PARA EL TEMPORIZADOR
19 timer_t temporizador; //Declaración del nombre del temporizador
20 struct itimerspec periodo;//Estructura para indicar cada cuanto
       tiempo se lanza el temporizador
21 struct timespec inicial;
22 struct timespec tiempo; //Estructura para definirla duración del
       temporizador
23 struct sigevent aviso; //Establezco una señal que avise que se ha
       cumplido el temporizador
24 sigset_t sigset;
25 struct sigaction act;
26 int flag=0; //variable que se inicializa a cero para comenzar la
       cuenta del primer cálculo del filtro complementario
27 void Manejador(int signo, siginfo_t *info,
28  void*context)
29 {
30 ////////DEPENDE DEL CONTROLADOR////////
31 }
32 //Función principal
33 int main(void )
34 {
35 //Inicio de conexión con Arduino y con el MPU-6050
36 inicio_mpu(0x68);
37 inicio_arduino(0x05);
38 //inicia la cuenta para calcular el primer dato del filtro
       complementario
```

```
39 if (flag==0){
40 flag=1;
41 iniciall=clock();
42 }
43 //TRATAMIENTO DE SENALES
44 sigemptyset(&sigset); //crea una mascara vacía
45 sigaddset(&sigset, SIGUSR1); //anade la señal SIGALARM al conjunto
46 pthread_sigmask(SIG_UNBLOCK,&sigset, NULL); //bloqueo la señal para
       el proceso
47 /* programo la señar SIGVTALRM para que salte la funcion manejador
       cuando se cumpla el temporizador */
48 act.sa_sigaction=Manejador;
49 sigemptyset(&(act.sa_mask));
50 act.sa_flags=SA_SIGINFO;
51 sigaction(SIGUSR1, &act, NULL);
52 //configuro el temporizador
53 inicial.tv_sec=1;
54 tiempo.tv_nsec=XX; //cada XX ms salta el manejador
55 periodo.it_value=inicial; //Indico que el primer valor lo lance 1
       segundo despues de llamarlo
56 periodo.it_interval=tiempo; //Indico que los peridos sean cada XX ms
57 aviso.sigev_notify=SIGEV_SIGNAL;
58 aviso.sigev_signo=SIGUSR1;
59 timer_create(CLOCK_REALTIME, &aviso, &temporizador);//creo el
       temporizador llamado tempo, de tiempo real y que avise con la señ
       al configurada en el segundo argumento de entrada
60 //inicio el temporizador
61 timer_settime(&temporizador, TIMER_ABSTIME, &periodo , NULL);
62 while(1){}
63 exit(0);
64 }
```

5.3.1 Programación LQR y LQRI

Para la programación del control LQR y LQRI, se ha diseñado un ejecutivo cíclico dentro de la función manejador, de tal manera que se organiza las tareas de forma más clara. Podría haberse prescindido de dicho ejecutivo como se hace con el control MPC.

El funcionamiento es el siguiente:

El manejador salta cada 4 ms, en cada salto incrementa una variable entera y con un switch-case realiza el bloque que le corresponde. Se ha comprobado el coste temporal de cada acción y en ningún caso supera 1 ms, no se produce por tanto interferencia, siendo 4 ms tiempo suficiente para realizar la tarea encomendada. En el último bloque, se resetea la variable que permite el paso de un tarea a la siguiente.

```
1  //FUNCIÓN MANEJADOR
2  void Manejador(int signo, siginfo_t *info, void*context)
3  {
4  ciclo++;
5   //CADA VEZ QUE SE ACTIVA LA SENAL, CICLO AUMENTA EN UNA UNIDAD Y
       HACE LO QUE LE CORRESPONDA EN EL SWITCH-CASE
6  switch(ciclo)
7  {
8  case 1 :
9  getGyro(&gx_escalado, &gy_escalado, &gz_escalado);
10 getAccel(&ax_escalado, &ay_escalado, &az_escalado);
```

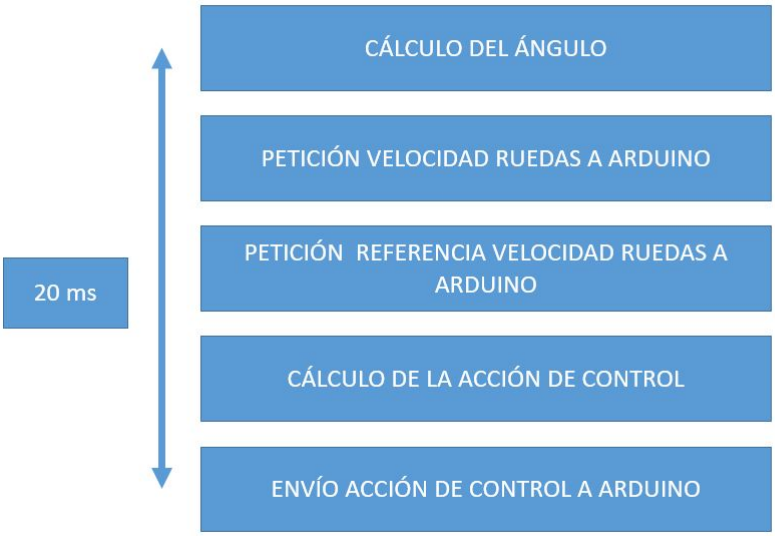


Figura 5.4 Ejecutivo ciclico LQR y LQRI.

```
11  accel_ang_y = atan2(az_escalado , (sqrt((ay_escalado*ay_escalado) +(
        ax_escalado*ax_escalado))))*(180/PI); //da el angulo en grados
12  giros_ang_y =filtro(accel_ang_y,gy_escalado);
13  break;
14
15  case 2:
16  read(file, &omega, sizeof(omega));
17  break;
18
19  case 3:
20  read(file, &omega_r, sizeof(omega_));
21  break;
22
23  case 4:
24  alpha=calculo_lqr(omega,omega_ref, giros_ang_y);
25  break;
26
27  case 5:
28  write(file, &alpha, sizeof(alpha);
29  ciclo = 0;
30  break;
31  }
32  }
33
34  ////CONTINUA////
35
36  float calculo_lqr(float A, float B, float gir)
37  {
38  phi=(gir+offset)*DEG_TO_RAD;
39  dphi=gy_escalado*DEG_TO_RAD;
40  u=K[0]*(phi)+K[1]*(dphi)+K[2]*(B+A);
```

```
41 return (u);
42 }
```

Para el caso del control LQRI, a la función *calculo_lqr*() se le debe añadir el término integral:

```
1  float calculo_lqr(float A, float B, float gir)
2  {
3  phi=(gir+offset)*DEG_TO_RAD;
4  dphi=gy_escalado*DEG_TO_RAD;
5  int=int+B-A;
6  u=K[0]*(phi)+K[1]*(dphi)+K[2]*(B+A)+K[3]*int;
7  return (u);
8  }
```

5.3.2 Programación MPC

Para la programación del control MPC, se recurre al algoritmo de optimización diseñado por Pablo Kupra [12]. El algoritmo calcula la acción de control y recibe como argumento de entrada las variables de estado y la referencia que se desea alcanzar.

Para implementar la función se va a prescindir del ejecutivo cíclico que se aplica en los dos controles anteriores. Esto es debido a que el tiempo que tarda la función en obtener el resultado ronda entorno a los 3-4 ms. Con la eliminación del ejecutivo evitamos que salte de nuevo el manejador sin que haya terminado de realizar el cálculo.

Se ha programado el temporizador para que salte cada 20 ms. Se puede apreciar que lo primero que hace la función es enviar el dato de la aceleración calculado en el ciclo anterior, así queda garantizado que Arduino reciba dicho dato cada 20 ms y no dependa del tiempo que tarde la función en calcularlo.

```
1  //FUNCIÓN MANEJADOR
2  void Manejador(int signo, siginfo_t *info, void*context)
3  {
4  write(file, &alpha_A, sizeof(alpha_A));
5  usleep(1000);
6
7  getGyro(&gx_escalado, &gy_escalado, &gz_escalado);
8  getAccel(&ax_escalado, &ay_escalado, &az_escalado);
9  accel_ang_y = atan2(az_escalado , (sqrt((ay_escalado*ay_escalado) + (
       ax_escalado*ax_escalado))))*(180/PI); //da el angulo en grados
10 giros_ang_y =filtro(accel_ang_y,gy_escalado);
11
12 read(file, &omega, sizeof(omega));
13 usleep(1000);
14
15 read(file, &omega_r, sizeof(omega_r));
16 usleep(1000);
17
18 x0Eng[0]=-(giros_ang_y+offset)*0.01745329252;
19 x0Eng[1]=-gy_escalado*0.01745329252;
20 x0Eng[2]=omega;
21 refEng[2]=omega_r;
22 MPCT_EADMM(u_ctrl, x0Eng, refEng);
23 usleep(1000);
24 alpha_A=u_ctrl[0];
25
```

```
26 }
```

5.4 Comunicación Arduino-Raspberry Pi

Echando una vista atrás y acudiendo a la figura 5.1, vemos que es necesario una comunicación bidireccional entre los dispositivos RPi y Arduino. Para llevar a cabo dicha comunicación se ha optado por seguir el protocolo de comunicación I2C. Existen diversas maneras de realizar la comunicación como pueden ser mediante el puerto serie o SPI.

La elección del protocolo I2C es debido entre otros aspectos, a que requiere pocas conexiones y dispone de mecanismos de verificación que indican que el dato ha llegado. Podemos citar como desventaja su velocidad, la cual es media o baja, pero válida para el caso que nos requiere. Se ha comprobado el tiempo de transmisión de datos y ronda en torno a 1 ms. También citar que la conexión no es Full-Duplex, por lo que no se permite el envío y recepción de datos al mismo tiempo, pero dicho inconveniente es salvado con el ejecutivo que se ha programado en el caso del control LQR y con las esperas programas en el caso del MPC .

Los datos a transmitir entre ambos dispositivos serán la velocidad de las ruedas, la referencia de velocidad y la aceleración. Los dos primeros datos serán enviados por Arduino cuando la Pi los requiera, mientras que el último dato será enviado desde la Pi a Arduino. Todos los datos anteriores serán flotantes, por ello, tendremos que hacer una lectura bit a bit para que el dato sea correcto.

5.4.1 Códido Rpi

En la Rpi se ha programado una librería llamada *"ARDUINO.h"* que se encarga de habilitar la comunicación I2C con Arduino. Antes de entrar en detalle y explicar la función, vamos a proceder a mostrarla.

```
1  .
2  .
3  .
4  //Inclusión de librerías
5  .
6  .
7  .
8  //VARIABLES PARA LA COMUNICACIÓN CON ARDUINO
9  int ADRESS_ARDUINO;
10 int file;
11
12 void inicio_arduino(int ddr)
13 {
14 ADRESS_ARDUINO=ddr;
15 file = open("/dev/i2c-1", O_RDWR);
16   if (file < 0) {
17 printf("error abriendo la conexión");
18   }
19     if (ioctl (file, I2C_SLAVE, ADRESS_ARDUINO) < 0) {
20     printf("I2C: error al acceder al esclavo en 0x%x", ADRESS_ARDUINO)
           ;
21 }
22 }
```

Como se puede apreciar, hacemos uso de las llamadas al sistema para el manejo de ficheros en UNIX. Las funciones aquí presentes se encuentran dentro de la librería $< stdio.h >$.

La función ***open()*** se encarga de abrir un fichero existente y adquiere el siguiente prototipo:
int open (char* nombre, int modo, int permisos);

El primer argumento es una cadena que contiene la descripción o nombre del archivo que se desea abrir, en nuestro caso se abre el archivo de comunicación I2C que posee la RPi. El segundo argumento indica el modo con el que se va a trabajar con el archivo, en nuestro caso, se abre el archivo en modo lectura y escritura para poder tener una comunicación bidireccional y poder recibir y enviar datos. El tercer argumento indica los permisos del archivo, este argumento solo se introduce cuando creamos el archivo con *O_CREAT* en el segundo argumento, como no es nuestro caso, prescindimos de este tercer argumento de entrada. La función devuelve un valor -1 en caso de error.

En la librería creada, se hace también uso de la función **ioctl**. Esta función nos permite controlar o comunicarnos con un dispositivo. El primer argumento que recibe es el descriptor del archivo abierto, el segundo argumento es un número de código de requerimiento, mientras que el último argumento es la dirección I2C del dispositivo con el que entabla comunicación.

Una vez explicada la forma con la que se establece la comunicación entre ambos dispositivos en la Rpi, vamos a proceder a explicar como se realiza el envío y la petición de datos.

- Para el envío de datos se utiliza la función **write()**. La función anterior, adquiere la siguiente forma $intwrite(intfichero, void*buffer, unsignedbyte)$;. El primer argumento será el descriptor del fichero sobre el que se actúa, el segundo argumento es un puntero que apunta a la zona de memoria donde se efectúa la transferencia, es decir, donde está el dato que se quiere transferir. Por último, el tercer argumento es el número de bytes que se transfieren.
- De forma análoga, para la recepción de datos se utiliza la función **read()**.Dicha función adquiere una forma similar a **write()**, $intread(intfichero, void*buffer, unsignedbyte)$;. El primer argumento será el descriptor del fichero sobre el que se actúa, el segundo argumento es puntero que apunta a la zona de memoria donde se efectúa la transferencia, es decir, donde se va a almacenar el dato que se recibe. Por último, el tercer argumento es el número de bytes que se transfieren.

A modo de ejemplo, se muestran a continuación las funciones presentes en nuestro código. Indicar que para el envío del número de bytes, se usa la función $sizeof()$, la cuál devuelve el tamaño del dato que recibe como argumento.

- $write(file, alfa, sizeof(alfa))$;
- $read(file, omega, sizeof(omega))$;

5.4.2 Código Arduino

Para la comunicación con la Pi desde Arduino, se hará uso de la librería *Wire.h*, la cual nos permite el uso del protocolo I2C. En la función *setup()* tenemos que configurar la frecuencia de reloj que debe alinearse con la tasa de baudios del bus I2C en la Rpi. A continuación tenemos que establecer la comunicación con la dirección que hayamos asignado mediante la función *Wire.begin(dirección)*, puede asignarse cualquier dirección que no este ocupada por otro dispositivo, en nuestro caso se usa la dirección 0x05. Una vez asignada la dirección, usaremos dos funciones para la lectura y la escritura de datos en el bus. Por un lado estará la función $dato\_recibido()$ y por otro lado, $dato\_enviado()$, ambas recibirán una llamada cuando se escriba o se pida dato por parte del dispositivo maestro.

- función $dato\_recibido()$, guarda en una variable entera que hemos llamado alfa, el dato que recibe de la RPi, que será la aceleración a aplicar. La lectura se realiza mediante la función **Wire.read()**. Observese en el siguiente código como se trabaja con bytes, ya que la transferencia vía I2C es en este formato. Aparece también dos valores enteros que se ponen a cero cuando se recibe un dato. El primer valor j, actualiza el componente del vector que se

envía, de tal manera que cada vez que se recibe un dato, en el siguiente envío Arduino mande el componente cero del vector de envío. El vector de envío está formado por dos componentes, el primero es la velocidad de la ruedas mientras que el segundo es la referencia de velocidad de las ruedas que proporciona el joystick. La segunda variable entera *flag* se pone a cero para que cuando salte la interrupción del timer 2, se actualice el valor de la aceleración que se aplica a las ruedas.

```
1 void dato_recibido(int howMany) {
2 interrupts();
3 volatile float fnum;
4 volatile byte * p = (byte*) &fnum; // CONVIERTE EL VALOR DE UN FLOTANTE EN BYTES
5 while(Wire.available()) // BUCLE PARA LEER MIENTRAS QUEDEN DATOS DISPONIBLES
6 {
7 for (int i = 0; i < 4; i++)
8 *p++=Wire.read(); // GUARDA EL VALOR LEÍDO EN EL PUNTERO A BYTE
9 }
10 alfa=(float)fnum; // GUARDA EN ALFA EL VALOR RECIBIDO COMO FLOTANTE
11 j=0; // PONE A 0 EL VALOR DEL VECTOR DE ENVÍO PARA QUE CUANDO LA RPI PIDA
   UN VALOR SE LE ENVÍE PRIMERO LA VELOCIDAD DE LOS MOTORES
12 flag=0; // PONE LA BANDERA A CERO PARA QUE SE PROCESE EL DATO RECIBIDO EN
   LAS INTERRUPCIONES DEL TIMER 2, ACTUALIZÁNDOSE EL VALOR DE LA ACELERACI
   ÓN A APLICAR.
13 }
```

- Función *dato_enviado*(), envía el dato que contiene el vector de envío cuando la Rpi realiza una petición de lectura. El envío se realiza mediante la función **Wire.write()**. Observese que se envía un dato en bytes con el tamaño de un flotante. El primer dato enviado será la velocidad de las ruedas, una vez enviado el dato, se actualiza el índice del vector de envío, para que en la siguiente petición se envíe el siguiente componente que será la referencia de velocidad. La sincronización entre todos los códigos hace que el entero *j* adquiera únicamente el valor 0 ó 1.

```
1 void dato_enviado() {
2 // VECTOR ENVÍO, PRIMERO SE ENVÍA LA COMPONENTE 0 QUE EQUIVALE A LA VELOCIDAD DE
   LOS MOTORES
3 Wire.write((byte*) &vel, sizeof(float));
4 j++; // INCREMENTA EL VALOR DE J PARA QUE EN LA SIGUIENTE PETICIÓN DE DATO SE ENVÍE LA
   COMPONENTE 1 QUE EQUIVALE A LA REFERENCIA EN VELOCIDAD.
5 }
```

5.5 Comunicación Bluetooth

Para la comunicación Bluetooth se hace uso del módulo HC-06 que irá conectado a Arduino. Los datos que recibe serán enviados desde la App Android *Joystick BT Commander*.

Lo primero que tendremos que hacer es configurar el módulo Bluetooth mediante los comandos AT de forma manual.

```
1 #include <SoftwareSerial.h>
2 SoftwareSerial Bluetooth(10,11);
3
4 void setup()
5 {
6 BT.begin(9600); // Velocidad del puerto del módulo Bluetooth
```

```
7   Serial.begin(9600); // Abrimos la comunicación serie con el PC y establecemos velocidad
8 }
9
10 void loop()
11 {
12   if(Bluetooth.available())
13   {
14     Serial.write(Bluetooth.read());
15   }
16   if(Serial.available())
17   {
18     Bluetooth.write(Serial.read());
19   }
20 }
```

- Tecleando *"AT+NAMEXX"*, cambiaremos el nombre del módulo que pasará a llamarse XX. El nombre estará compuesto como máximo por 20 caracteres.
- Tecleando *"AT+PINXXXX"*, cambiaremos el pin del módulo que pasará a ser XXXX.
- Tecleando *"AT+BAUDX"*, cambiaremos la velocidad del módulo qué pueder ser:

Tabla 5.1 Velocidades módulo HC-06.

X	velocidad (baudios)
1	1200
2	2400
3	4800
4	9600
5	19200
6	38400
7	57600
8	115200
9	230400
A	460800
B	921600
C	1382400

El código que interpreta los datos que envía la APP será el siguiente:

```
1 void blue()
2 {
3 if(bluetooth.available())  {
4 delay(2);
5 cmd[0] =  mySerial.read();
6 if(cmd[0] == STX) {
7 int  i=1;
8 while(bluetooth.available())  {
9 delay(1);
10 cmd[i] = bluetooth.read();
11 if(cmd[i]>127 ||  i>7)                    break;
12 if((cmd[i]==ETX) && (i==2 || i==7))    break;
13 i++;
14 }
```

```
15 if (i==7)                getJoystickState (cmd);
16 }
17 }
18 }
19
20
21 void  getJoystickState (byte data[8])     {
22 int  joyX = (data[1]-48)*100 + (data[2]-48)*10 + (data[3]-48);
23 int  joyY = (data[4]-48)*100 + (data[5]-48)*10 + (data[6]-48);
24 joyX = joyX - 200;
25 joyY = joyY - 200;
26 if (joyX<-100 || joyX>100 || joyY<-100 || joyY>100)      return ;
27 giro_M1=joyX;
28 vec[1]=joyY*0.4;
29 }
```

Se aprecia que los valores del eje X son los que determinarán el giro, mientras que los valores del eje Y serán los que nos varíen la referencia de velocidad. El joystick puede considerarse una circunferencia de radio 100, de tal manera que se escala la velocidad multiplicando el valor recibido por 0.4. Así, la referencia de velocidad máxima que vamos a obtener en el control remoto será de +-40 rad/s.

6 Resultados

Llegados a este punto de la memoria, ya tenemos diseñado los controladores a aplicar así como la explicación de los códigos involucrados. Se mostrarán a continuación los resultados obtenidos mediante los diferentes métodos así como las condiciones en las que se han llevado a cabo los experimentos.

Como nota, indicaremos que se mostrarán 3 gráficas por experimento. La primera de ella contendrá conjuntamente las 3 variables de estado (ángulo de inclinación del vehículo con respecto al plano vertical, velocidad de caída del vehículo y velocidad angular de las ruedas) así como la acción de control (aceleración angular de las ruedas). La segunda gráfica mostrará el ángulo de inclinación del vehículo de la gráfica primera por separado para apreciar con mayor claridad sus valores. La tercera gráfica será análoga a la segunda pero mostrando la velocidad angular de las ruedas.

6.1 Control LQR

6.1.1 Seguimiento de referencia nula en velocidad de las ruedas

Para este experimento, se ha dejado al vehículo quieto sobre un plano horizontal durante 50 segundos. Se aprecia como aparecen pequeñas variaciones del ángulo no superiores a ningún momento a 2°. Este pequeño ruido que puede apreciarse es debido a pequeños errores en la medida del MPU-6050. La amplitud de las pequeñas ondas es similar a las obtenidad con el control LQRI, pero mayores a las que se obtienen con el control MPC que estabiliza mejor en referencias nulas de velocidad.

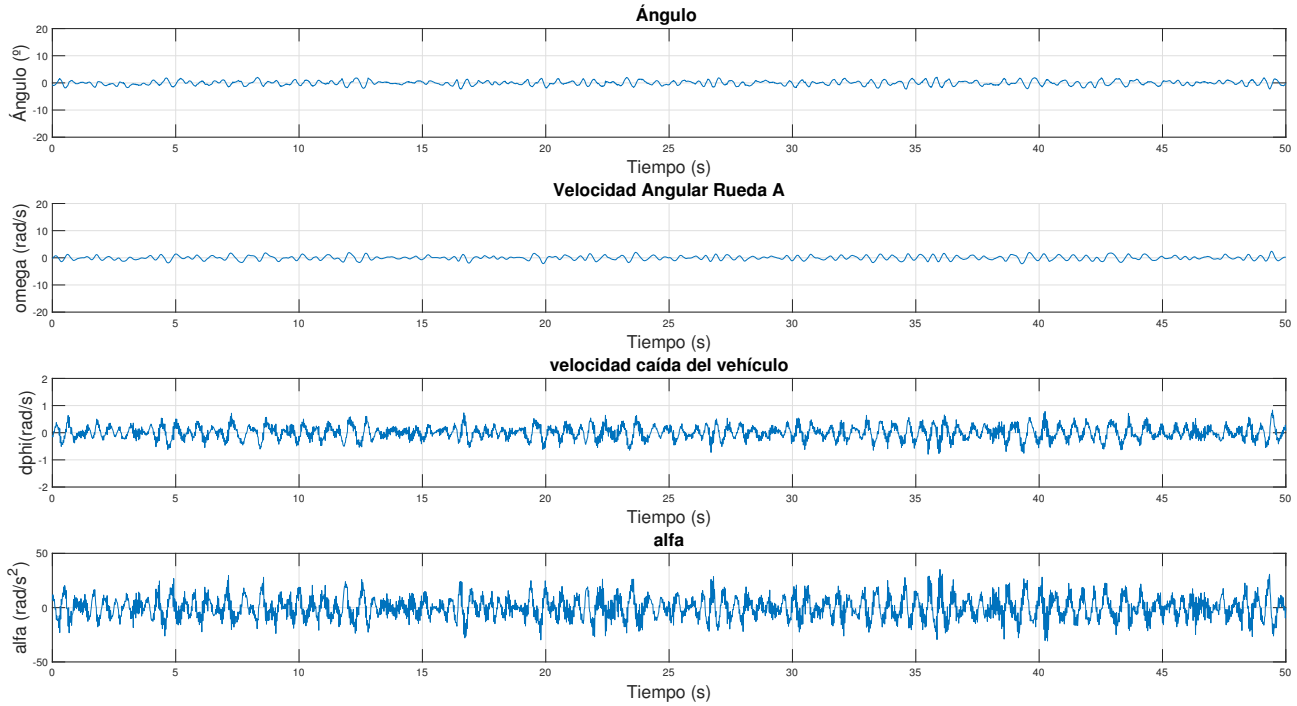


Figura 6.1 LQR: Variables ante ensayo de seguimiento de referencia nula en velocidad.

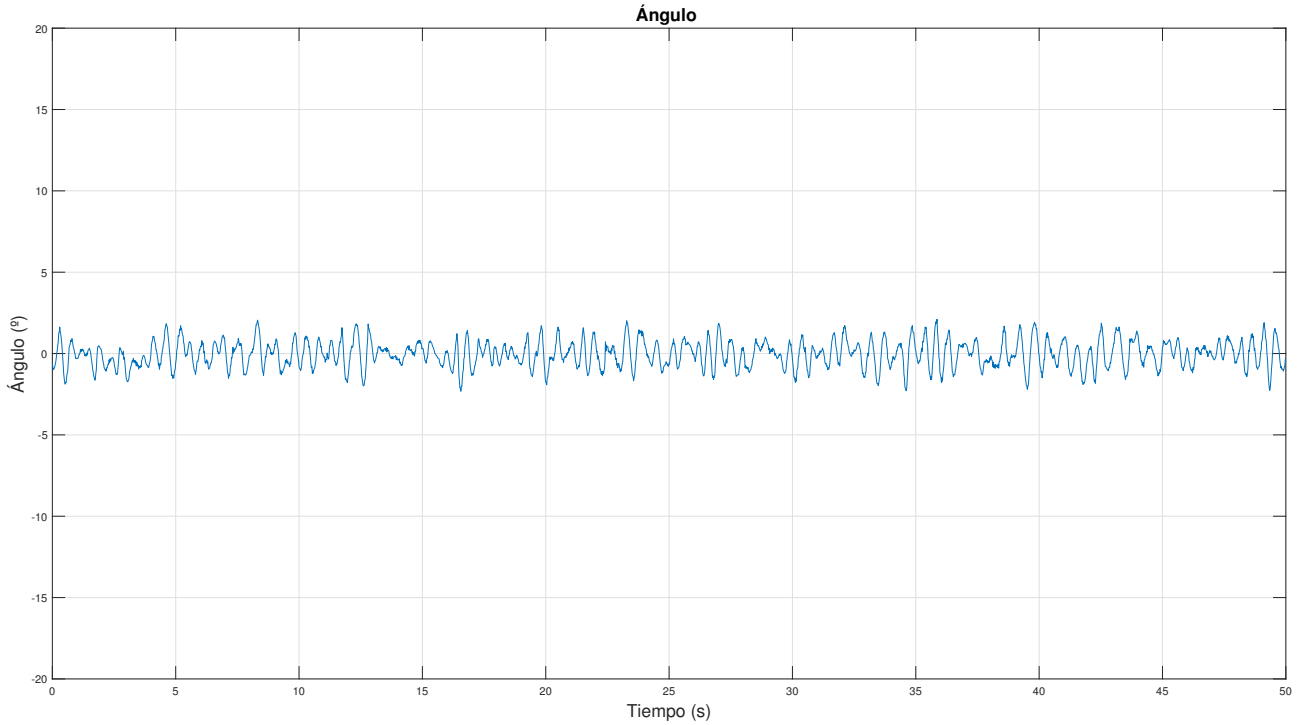


Figura 6.2 LQR: Ángulo ante ensayo de seguimiento de referencia nula en velocidad.

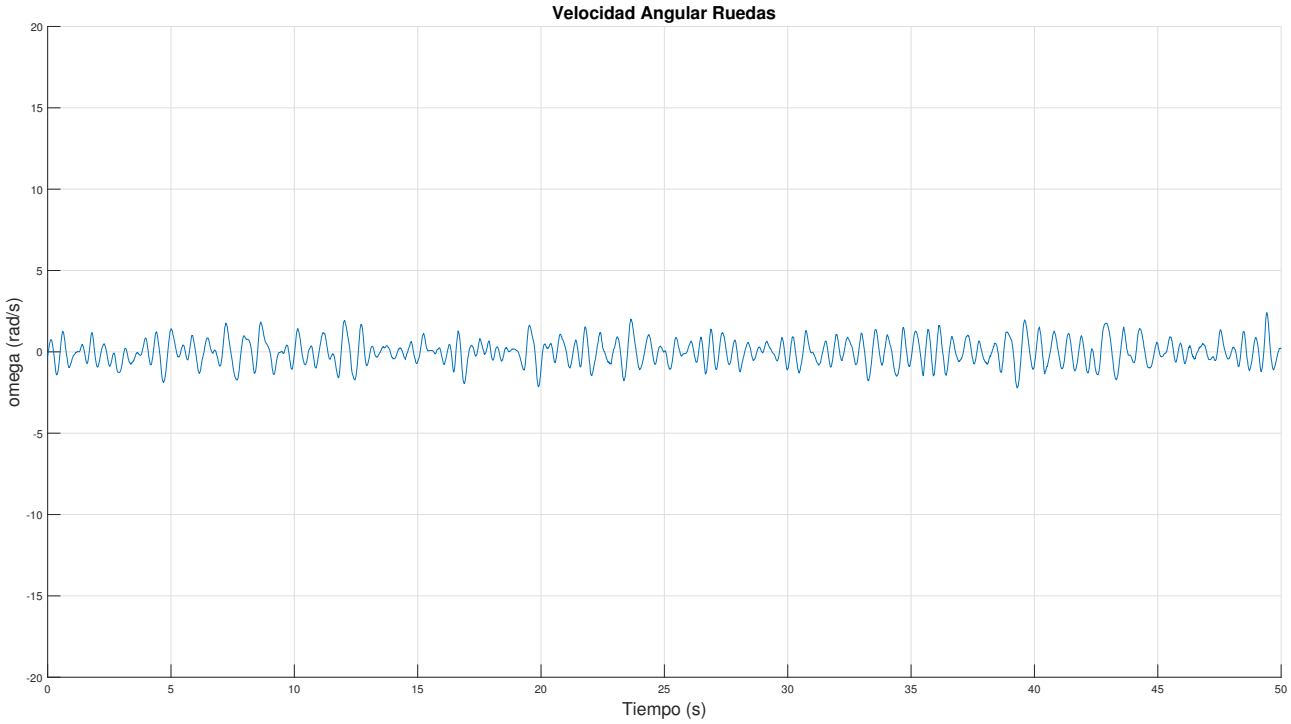


Figura 6.3 LQR: Velocidad angular de las ruedas ante ensayo de seguimiento de referencia nula en velocidad.

6.1.2 Perturbaciones tipo pulso

En este experimento, colocamos el vehículo sobre un plano horizontal con un seguimiento en referencias de velocidad nula y cada cierto tiempo se dieron pequeños pulsos. El experimento dura 40 segundos, el primer pulso se da poco antes de los 5s , el segundo pulso pasados los 10s y el último pulso pasados los 25s. Se aprecia que el vehículo se inclina sobre -15° y vuelve a recuperar su posición de equilibrio.

El funcionamiento es simple, al darle el pequeño golpe seco, el vehículo se inclina y tiende a acelerar en la dirección hacia donde se esta cayendo, aumentando la velocidad de las ruedas, momento en el que empieza a actuar el sistema de control que se opondrá a esta caída, cambiando el sentido de la velocidad y volviendo a su posición de equilibrio.

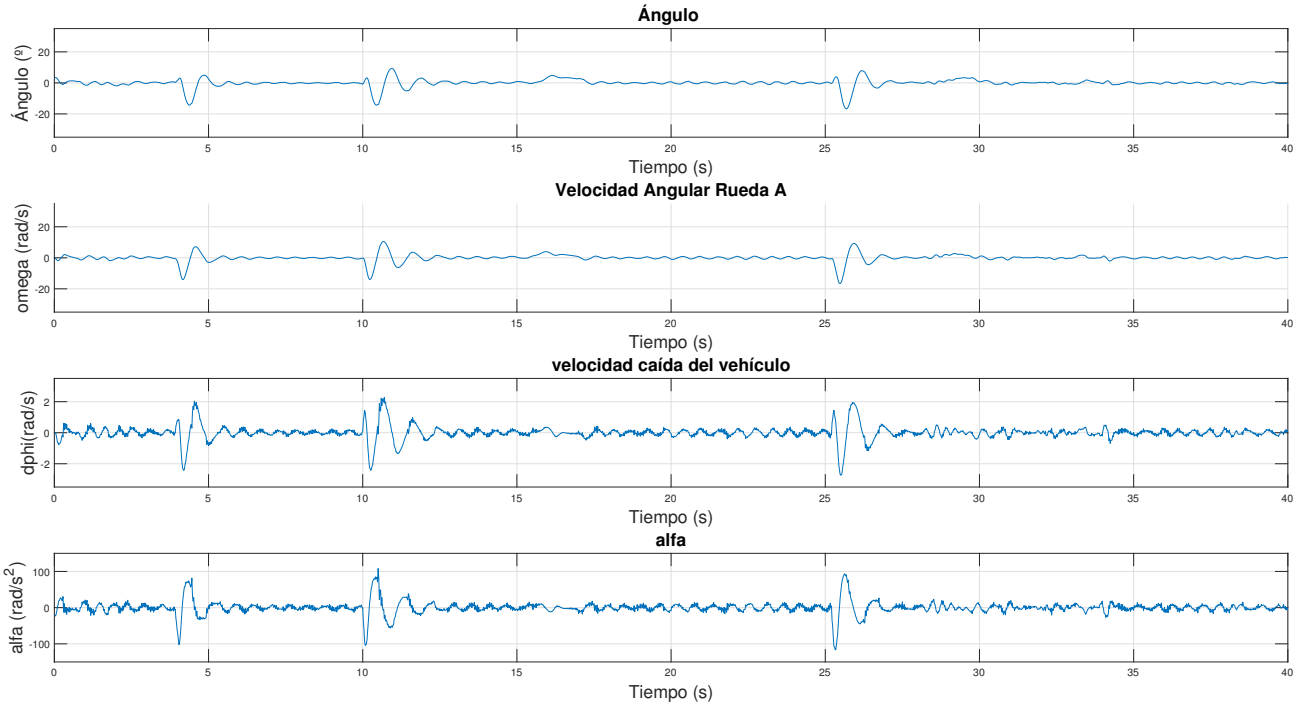


Figura 6.4 LQR: Variables ante ensayo de perturbaciones tipo pulso.

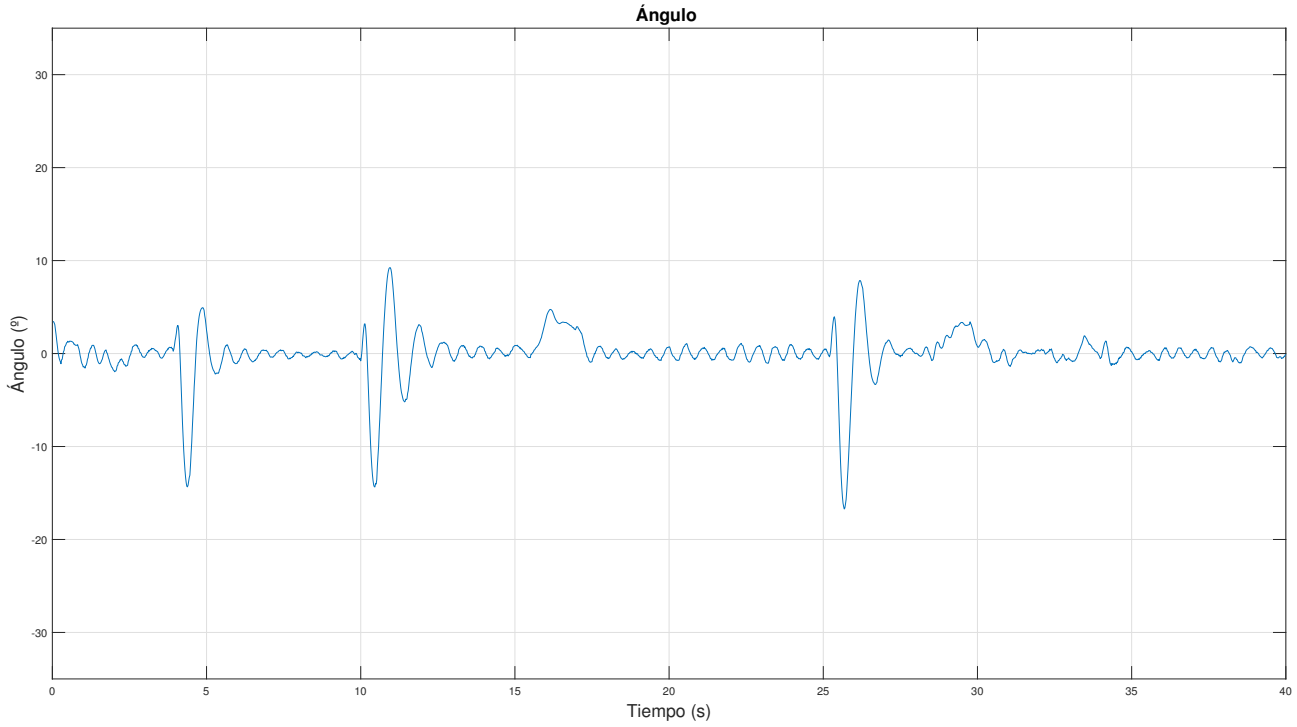


Figura 6.5 LQR: Ángulo ante ensayo de perturbaciones tipo pulso.

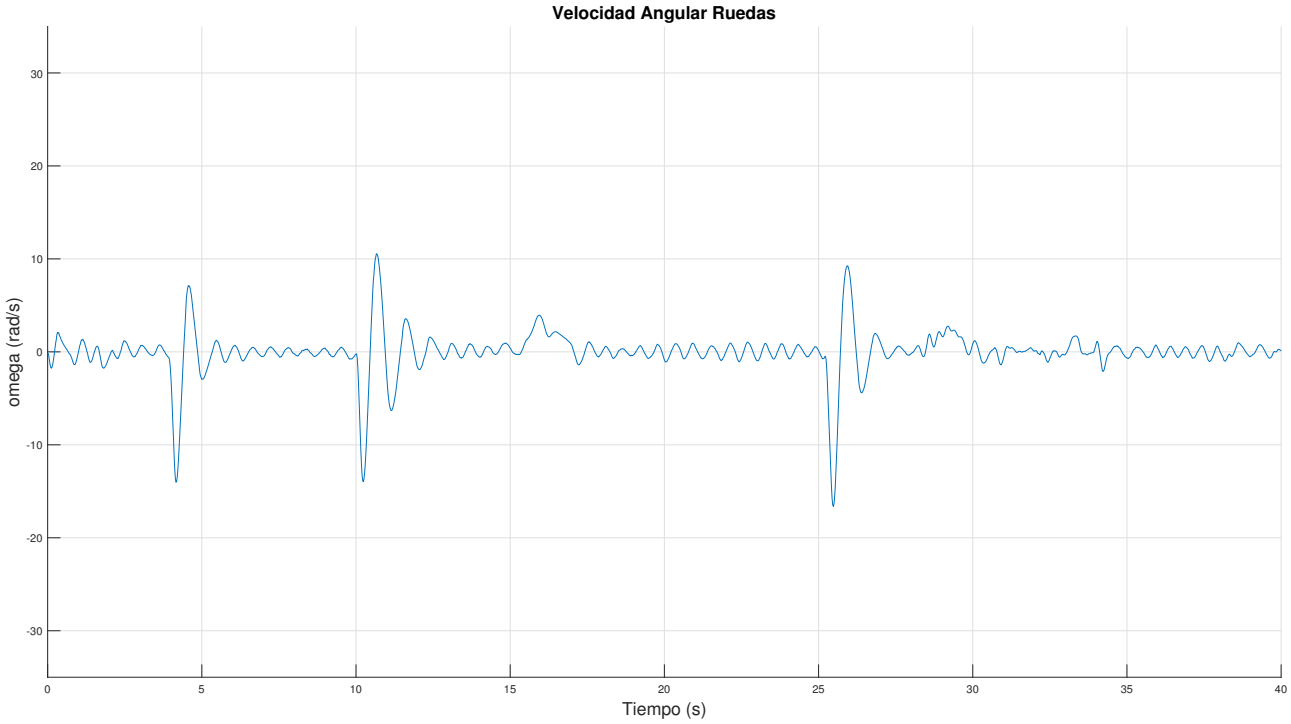


Figura 6.6 LQR: Velocidad angular de las ruedas ante ensayo de perturbaciones tipo pulso.

6.1.3 Seguimiento ante cambios de referencia de velocidad de las ruedas

En este ensayo se ha producido un cambio en la referencia de velocidad de las ruedas. Este cambio en las referencias de velocidad se produce cada 2s y es de -10 rad/s descendente hasta llegar a -20 rad/s, y de -5 rad/s desde -20 rad/s hasta llegar a -30 rad/s.

Se puede apreciar un cambio brusco en el ángulo al cambiar la referencia de velocidad, en cada cambio de referencia el vehículo llega a inclinarse en torno a 18°, es por ello que se decide que el cambio de referencia sea el indicado, pues para cambios mayores el vehículo alcanzaba tal ángulo que no era capaz de compensar el ángulo que alcanzaba y por ello caía. Este problema que se verá solucionado en los dos siguientes controladores.

Se puede apreciar que se produce un error en régimen permanente, pues la velocidad a la que llegarán las ruedas no alcanza la referencia deseada. Problema que será solucionado con los controladores venideros.

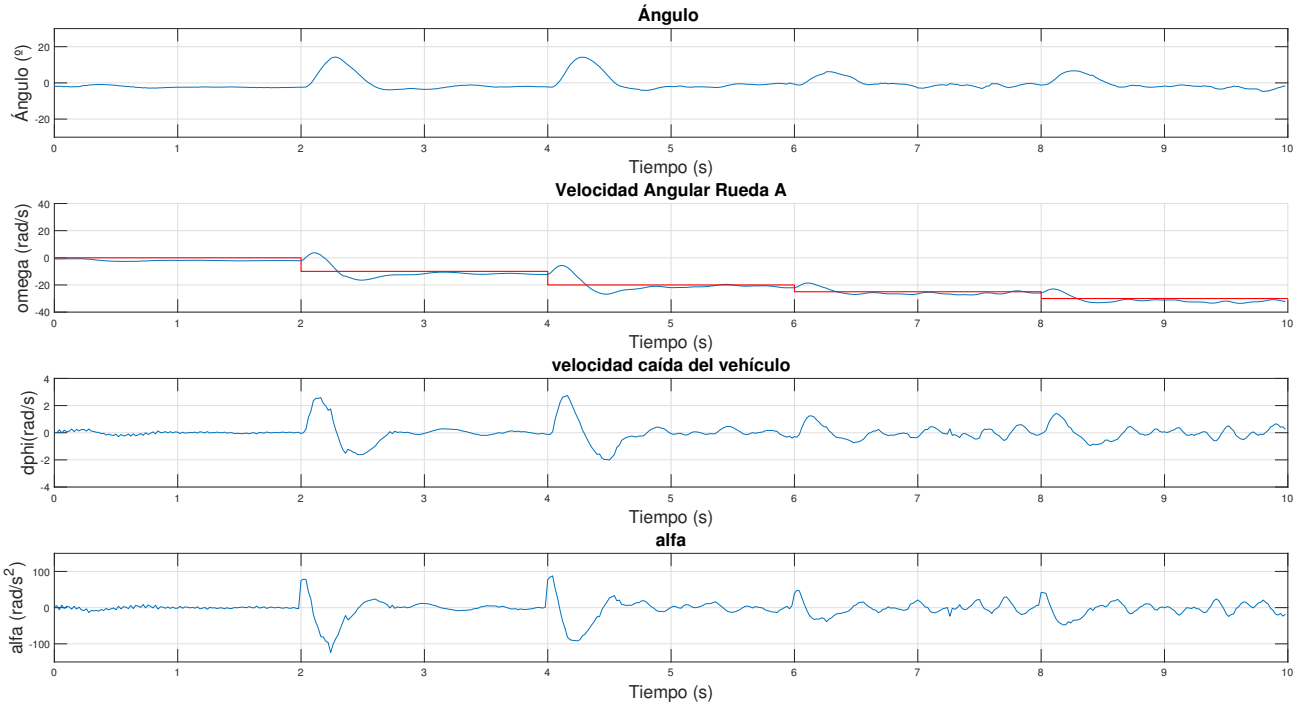


Figura 6.7 LQR: Variables ante ensayo de seguimiento de referencia cambiante en velocidad.

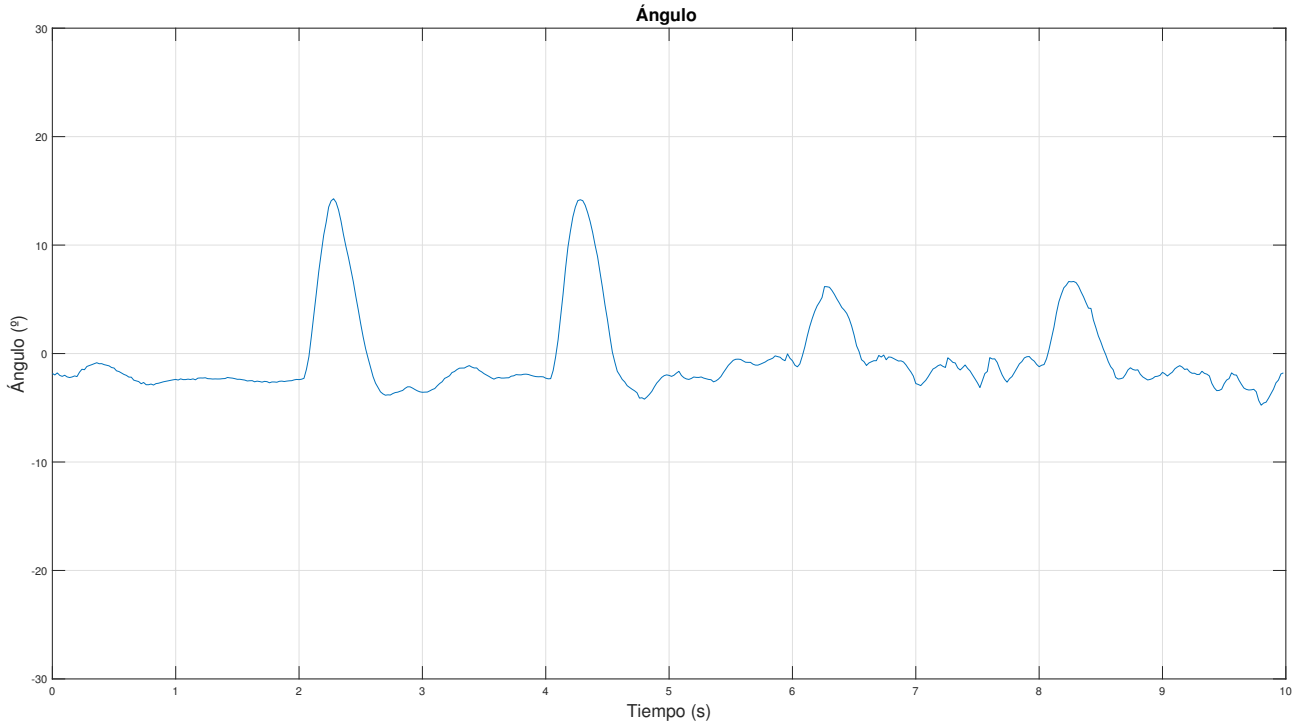


Figura 6.8 LQR: Ángulo ante ensayo de seguimiento de referencia cambiante en velocidad.

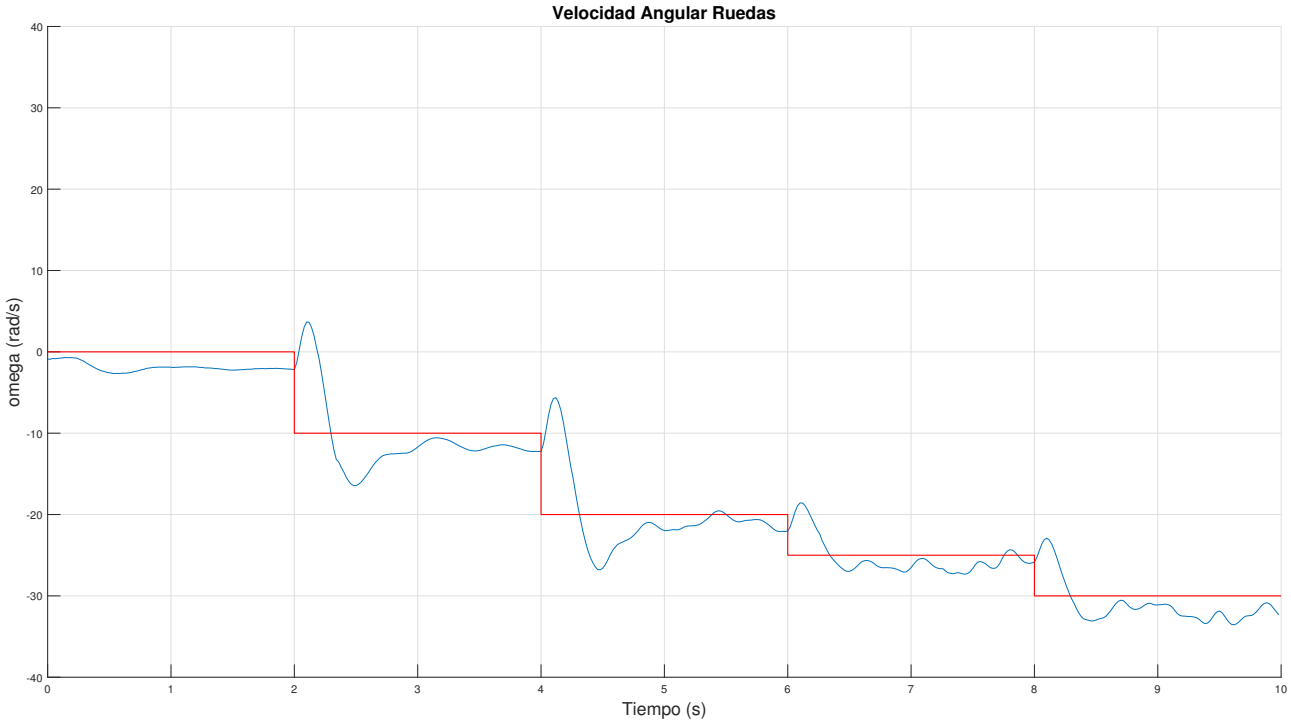


Figura 6.9 LQR: Velocidad angular de las ruedas ante ensayo de seguimiento de referencia cambiante en velocidad.

6.1.4 Ascenso y Descenso en plano inclinado

Este ensayo consistió en forzar al vehículo, aplicándole una fuerza con la mano, para subir un plano inclinado de 20°, una vez suba la rampa se estabilizara durante unos cuantos segundos y se lanzará por ella. En la subida, el vehículo se inclina hacia donde se le aplica la fuerza, es decir, hacia abajo de la cuesta, para compensar esta perturbación que será superior a la fuerza peso que aplica la gravedad. Al caer, el vehículo se inclinará hacia atrás, hacia el alto de la cuesta, detectando una variación en su ángulo de inclinación que será corregido por el sistema de control para volver a la referencia nula de sus variables. El funcionamiento es similar a la perturbación tipo pulso explicada en la sección anterior.

Nótese como en la caída, el ángulo inclinación alcanza valores cercano a los 30° y aceleraciones de hasta 100 rad/s^2. Este ensayo dura 60 segundos y se realizan 5 descensos por el plano inclinado. Se parte del vehículo en lo alto de la rampa y a los 6 segundos se deja caer viéndose como el vehículo se inclina (picos hacia arriba en figura 6.11) y la velocidad de las ruedas aumentan hasta que el sistema de control lo estabiliza. Sobre el segundo 12 se empieza aplicar durante 2 segundos un empuje por nuestra parte (mesetas hacia abajo en figura 6.11) para que el vehículo suba la rampa.

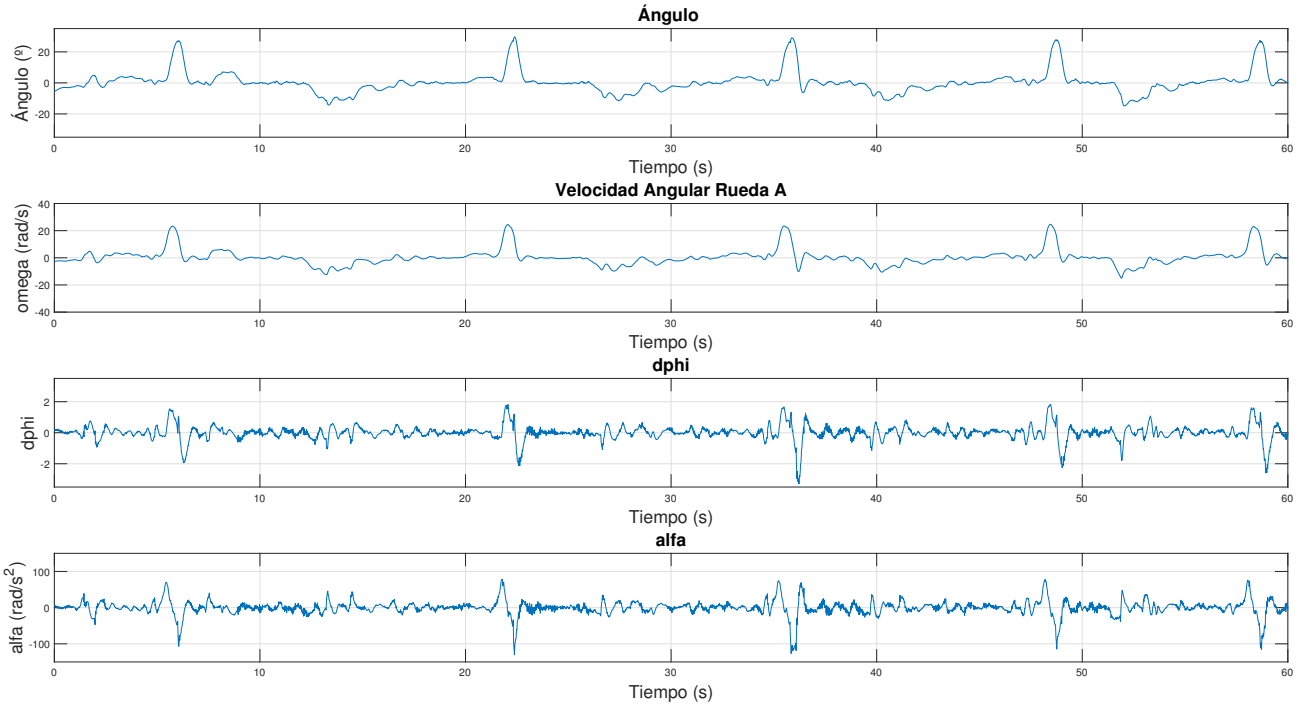


Figura 6.10 LQR: Variables ante ensayo en plano inclinado.

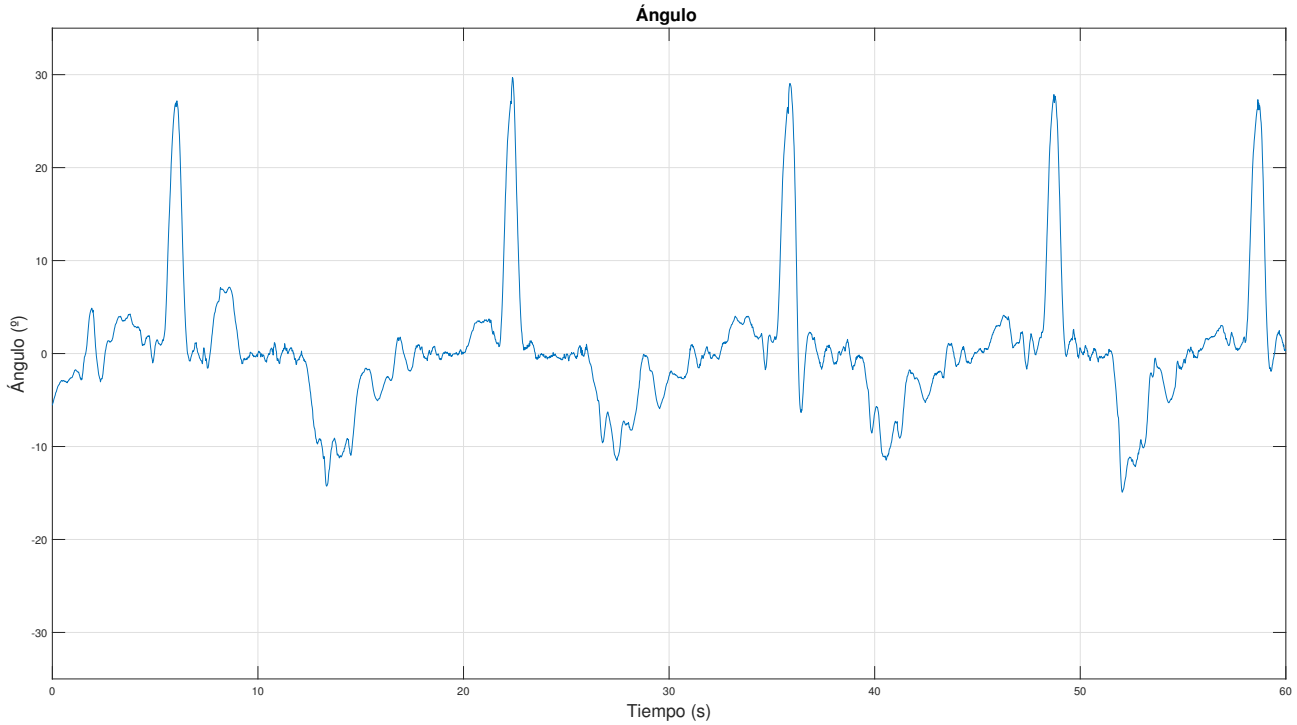


Figura 6.11 LQR: Ángulo ante ensayo en plano inclinado.

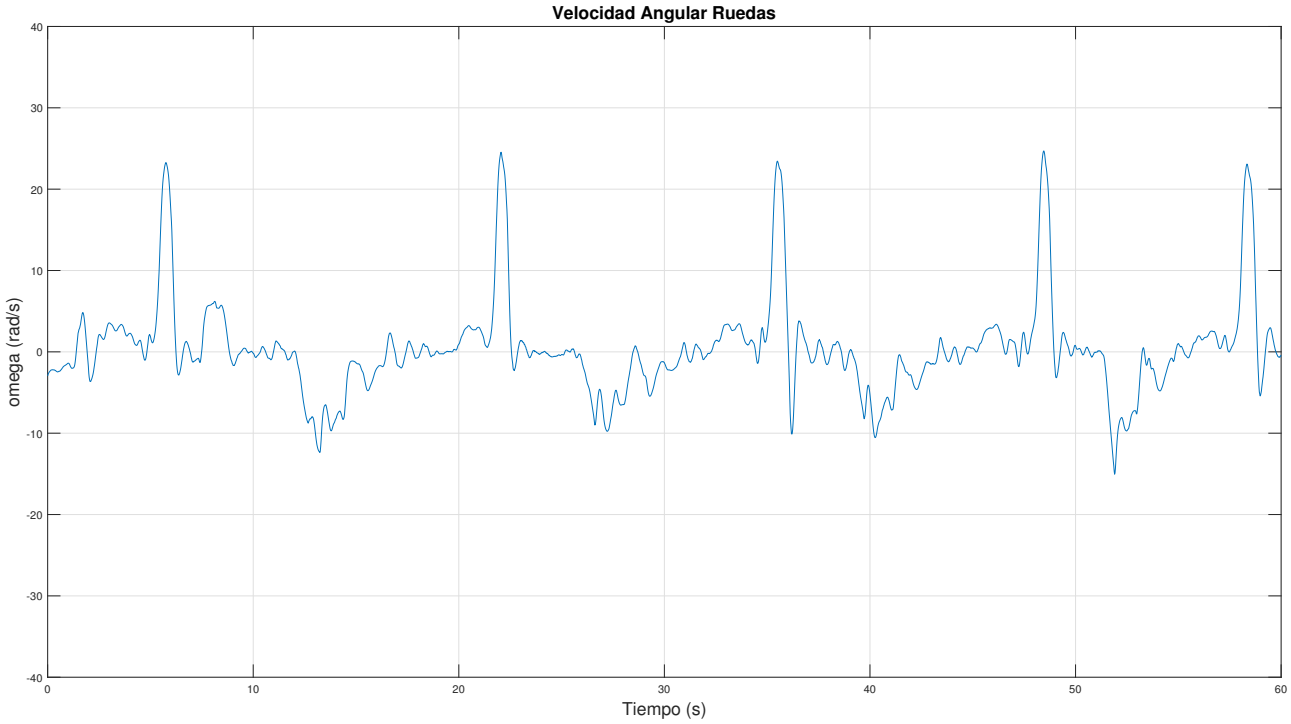


Figura 6.12 LQR: Velocidad angular de las ruedas ante ensayo en plano inclinado.

6.1.5 Ascenso y Descenso en plano inclinado control remoto

A diferencia del ensayo anterior, en este ensayo se hizo subir al vehículo por un plano inclinado de 15°, mediante el uso del joystick, una vez suba la rampa se estabilizará durante unos cuantos segundos y se le hará bajar. El vehículo al caer se inclinará hacia lo alto de la cuesta detectando una variación en su ángulo de inclinación que será corregido por el sistema de control para volver a la referencia nula de sus variables. Para el ascenso de la cuesta, se varía la velocidad de las ruedas mediante el control remoto. En el movimiento de ascensión, la única fuerza presente será la gravitatoria, el vehículo al detectarla se inclinará hacia lo alto de la cuesta para vencerla y conseguir la ascensión.

Este ensayo dura 50 segundos y se realizan 3 descensos por el plano inclinado. Se parte del vehículo al inicio de rampa y en el segundo 3 se varía la referencia de velocidad para hacer subir al vehículo por la rampa. Se aprecia que la subida es lenta, pues dura cerca de 4 segundos y el ángulo que alcanza es cercano a 20°, para compensar la fuerza de la gravedad. El descenso se produce en el segundo 11, se dirige el vehículo hasta donde comienza la rampa en sentido descendente. Se repite la operación de subida en los segundos 18 y 35.

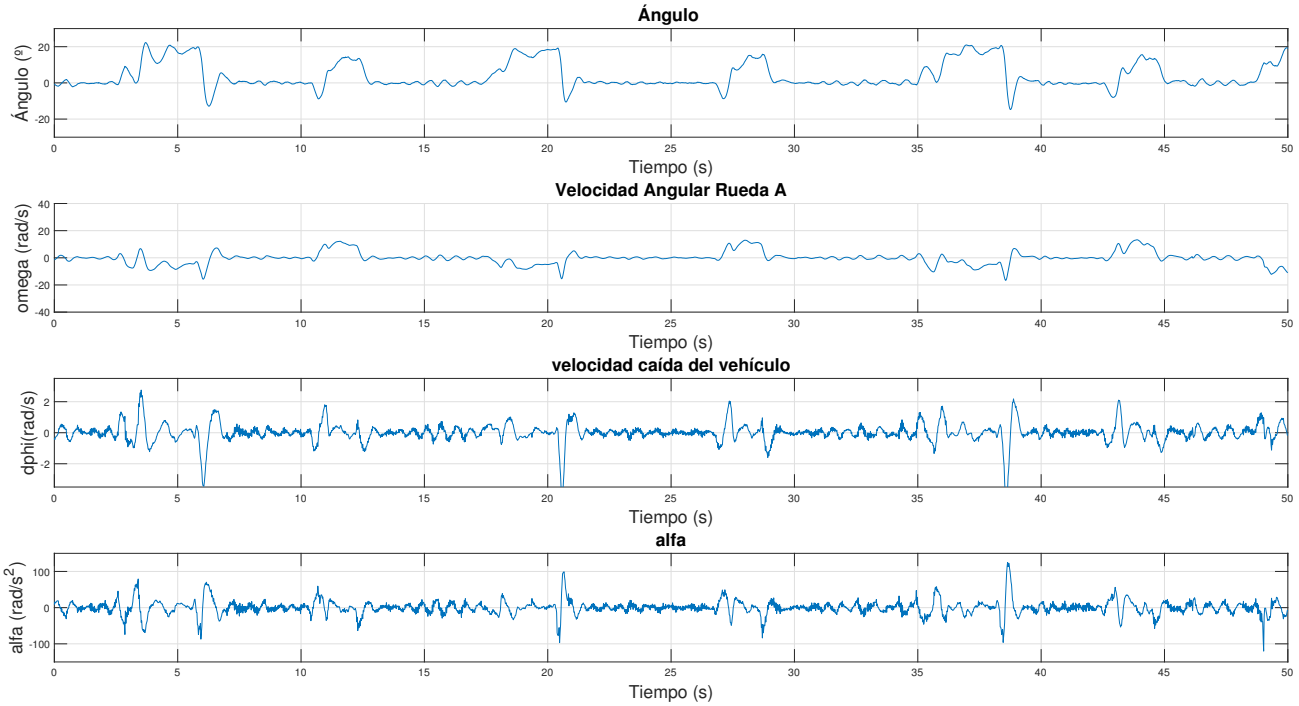


Figura 6.13 LQR: Variables ante ensayo en plano inclinado. Control remoto.

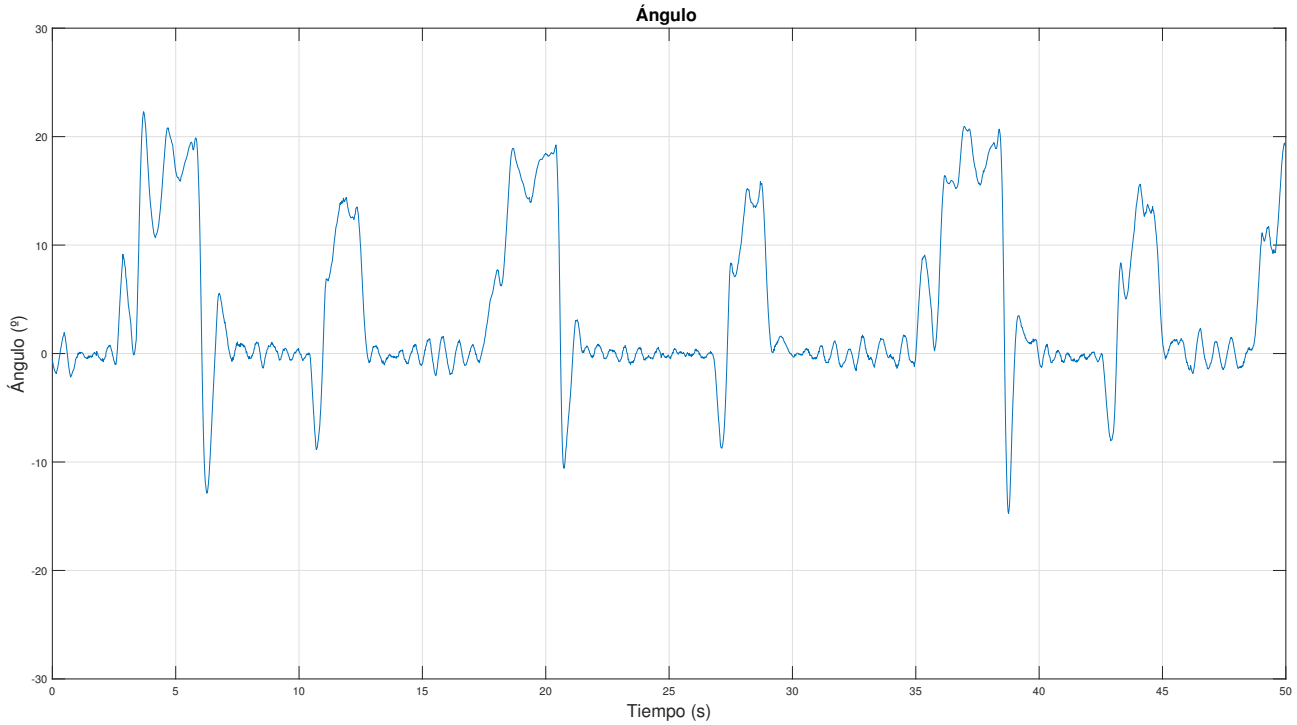


Figura 6.14 LQR: Ángulo ante ensayo en plano inclinado. Control remoto.

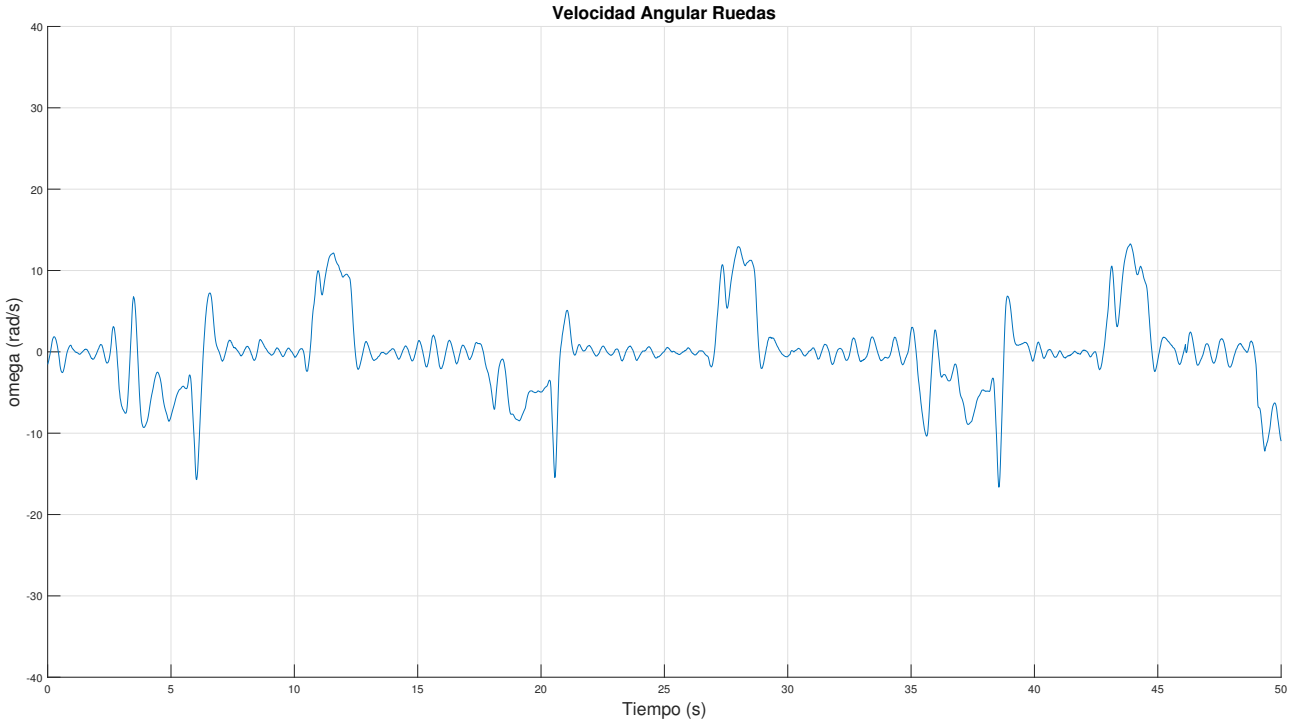


Figura 6.15 LQR: Velocidad angular de las ruedas ante ensayo en plano inclinado. Control remoto.

6.2 Control LQR con efecto integral

6.2.1 Seguimiento de referencia nula en velocidad de las ruedas

Poco que añadir con respecto a este experimento, al igual que el control LQR, se deja al vehículo quieto sobre un plano horizontal durante 35 segundos. Se aprecia como aparecen pequeñas variaciones del ángulo no superiores a ningún momento a 2°, similares al control anterior.

La adición del término integral pretende corregir la desviación del centro de masas del vehículo. Sin embargo sigue apreciándose un minúsculo ruido en la aceleración de las ruedas que proviene de los datos que obtenemos mediante el MPU-6050.

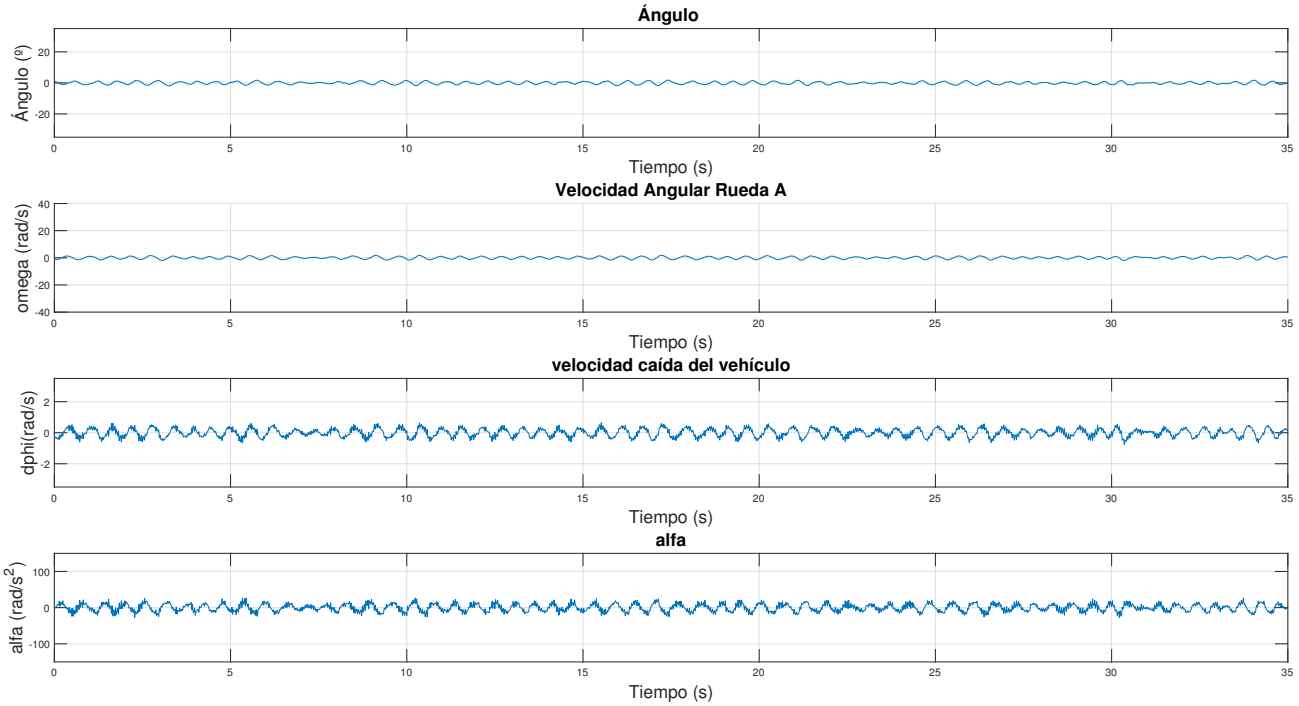


Figura 6.16 LQRI: Variables ante ensayo de seguimiento de referencia nula en velocidad.

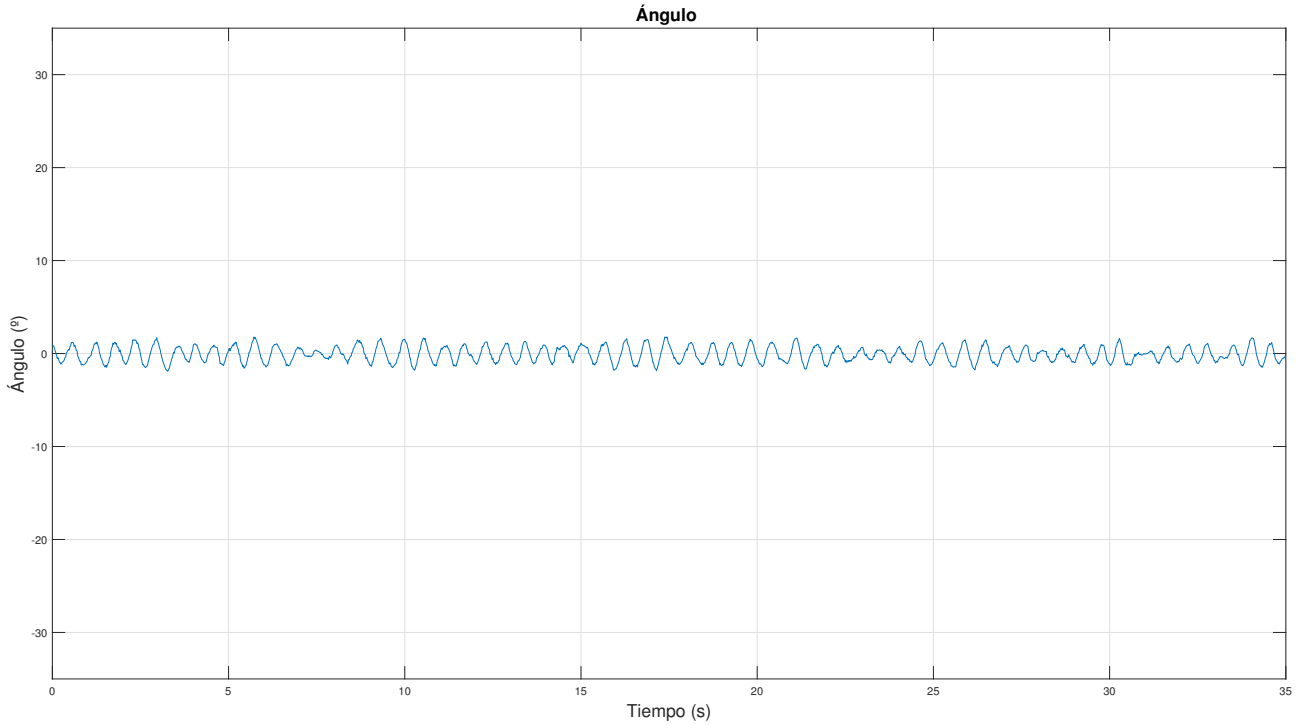


Figura 6.17 LQRI: Ángulo ante ensayo de seguimiento de referencia nula en velocidad.

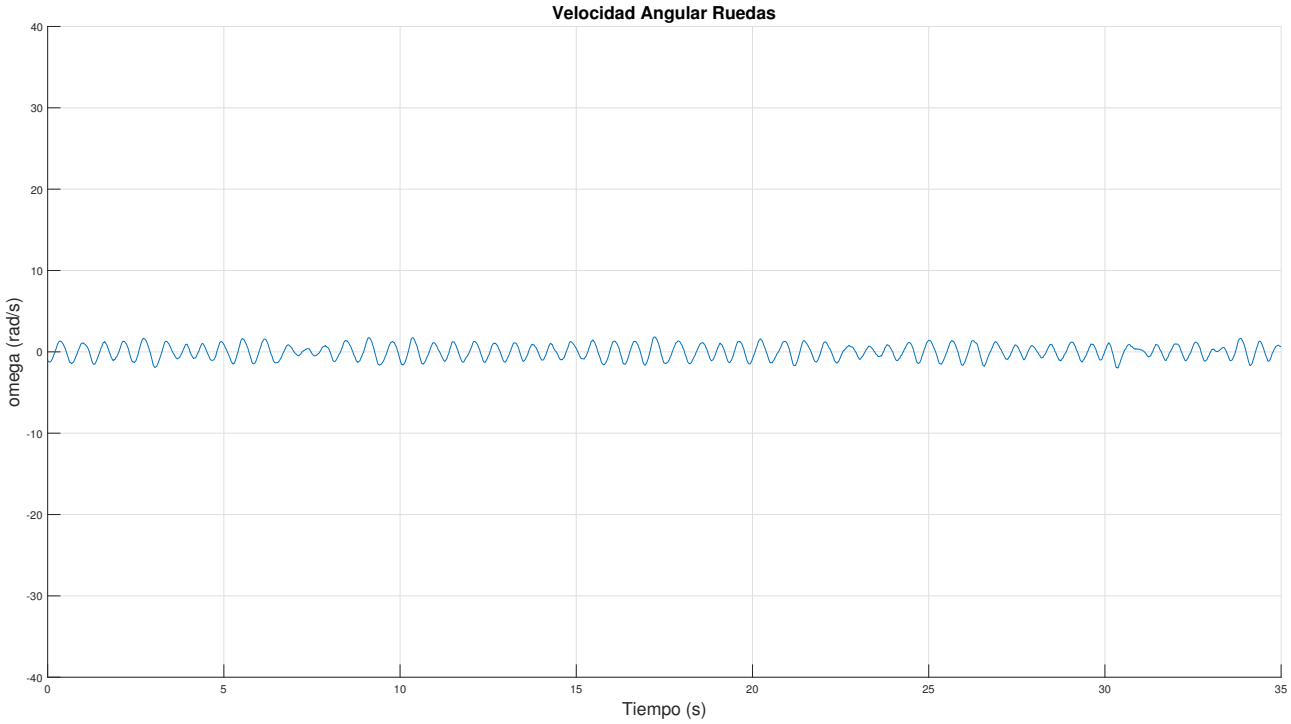


Figura 6.18 LQRI: Velocidad angular de las ruedas ante ensayo de seguimiento de referencia nula en velocidad.

6.2.2 Perturbaciones tipo pulso

En este experimento, colocamos el vehículo quieto durante 50 segundos sobre un plano horizontal con un seguimiento de referencia nula en sus variables y cada cierto tiempo intentamos perturbar su posición de equilibrio dando pequeños golpes secos. Se aprecia que el vehículo alcanza inclinaciones superiores a las del control anterior. En este caso, llega a alcanzar casi 30° y vuelve a recuperar su posición de equilibrio.

Se realizan 6 pulsos en cada cara del vehículo. Entre cada pulso se deja que el vehículo se estabilice entorno a 0° durante 3 segundos.

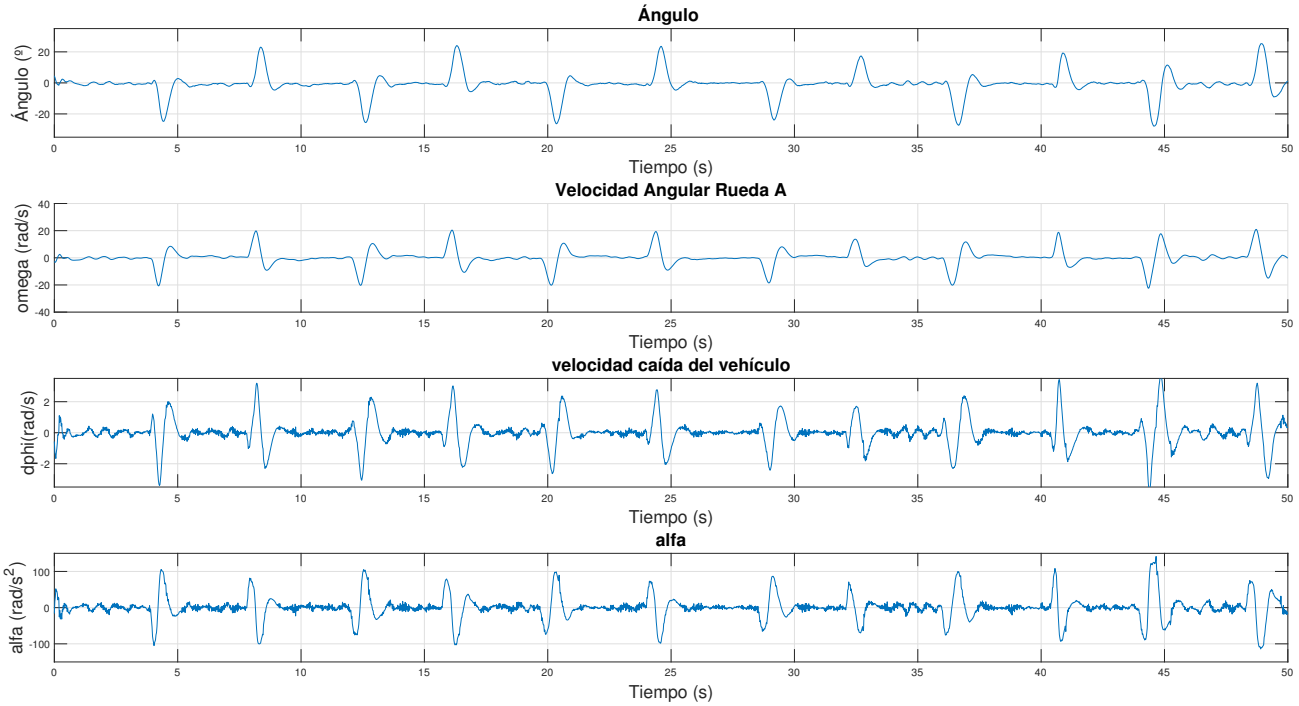


Figura 6.19 LQRI: Variables ante ensayo de perturbaciones tipo pulso.

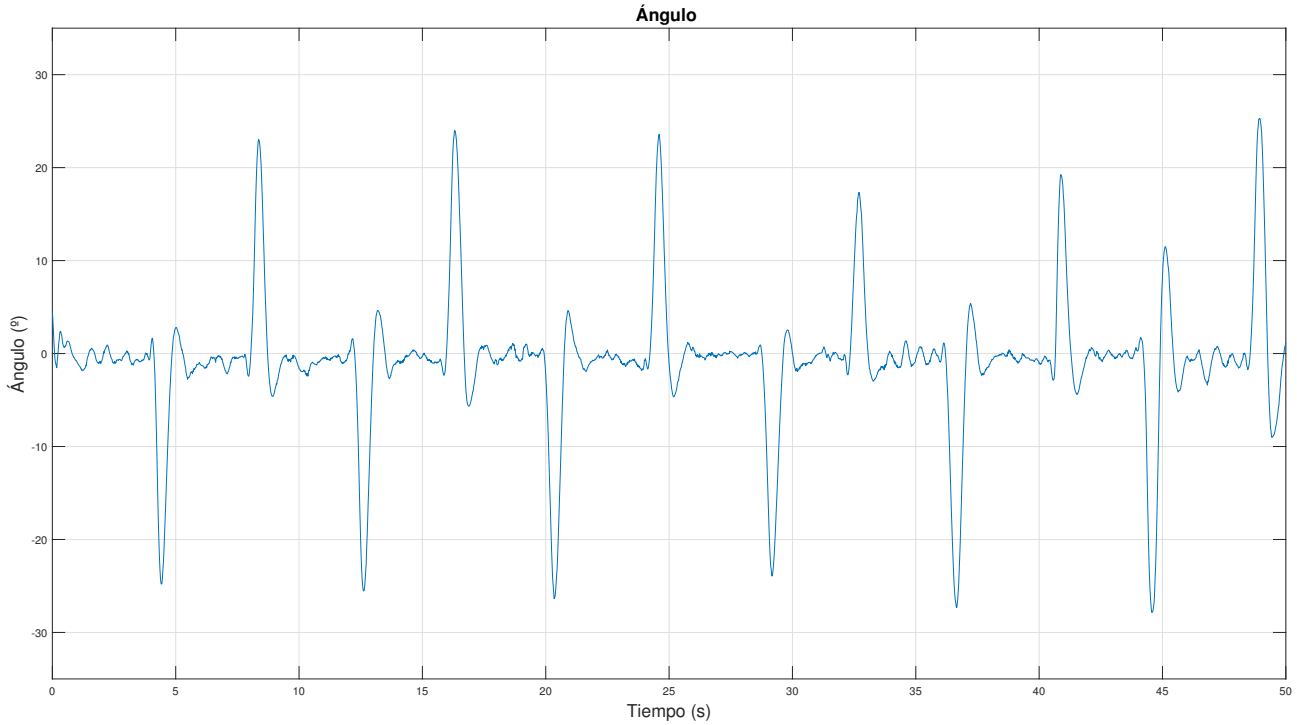


Figura 6.20 LQRI: Ángulo ante ensayo de perturbaciones tipo pulso.

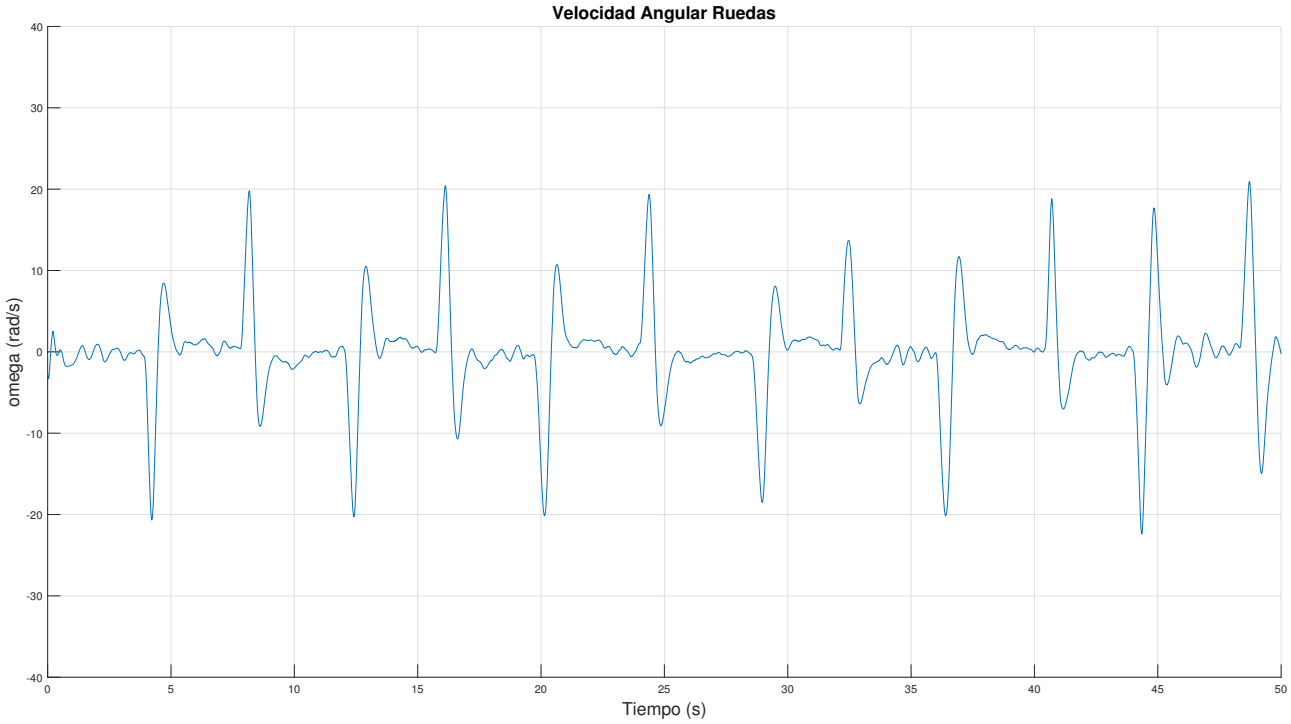


Figura 6.21 LQRI: Velocidad angular de las ruedas ante ensayo de perturbaciones tipo pulso.

6.2.3 Seguimiento ante cambios de referencia de velocidad de las ruedas

En este ensayo se ha producido un cambio en la referencia de velocidad de las ruedas. Este cambio en las referencias de velocidad se produce cada 4s y es de 10 rad/s ascendente y -10rad/s descendente. Se parte de una velocidad nula hasta llegar a 20 rad/s seguidamente se cambia el sentido de movimiento hasta llegar a los -20 rad/s y volver a 0 rad/s, así repetidas veces. El vehículo es capaz de alcanza mayores velocidades de las que aquí se muestran, sin embargo, se decide incluir esta gráfica para que se aprecie el movimiento de ida y vuelta del vehículo.

Aunque se aprecia un cambio algo más suave en la variación del ángulo al cambiar la referencia de velocidad, sigue siendo más brusco que con el control MPC. En cada cambio de referencia el vehículo llega a inclinarse en torno a 15°, se decide mantener el cambio de referencia en +-10 rad/s, aunque en este caso nos permitía cambios mayores sin llegar a caerse. Se corrige el error en régimen permanente del control anterior, pues la velocidad a la que llegan las ruedas alcanza la referencia indicada.

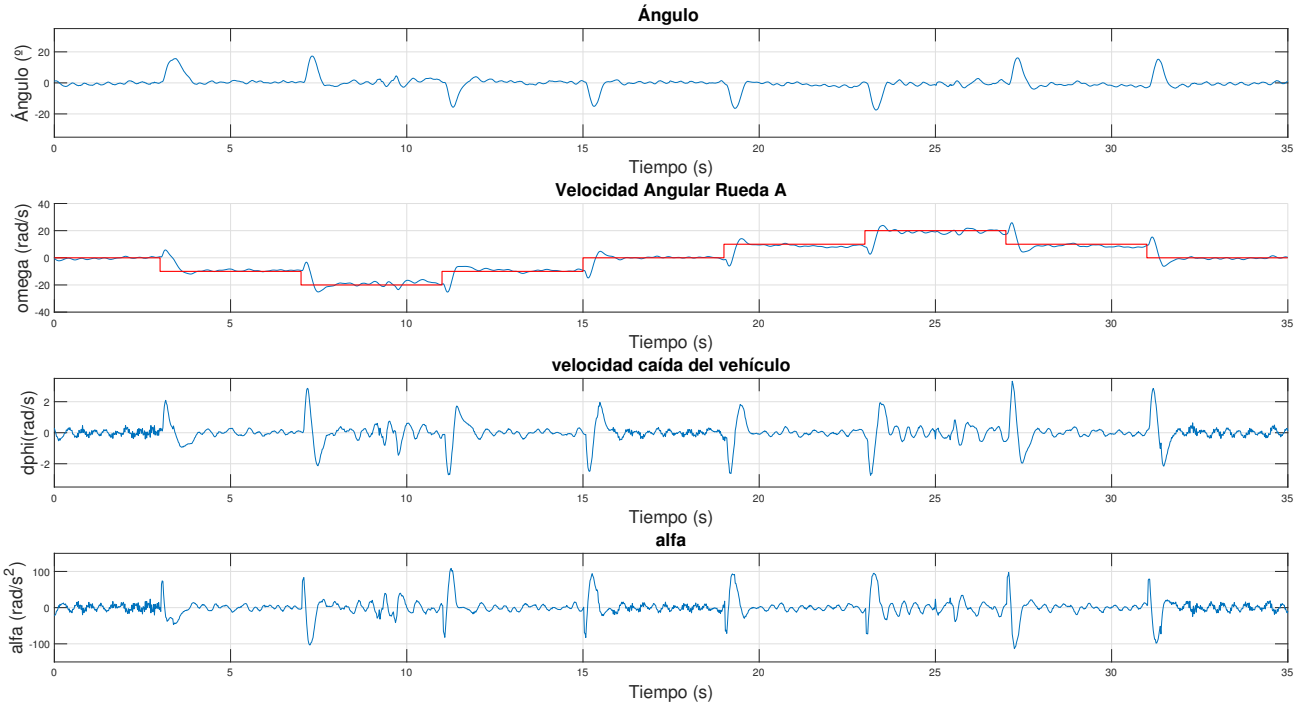


Figura 6.22 LQRI: Variables ante ensayo de seguimiento de referencia cambiante en velocidad.

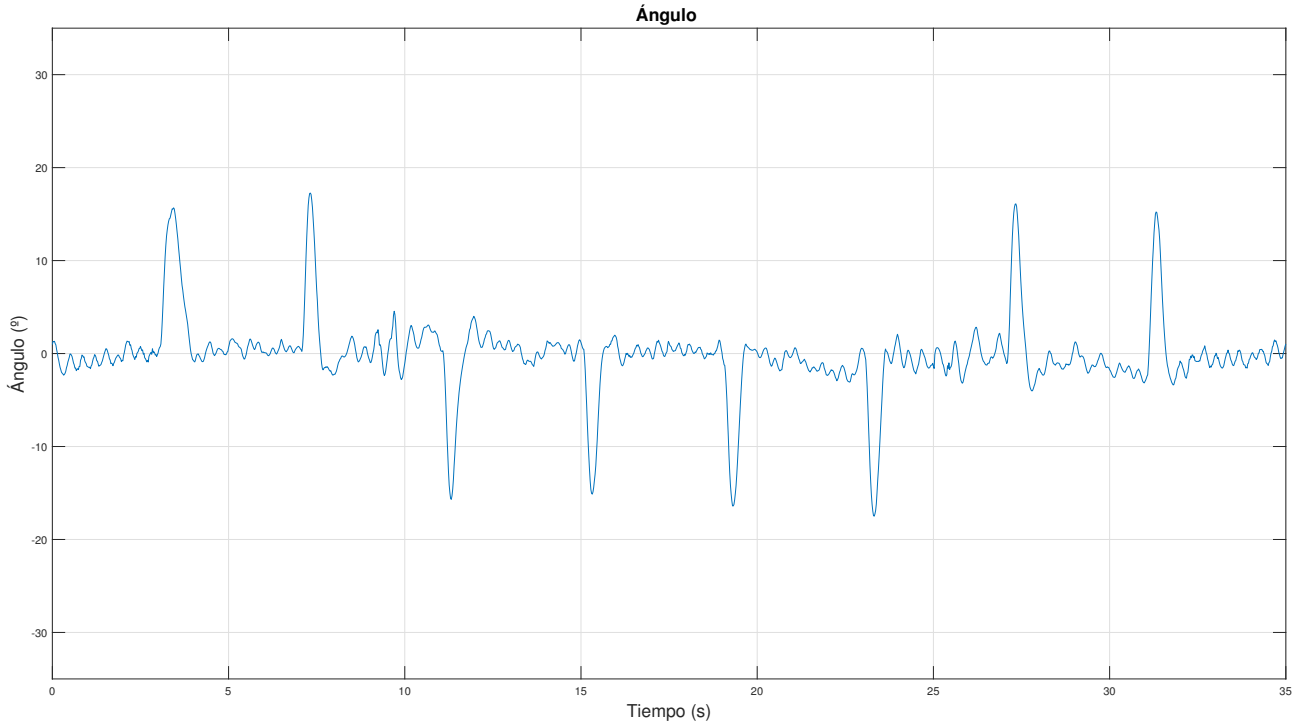


Figura 6.23 LQRI: Ángulo ante ensayo de seguimiento de referencia cambiante en velocidad.

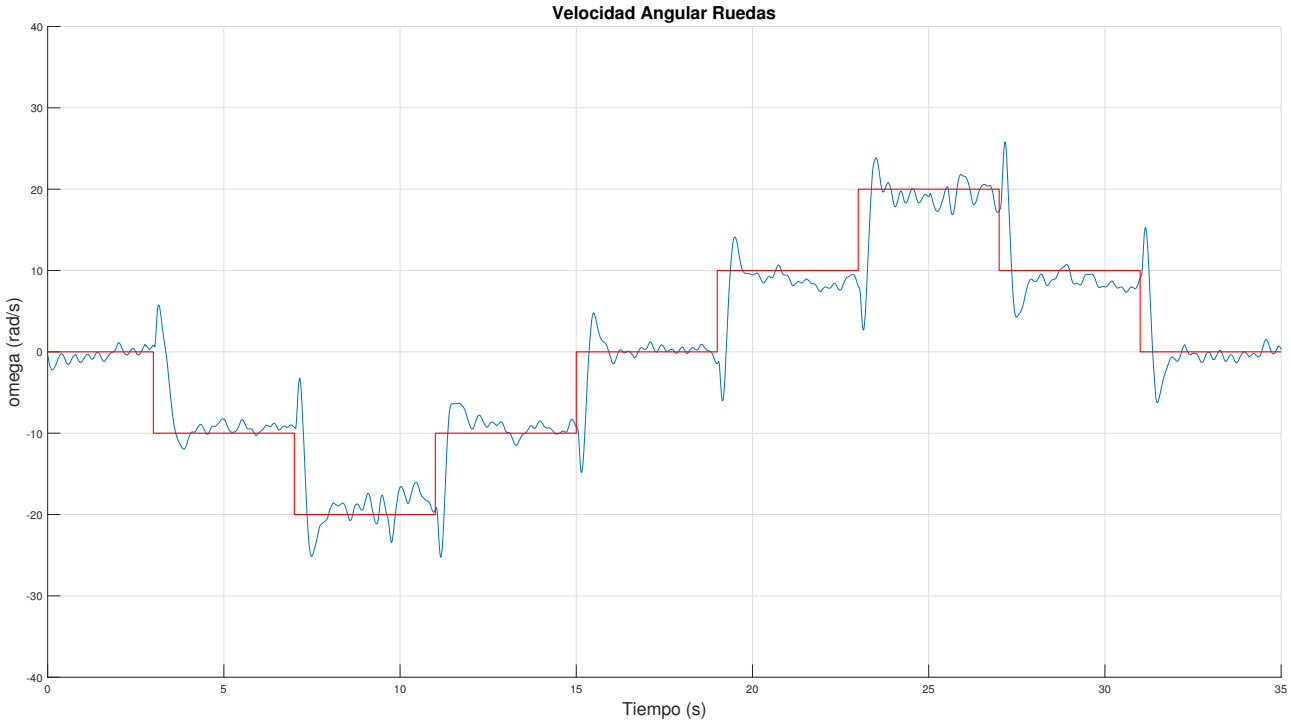


Figura 6.24 LQRI: Velocidad angular de las ruedas ante ensayo de seguimiento de referencia cambiante en velocidad.

6.2.4 Fuerza creciente en parte superior del vehículo

Este ensayo consiste en aplicar una fuerza creciente, en forma de empuje, sobre una cara del vehículo durante 4 segundos. Puede apreciarse que el vehículo se inclina hasta llegar a los 25°, momento en el cuál dejamos de aplicar la fuerza. Una vez que se deja de aplicar la fuerza, el sistema de control detecta que el vehículo no se encuentra en la referencia deseada y actuará la señal de control (aceleración angular de las ruedas) en sentido contrario a la fuerza aplicada para variar esta perturbación y volver a su posición de equilibrio.

El control LQRI presenta la adición del término integral que se encargará de integrar el error en velocidad. Por ello, una vez deje de aplicarse la fuerza, el vehículo volverá a la posición donde se empezó a aplicar dicha fuerza.

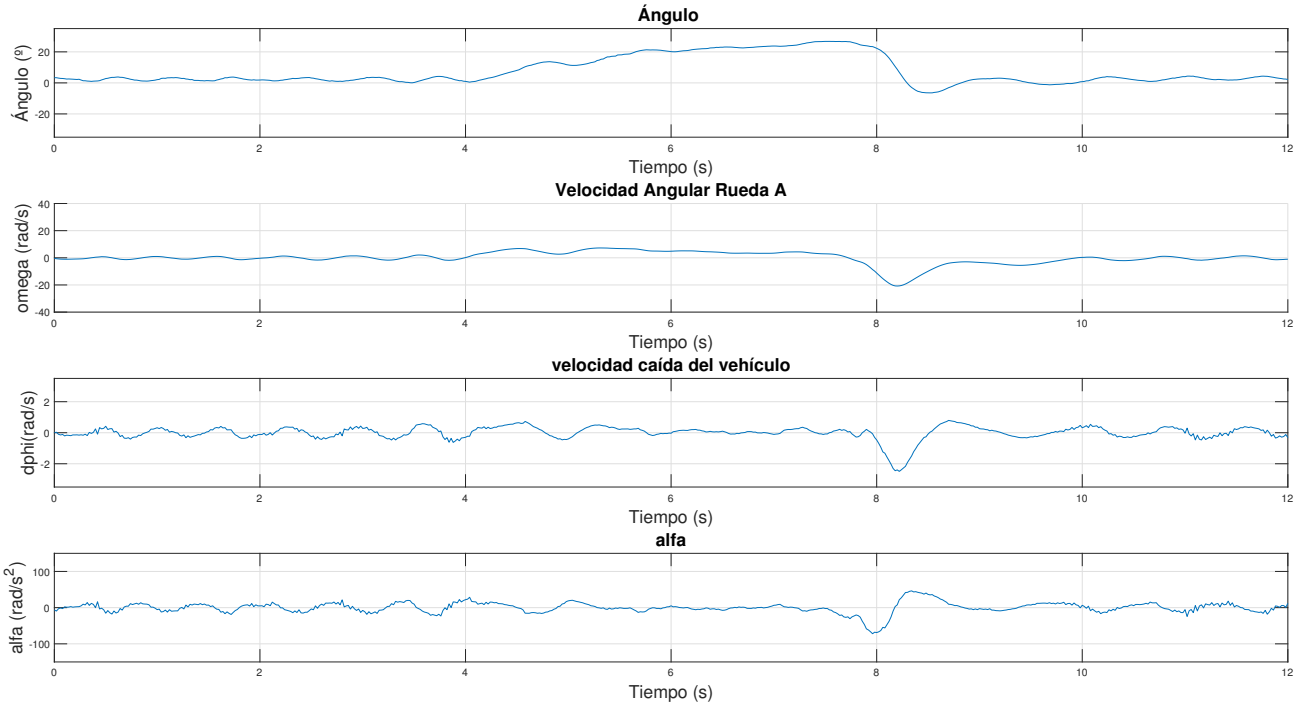


Figura 6.25 LQRI: Variables ante ensayo de fuerza creciente en parte superior del vehículo.

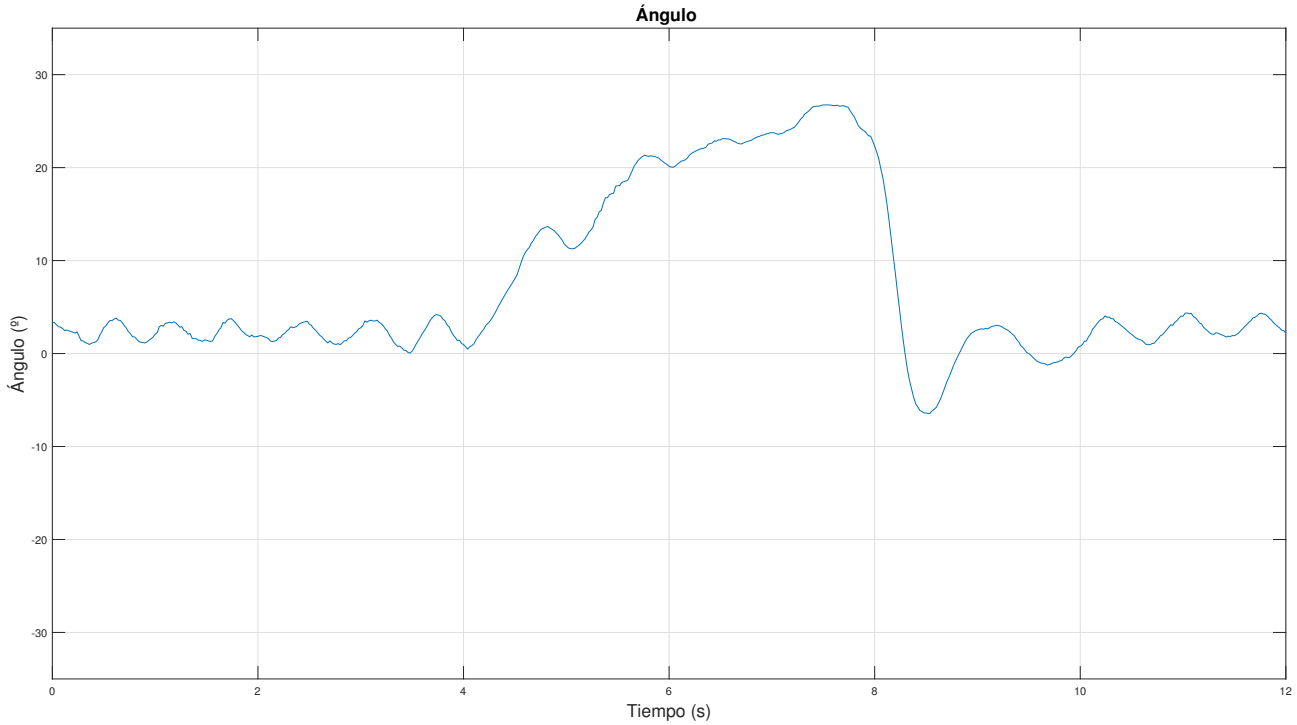


Figura 6.26 LQRI: Ángulo ante ensayo de fuerza creciente en parte superior del vehículo.

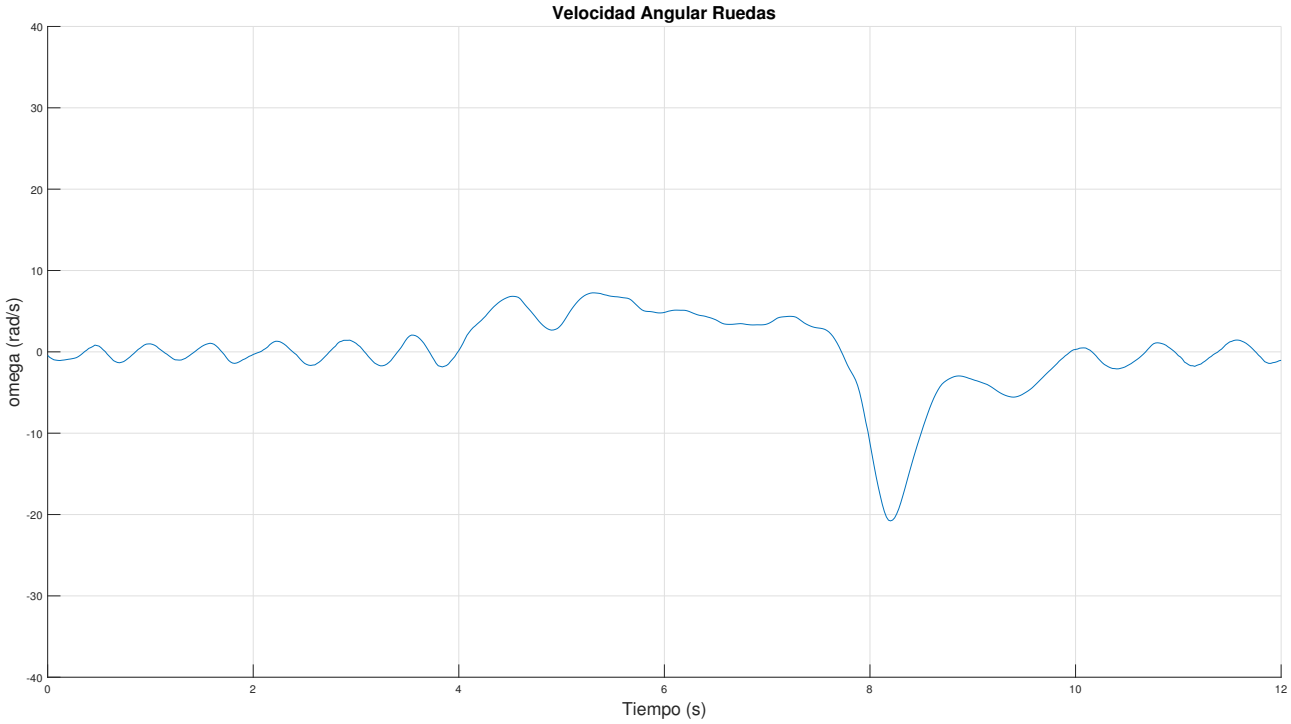


Figura 6.27 LQRI: Velocidad angular de las ruedas ante ensayo de fuerza creciente en parte superior del vehículo.

6.2.5 Ascenso y Descenso en plano inclinado control remoto

En este ensayo, se hizo subir al vehículo por un plano inclinado de 15°, mediante el uso del joystick, una vez suba la rampa se estabilizara durante unos cuantos segundos y se le hará bajar. Su resultados son similares a los explicados, para el mismo experimento, en el apartado anterior.

Este ensayo dura 50 segundos y se realizan 3 descensos por el plano inclinado. Se parte del vehículo al inicio de rampa y en el segundo 2 se varía la referencia de velocidad para hacer subir al vehículo por la rampa. Se aprecia que la subida sigue siendo lenta, pues dura cerca de 4 segundos y el ángulo que alcanza en este caso, es superior a 20°, para compensar la fuerza de la gravedad. Se estabiliza entorno a 0° en lo alto de la rampa durante 6 segundos y en el segundo 11 se dirige el vehículo para que descienda la cuesta. Se repite la operación de subida en los segundos 20 y 33.

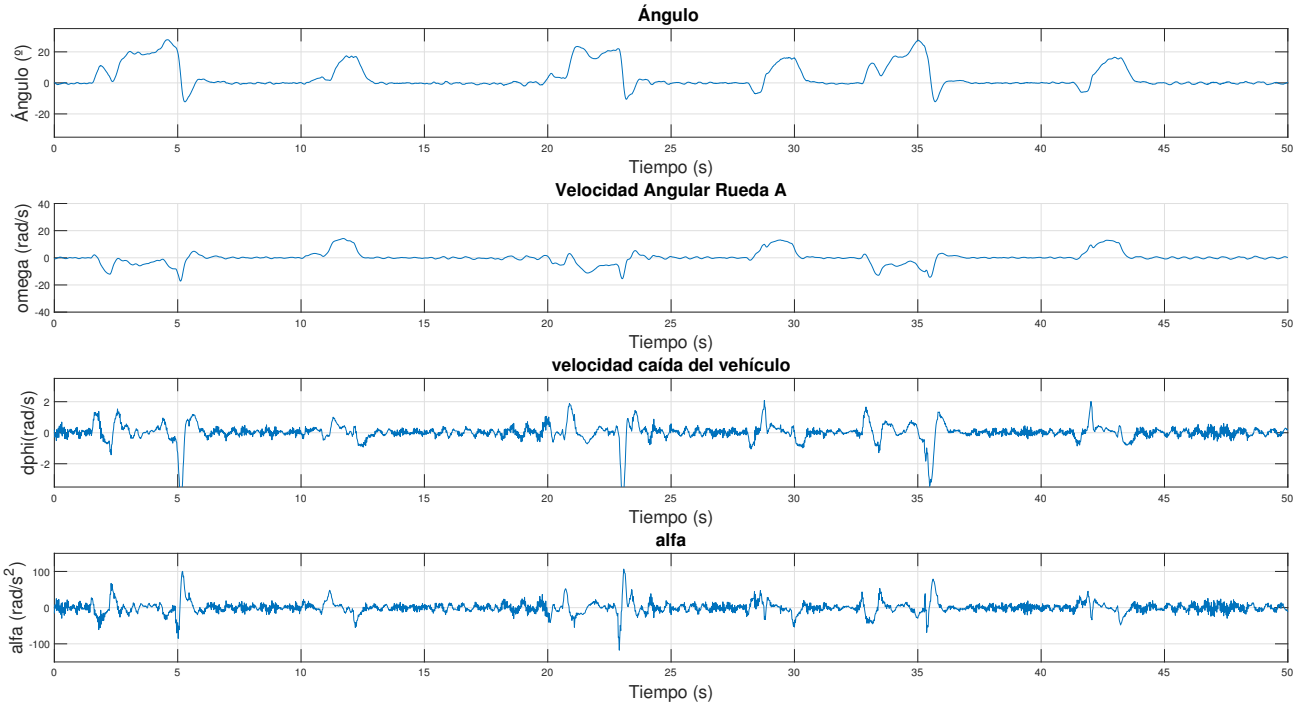


Figura 6.28 LQRI: Variables ante ensayo en plano inclinado. Control remoto.

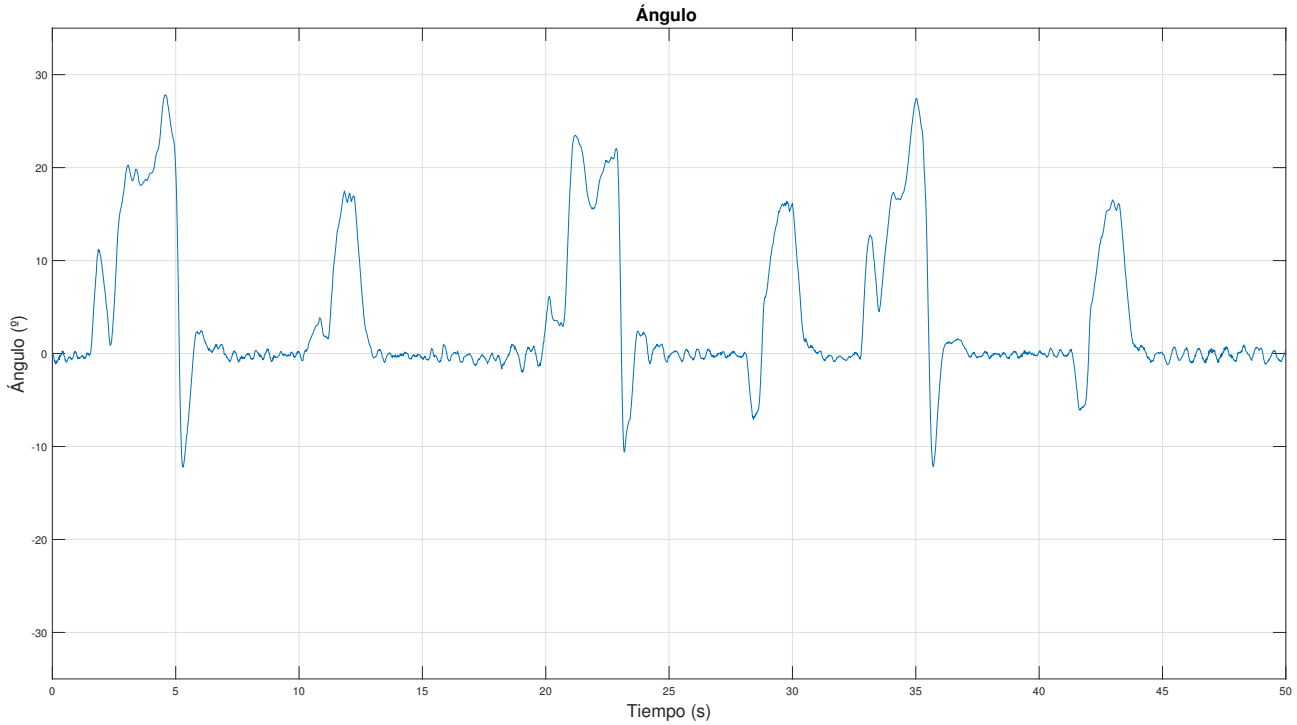


Figura 6.29 LQRI: Ángulo ante ensayo en plano inclinado. Control remoto.

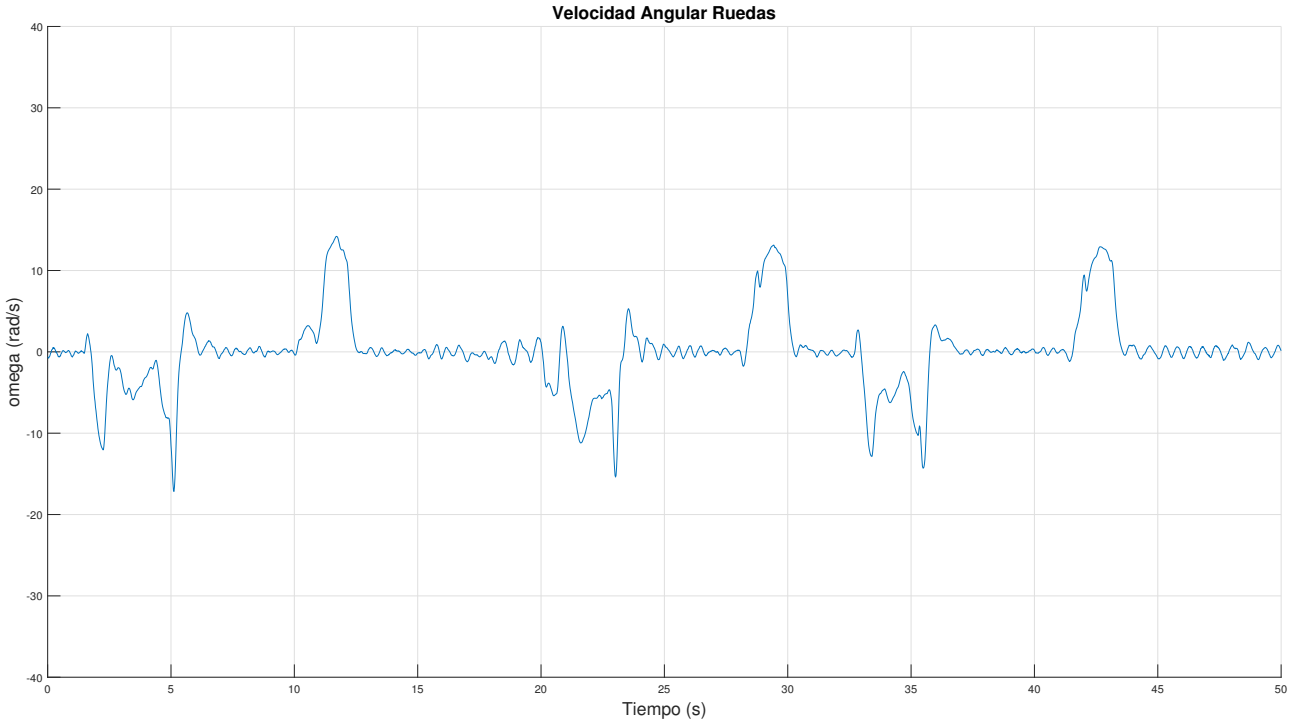


Figura 6.30 LQRI: Velocidad angular de las ruedas ante ensayo en plano inclinado. Control remoto.

6.3 Control MPC

El control MPC que se muestra en las siguientes gráficas, mejora sustancialmente el control del vehículo, haciéndolo más suave, corrigiendo los errores en régimen permanente, alcanzando mayores velocidades.

6.3.1 Seguimiento de referencia nula en velocidad de las ruedas

Como se ha visto, este experimento consiste en dejar al vehículo quieto sobre un plano horizontal durante 50 segundos. En los controladores anteriores, se apreciaba como aparecían pequeñas variaciones del ángulo no superiores a ningún momento a 2°. En el MPC el vehículo queda prácticamente parado, siendo la amplitud de las pequeñas oscilaciones entorno a 0.5° y no superando en ningún momento 1°. Por todo ello, este controlador supera con creces a los dos anteriores en el ensayo de referencia de velocidad nula.

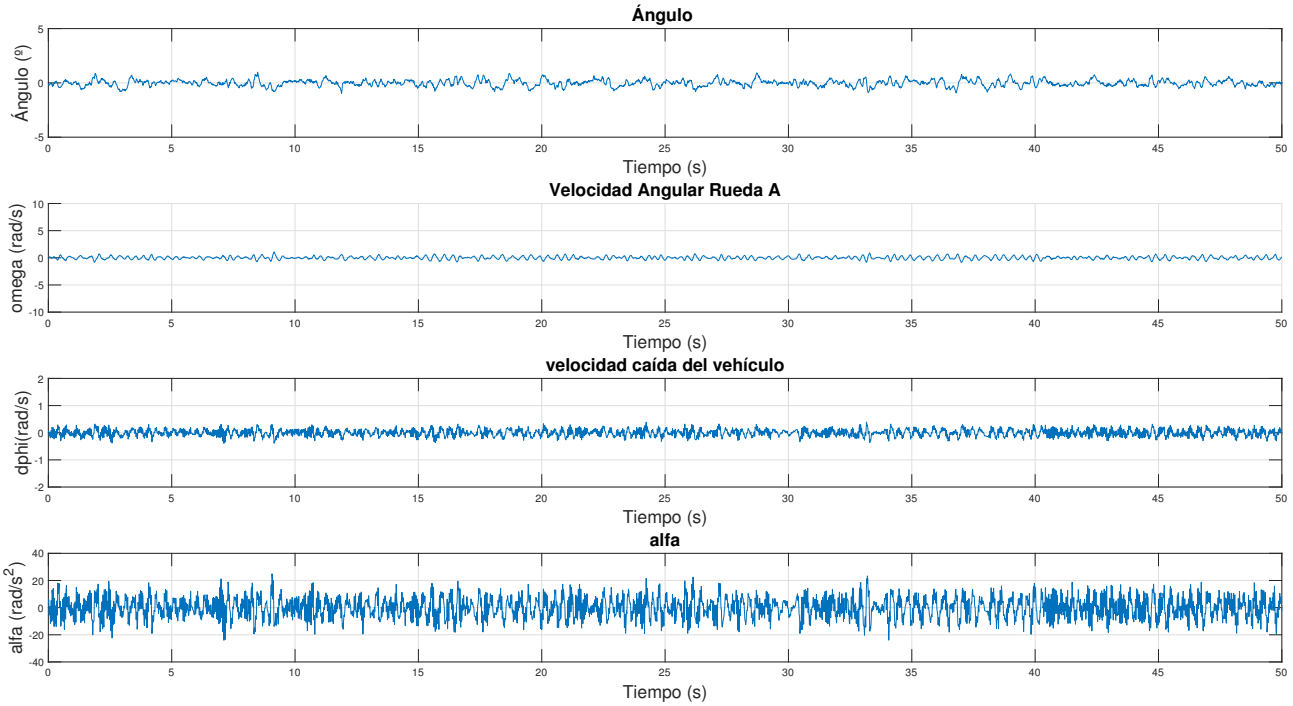


Figura 6.31 MPC: Variables ante ensayo de seguimiento de referencia nula en velocidad.

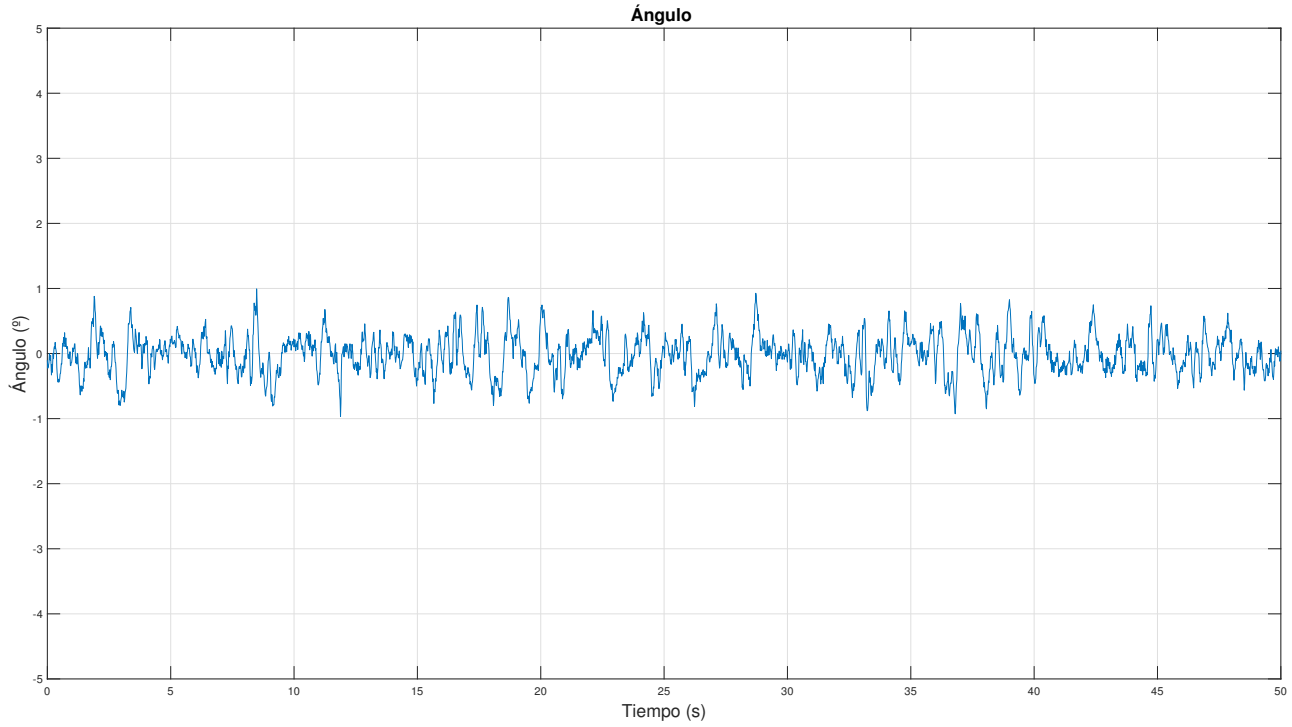


Figura 6.32 MPC: Ángulo ante ensayo de seguimiento de referencia nula en velocidad.

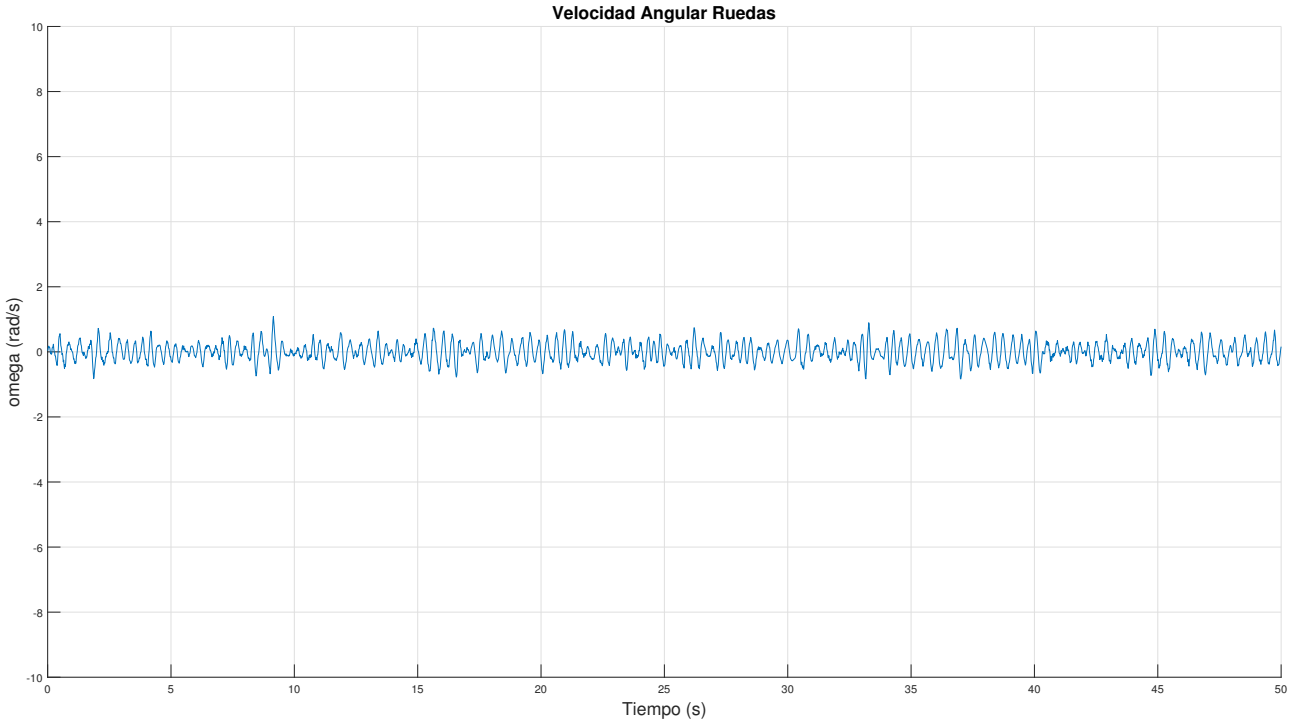


Figura 6.33 MPC: Velocidad angular de las ruedas ante ensayo de seguimiento de referencia nula en velocidad.

6.3.2 Perturbaciones tipo pulso

Este ensayo consiste en dejar al vehículo quieto que estabilice entorno a su posición de equilibrio, con una referencia nula en sus variables. Se aprecia que el vehículo alcanza menor inclinación que la alcanzada en los dos controladores anteriores, que recordemos superaba los 20°. . El vehículo presente una gran resistencia a la variación de su posición. Teniendo que dar golpes secos de mayor fuerza, para alcanzar menor inclinación. Puede apreciarse que la aceleración llega a saturar al valor que se ha indicado como restricción ($80 rad/s^2$). El control MPC es más robusto que los controladores anteriores.

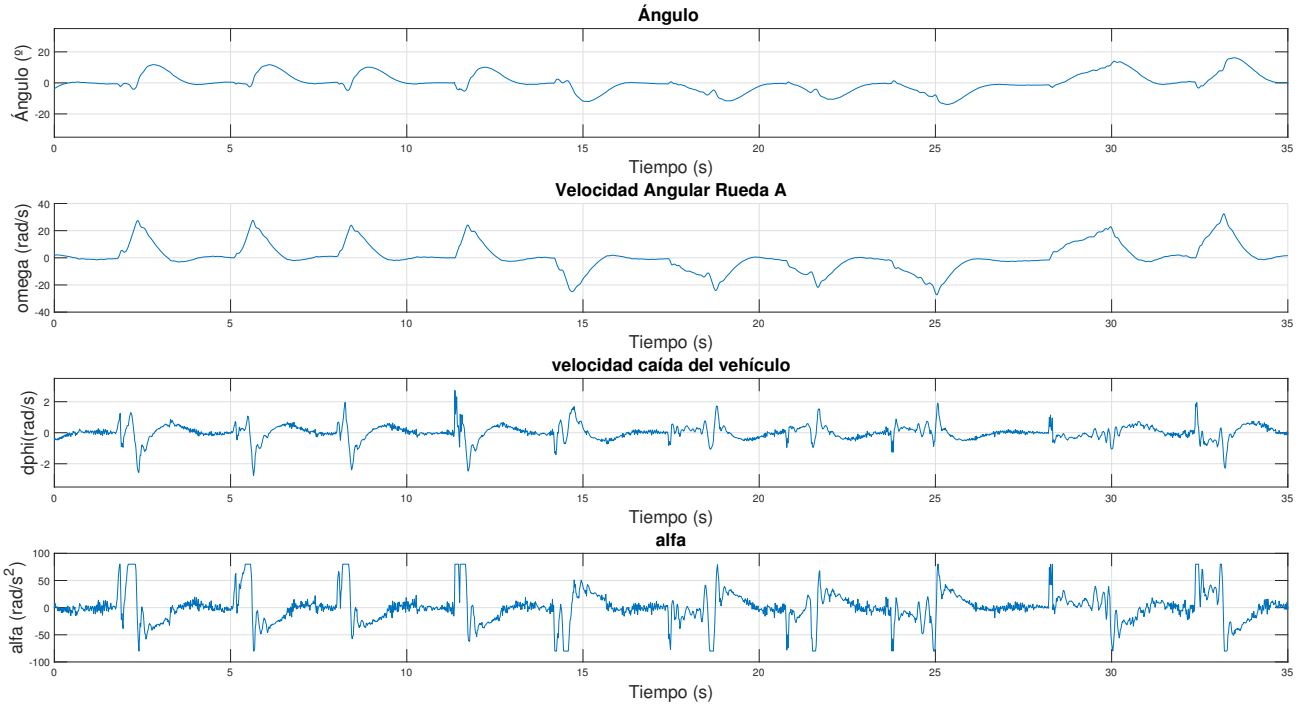


Figura 6.34 MPC: Variables ante ensayo de perturbaciones tipo pulso.

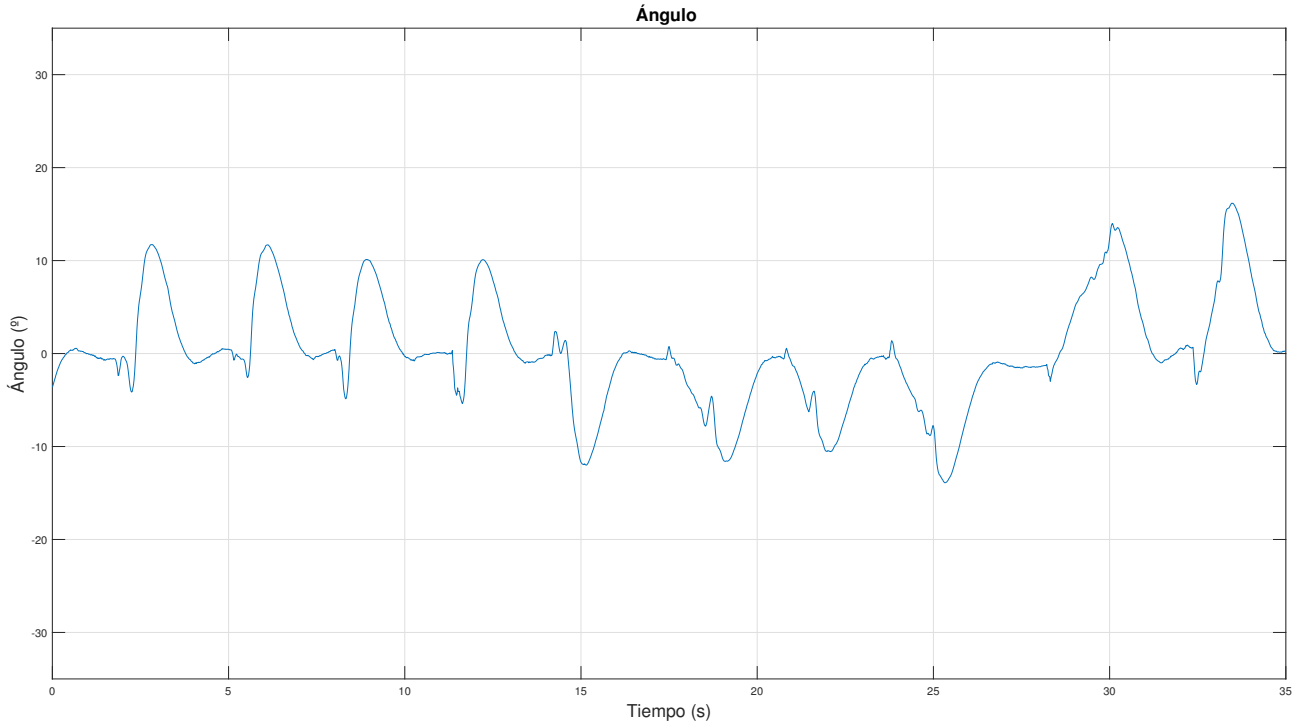


Figura 6.35 MPC: Ángulo ante ensayo de perturbaciones tipo pulso.

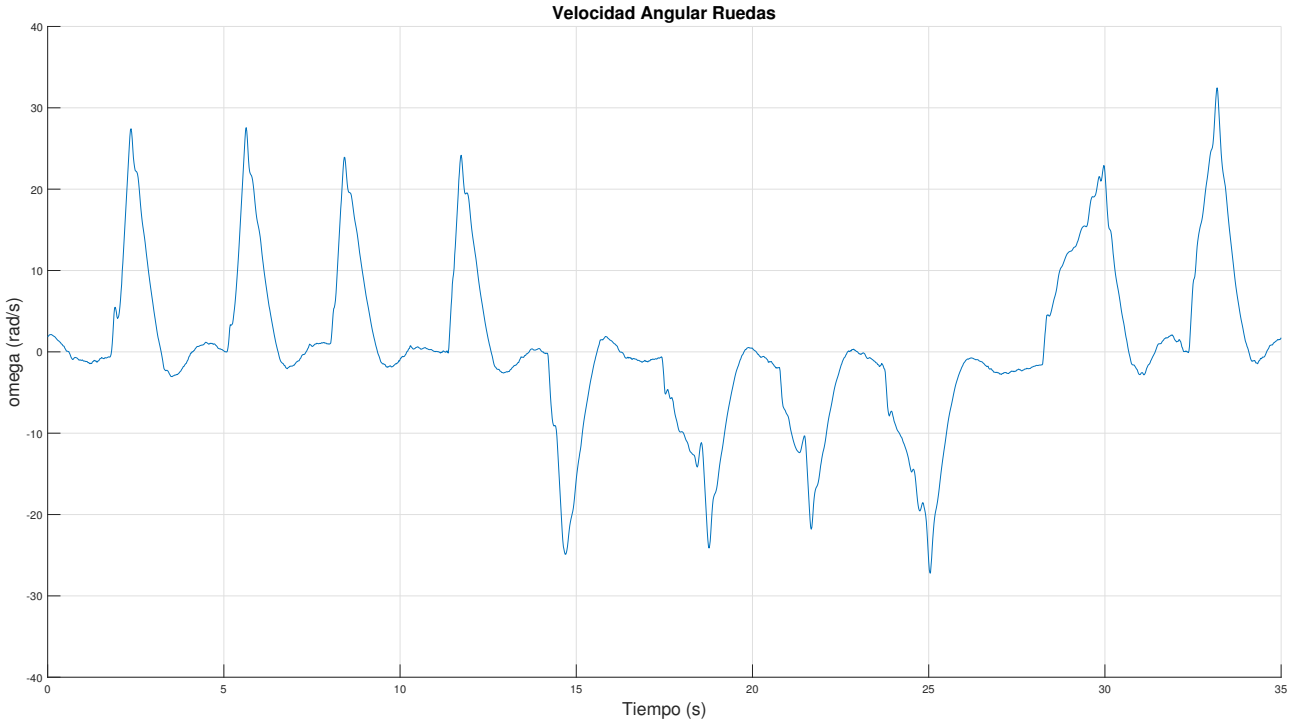


Figura 6.36 MPC: Velocidad angular de las ruedas ante ensayo de perturbaciones tipo pulso.

6.3.3 Seguimiento ante cambios de referencia de velocidad de las ruedas

En este ensayo se ha producido un cambio en la referencia de velocidad de las ruedas. Este cambio en las referencias de velocidad se produce cada 4s y es de 30 rad/s ascendente y -30rad/s descendente. Nótese que en los dos controladores anteriores, los cambios eran de 10 rad/s para evitar que una variación alta hiciera que el vehículo se cayera.

Se parte de una velocidad nula hasta llegar a 30 rad/s seguidamente se cambia el sentido de movimiento hasta llegar a los -30 rad/s y volver a 0 rad/s, así repetidas veces.

La variación del ángulo que se produce cuando se cambia la referencia es menor que en los casos anterior, lo cual explica que pueda realizarse saltos de referencia mayores. En cada cambio de referencia el vehículo llega a inclinarse en torno a 18°.

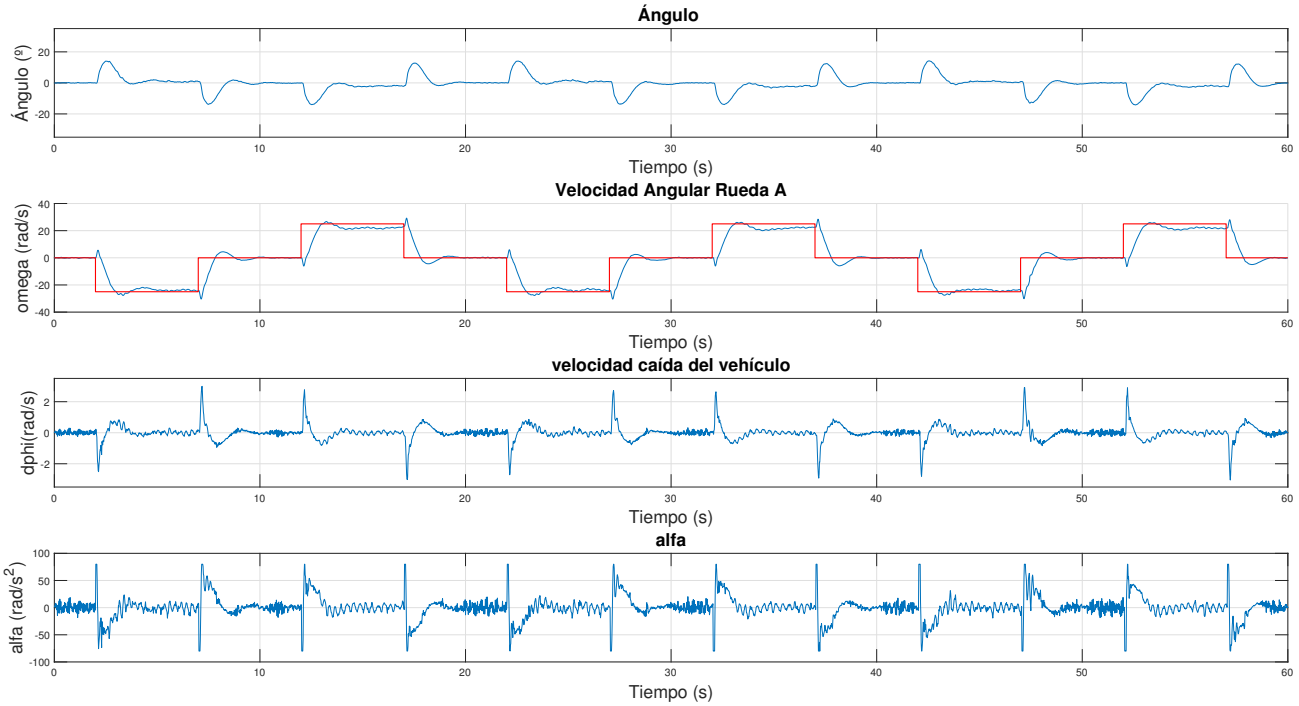


Figura 6.37 MPC: Variables ante ensayo de seguimiento de referencia cambiante en velocidad.

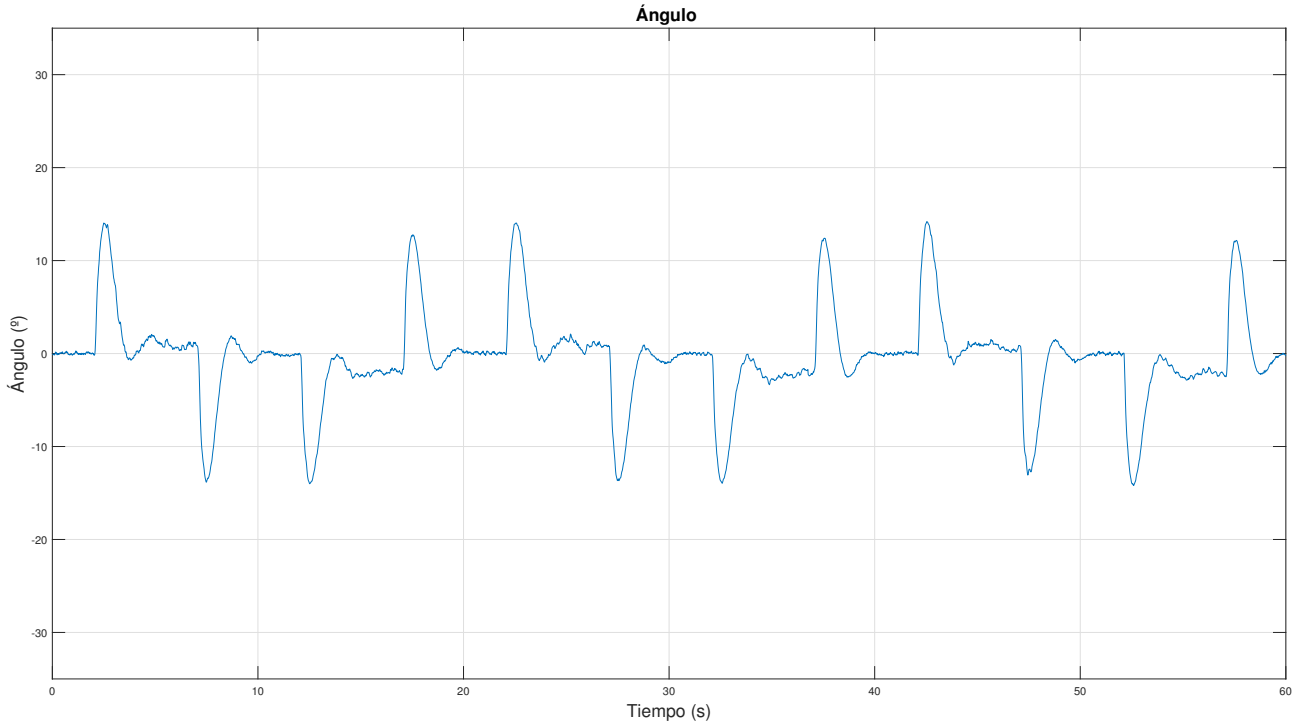


Figura 6.38 MPC: Ángulo ante ensayo de seguimiento de referencia cambiante en velocidad.

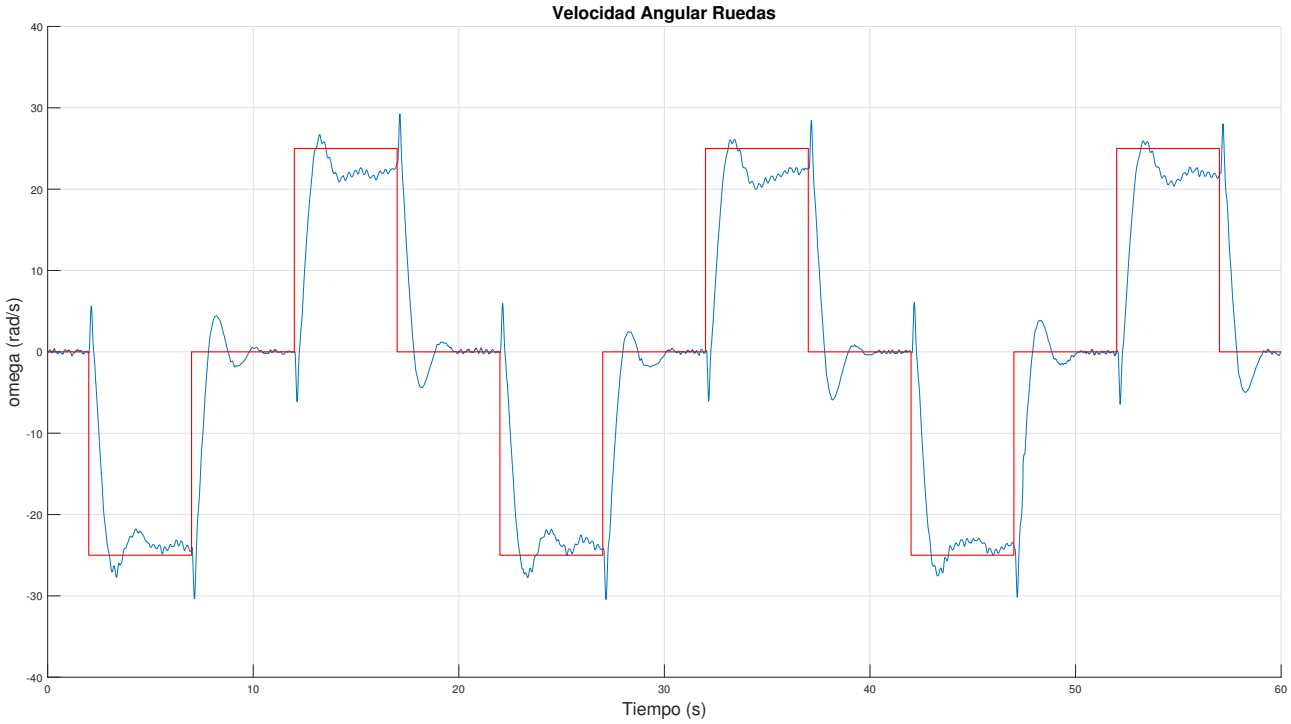


Figura 6.39 MPC: Velocidad angular de las ruedas ante ensayo de seguimiento de referencia cambiante en velocidad.

6.3.4 Ascenso y Descenso en plano inclinado

La explicación de este experimento es la que puede encontrarse su apartado análogo del control LQR. Haremos ascender el vehículo por un plano inclinado de 20° y lo dejaremos caer, apreciando que no llega a perder su estabilidad. El ensayo se ha repetido 4 veces durante 50 segundos.

En el control LQR, el vehículo llegaba a alcanzar un ángulo de inclinación cercano a los 30° en el descenso. En el caso del control MPC, el ángulo de inclinación que alcanza es cercano a la mitad, pues como se ha comentado en el ensayo tipo pulso, el control se opone fuertemente a la perturbación del estado de equilibrio.

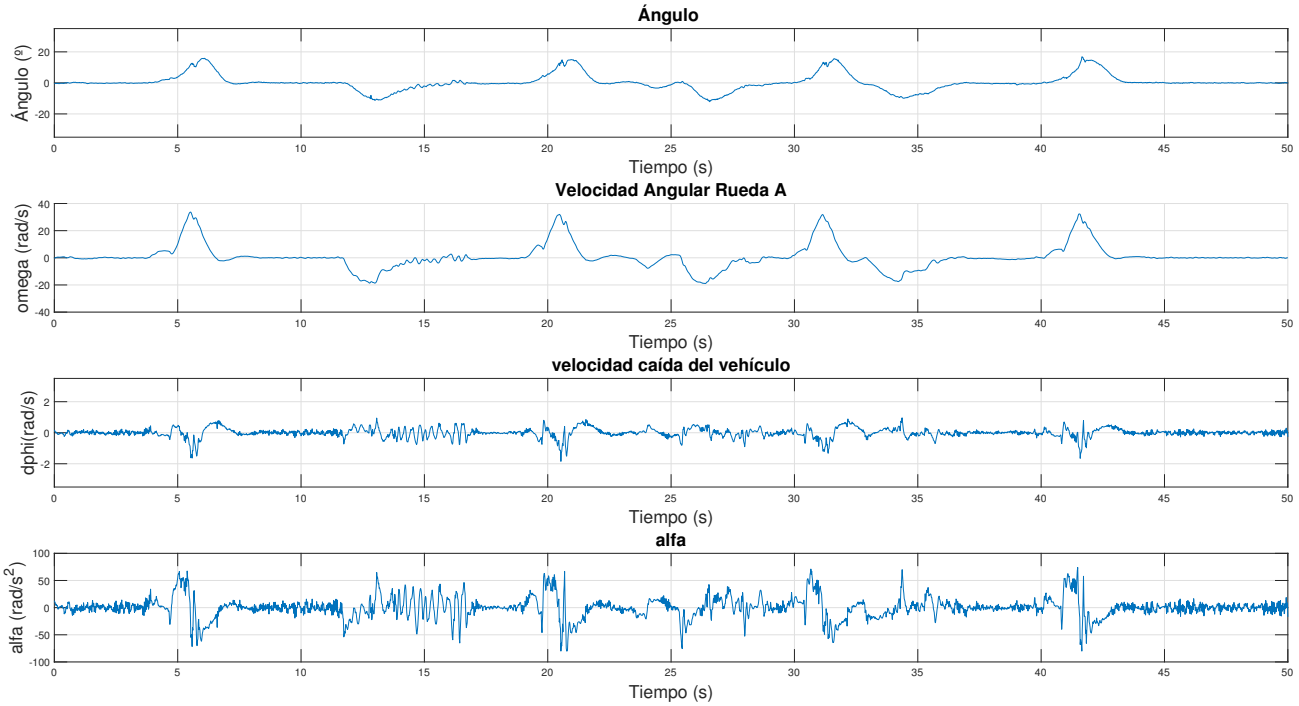


Figura 6.40 MPC: Variables ante ensayo en plano inclinado.

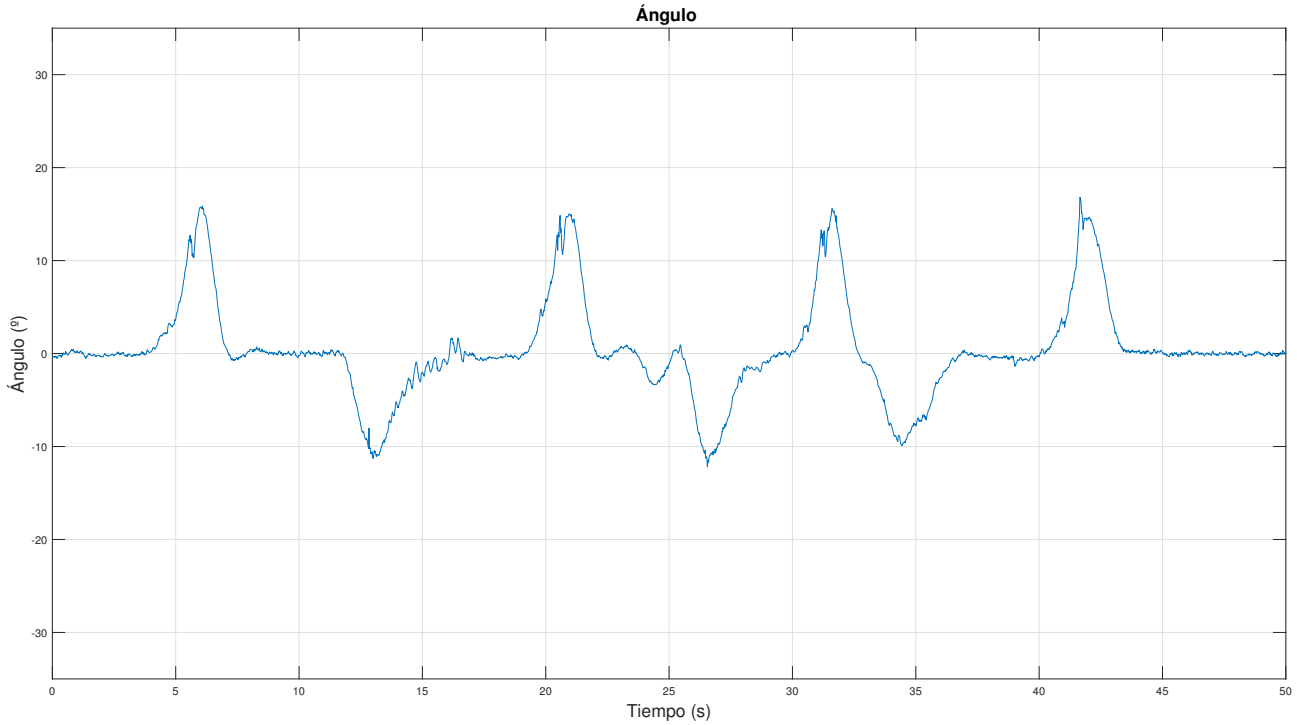


Figura 6.41 MPC: Ángulo ante ensayo en plano inclinado.

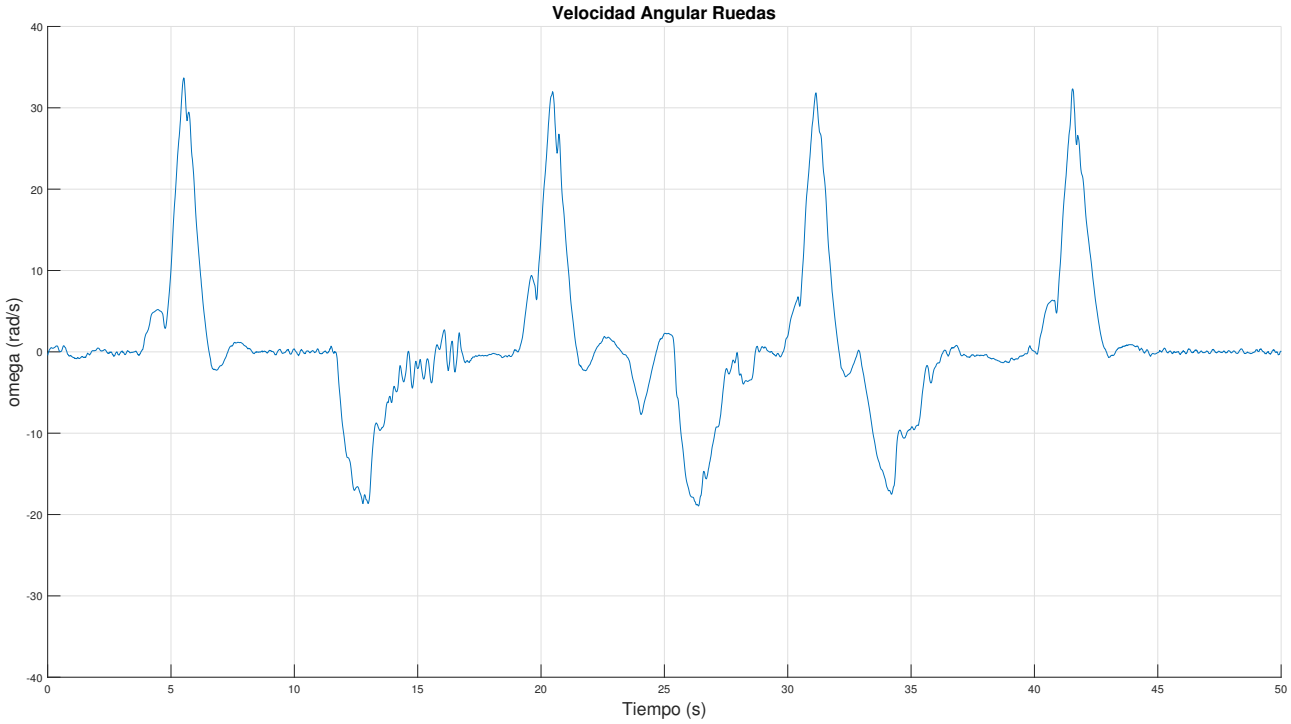


Figura 6.42 MPC: Velocidad angular de las ruedas ante ensayo en plano inclinado.

6.3.5 Ascenso y Descenso en plano inclinado control remoto

La explicación de este experimento es la que puede encontrarse su apartado análogo del control LQR. Con el control remoto, se hace ascender el vehículo por un plano inclinado de 15°, se estabiliza entrono 0° durante un cierto tiempo y lo dejaremos caer, apreciando que no llega a perder su estabilidad. El ensayo se ha repetido 4 veces durante 70 segundos.

Se parte del vehículo al inicio de rampa y en el segundo 4 se varía la referencia de velocidad para hacer subir al vehículo por la rampa. Se aprecia que la subida es más lenta que en los ensayos anteriores, pues dura cerca de 6 segundos y el ángulo que alcanza es cercano a 20°, para compensar la fuerza de la gravedad. Se estabiliza entorno a 0° en lo alto de la rampa durante 6 segundos y en el segundo 17 se dirige el vehículo para que descienda la cuesta. Se repite la operación de subida en los segundos 25, 43 y 58.

En las siguientes gráficas, se aprecia que el ángulo es al contrario que en el control LQR y LQRI, esto es debido a que se ha hecho subir al vehículo en dirección contrario a los casos anteriores.

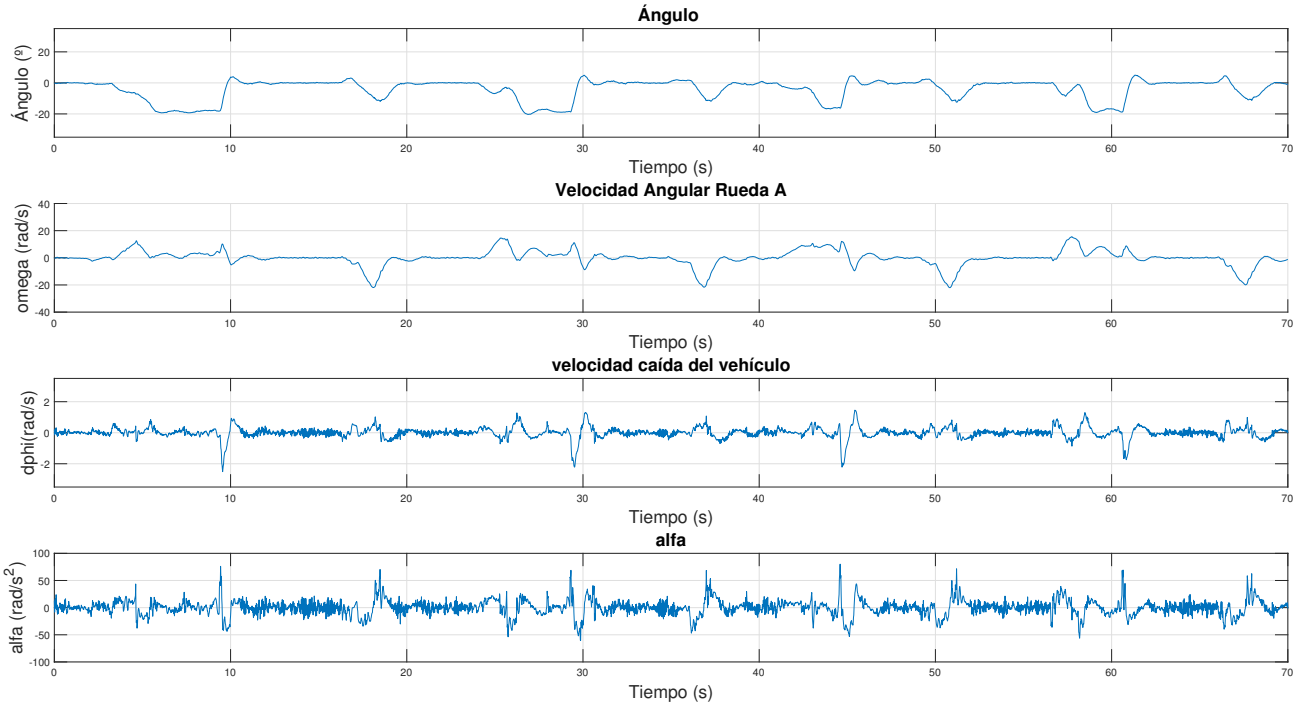


Figura 6.43 MPC: Variables ante ensayo en plano inclinado. Control remoto.

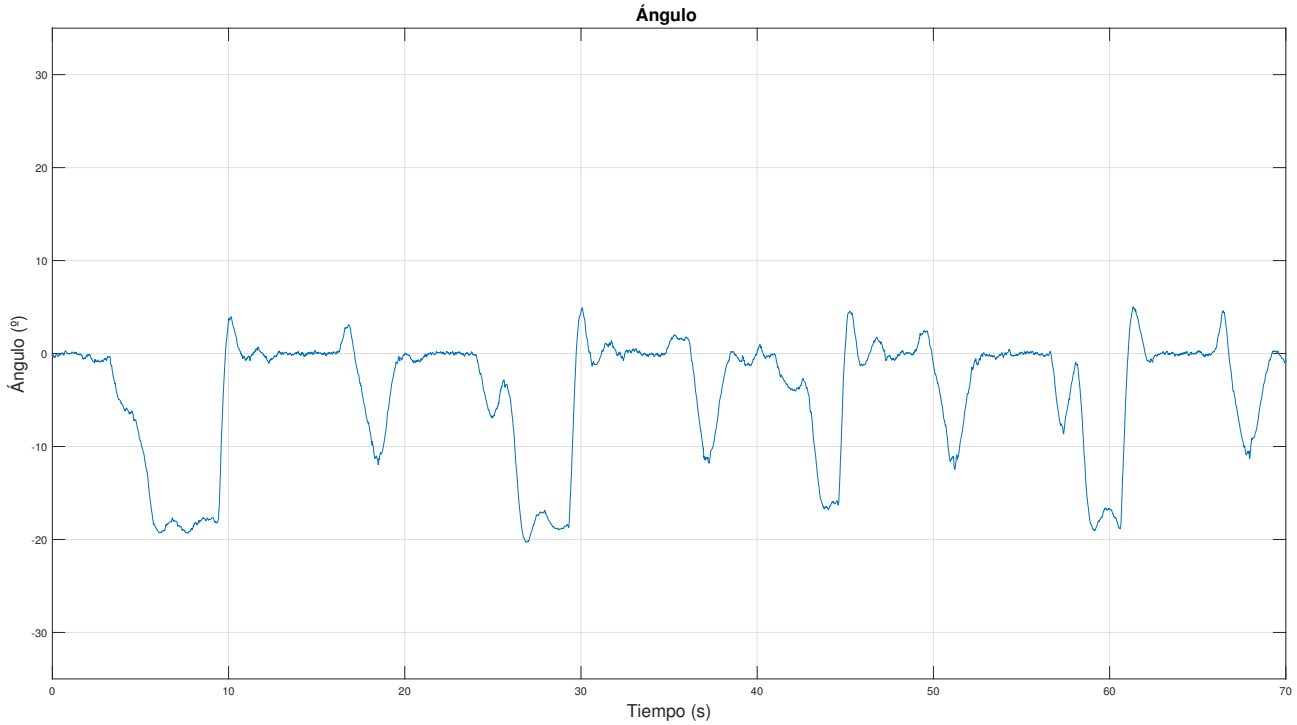


Figura 6.44 MPC: Ángulo ante ensayo en plano inclinado. Control remoto.

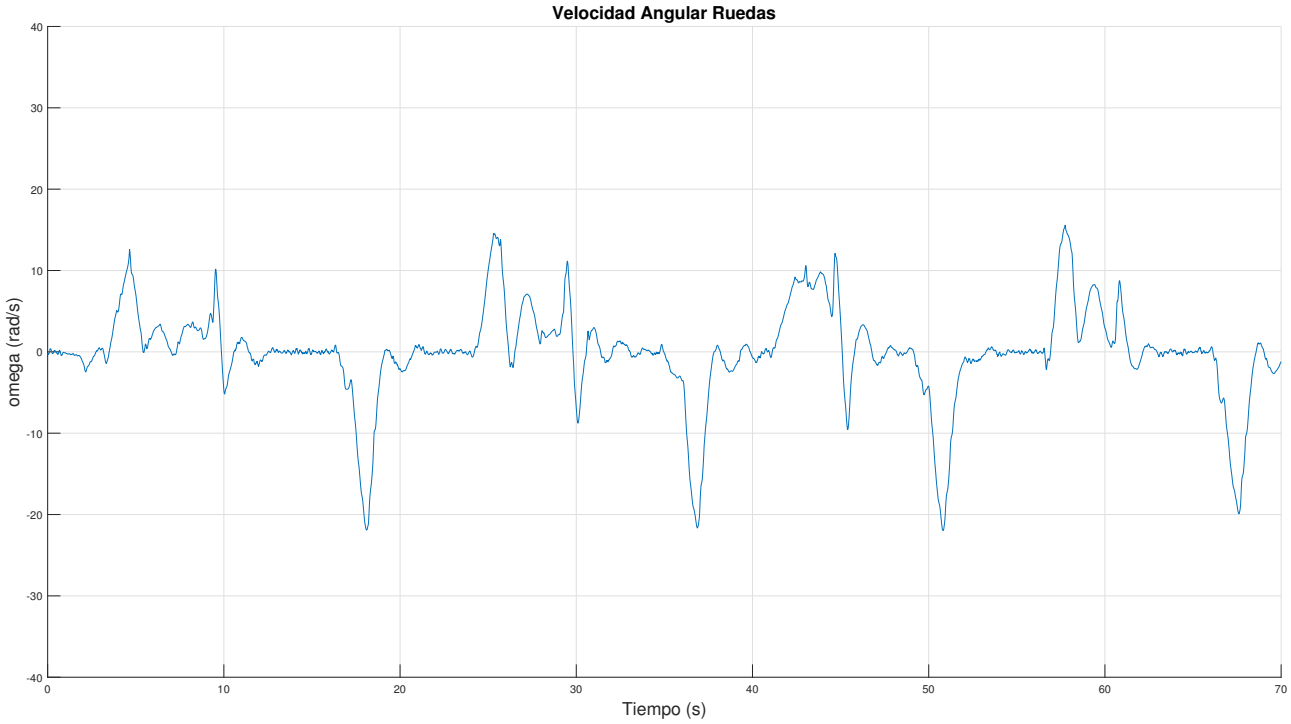


Figura 6.45 MPC: Velocidad angular de las ruedas ante ensayo en plano inclinado. Control remoto.

7 Conclusiones

El objetivo de este trabajo ha sido el diseño, fabricación, montaje y control remoto de un vehículo balancín sobre dos ruedas basado en la configuración del péndulo invertido. Dicho trabajo parte de una línea de trabajos iniciada hace algunos años en el departamento de Ingeniería Automática de la Escuela Técnica Superior de Ingeniería de Sevilla. Pretende ser una ampliación y mejora de los trabajos anteriores.

Con la adición de la microconputadora Rasberry Pi 3, se pretende dar un giro a las técnicas de control. La adición de este dispositivo nos abre un amplio horizonte de posibilidades. La potencia de cálculo nos permite implementar técnicas de control avanzadas como el control MPC, imposible de implementar en un controlador Arduino debido a los costes computacionales del mismo. Sin embargo ha sido necesario seguir contando con el controlador Arduino, pues la RaspberryPi, aunque cada día está más enfocada a aplicaciones robóticas, aún dista mucho de Arduino en este sentido. La necesidad de dotar de giro al vehículo y por tanto, de dos canales PWM a diferente frecuencia para implementar la velocidad de las ruedas, hicieron necesario el uso de Arduino. Si la RPi incorporase en sus modelos venideros un segundo canal PWM podría prescindirse del uso de Arduino. Aún así, la Rpi ha dado muy buenos resultados en este trabajo para la aplicación de controladores, dada su gran velocidad de cálculo y facilidad de comunicación con Arduino.

Los resultados obtenidos ante los diferentes ensayos y con los diferentes controladores que se han implementados, han sido bastante satisfactorios. A modo de resumen:

- Para el ensayo ante referencias nulas, los mejores resultados han sido los obtenidos con el control LQRI y MPC consiguiendo que el vehículo se mantenga en su posición de equilibrio prácticamente quieto sin apenas oscilaciones.
- Para el ensayos ante perturbaciones tipo pulso, el control LQRI adquiría mayor inclinación volviendo rápidamente a su posición de equilibrio. En este experimento el control MPC presentaba una gran resistencia a variar su posición, teniendo que dar golpes más fuertes pero aún así adquiría un menor ángulo.
- Para el ensayo ante variaciones en la referencia de velocidad, el control MPC nos permite dar saltos mayores en velocidad sin que se produzcan oscilaciones que tumben el vehículo. En las diferentes gráficas, se puede apreciar que con el control MPC se hacen variaciones de $\pm 25(rad/s)$, inclinándose el vehículo entorno a 15°, mientras que con el control LQR, para variaciones de $\pm 10(rad/s)$ el vehículo se inclinaba entorno a 18°.
- Para el ensayo en un plano inclinado, se llevaron a cabo dos tipo experimentos. El primero, consistía en forzar con la mano a que el vehículo subiese la cuesta, mientras que en el segundo, la subida se realizaba vía remota con un joystick. En el movimiento forzado, el vehículo se oponía a la fuerza, empujando hacia ella (hacia la base de la rampa), mientras que el segundo caso, la oposición era a la fuerza de la gravedad (Hacia lo alto de la rampa). En el

movimiento de bajada, el control LQR adquiere una mayor inclinación que el control MPC que, al igual que el ensayo ante perturbaciones tipo pulso, se opone fuertemente a la variación de su posición.

A pesar de los buenos resultados, el trabajo queda lejos de ser perfecto, pues admite numerosas mejoras y líneas de investigación futuras. Como hemos dicho, la Raspberry Pi nos permite abrir un horizonte de nuevas posibilidades como puede ser la incorporación de la PiCam. La PiCam es una pequeña cámara que nos daría la posibilidad de convertir nuestro vehículo Segway en un robot completamente autónomo que fuera procesando, en tiempo real, las imágenes que recibe de su entorno y actuase en consecuencia, pudiendo transmitir estas imágenes vía stremming. Por ejemplo, permitir que el vehículo se desplazara hacía un objetivo, mediante el reconocimiento de formas o colores.

En trabajos futuros, deberá modificarse la PCB diseñada para añadir dispositivos que no se tuvieron en cuenta en su diseño y fueron necesarios añadir a lo largo del trabajo, como puede ser el módulo bluetooth de Arduino o un pequeño ventilador que disipe todo el calor que generan las placas electrónicas. La Rpi y el regulador de tensión adquirían tal temperatura que se reiniciaba cada pocos segundos, con la adición de un ventilador axial y de aletas, desaparecieron los citados problemas de temperatura.

El diseño del vehículo debe mejorarse, haciéndolo algo más ligero, pues el elevado peso del mismo (1115 gr) impide que adquiera una mayor inclinación.

Al utilizar unos motores de mayor par, los controladores EasyDriver quedan algo cortos, investigar la existencia de otro tipo de controladores para motores paso a paso que sean capaces de proporcionar mayor par a los motores, lo que haría aumentar la fuerza de los mismos y por tanto permitiría una mayor inclinación y una mayor velocidad.

Bibliografía

[1] *Sistema operativo en tiempo real*, https://xenomai.org/documentation/xenomai-2.4/html/xenomai/.

[2] Brian Schmalz, *Easydriver stepper motor driver*, https://www.schmalzhaus.com/EasyDriver/.

[3] C.Gonzalez, I.Alvarado, and D.Muñoz La Peña., *Low cost two-wheels self-balancing robot for control education.*, IFAC PapersOnLine (2017.).

[4] José Antonio Borja Conde, *Desarrollo de un robot autoequilibrado basado en arduino con motores paso a paso*, 2018.

[5] FAIRCHILD, *Datasheet ka78txx*, https://pdf1.alldatasheet.es/datasheet-pdf/view/527328/FAIRCHILD/KA78T05.html.

[6] Nicolás Cortés Fernández, *Diseño, fabricación, montaje, estudio dinámico, control y teleoperación de un vehículo tipo péndulo invertido sobre dos ruedas*, 2019.

[7] A. Ferramosca, D. Limon, I. Alvarado, and E. Camacho, *MPC for tracking with optimal closed-loop perdormance*, 2009.

[8] Cecilia González González, *Mejora del software de un robot autoequilibrado de tipo péndulo invertido de bajo coste*, 2016.

[9] Guillermo Lehmann, *Instructivo para la distribución de componentes y diseño de pistas en placas de circuito impreso (pcb)*.

[10] InvenSense Inc, *Mpu-600 and mpu-6050 product specification revision 3.4*, https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[11] Rodríguez L. Arroyo S Jaramillo D, Loya.H, *Estabilización de un péndulo invertido aplicando MPC y LQR*, 2014.

[12] Pablo Kupra, Daniel Limon, and Teodoro Alamo, *Implementation of model predictive control in programmable logic controllers*, IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY (2020.).

[13] Victor Manuel Villalar Lara, *Control predictivo de un vehículo tipo segway*, 2017.

[14] Luis Llamas, *Determianr la orientación con arduino y el imu mpu-6050*, https://www.luisllamas.es/arduino-orientacion-imu-mpu-6050/.

[15] _____, *El bus i2c en arduino*, https://www.luisllamas.es/arduino-i2c/.

[16] _____, *Medir la inclinación con imu, arduino y filtro complementario*, https://www.luisllamas.es/arduino-i2c/.

[17] _____, *Qué son y cómo usar interrupciones en arduino*, https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/.

[18] Antonio J. Croche Navarro, *Vehículo autoequilibrado de tipo péndulo invertido de bajo coste*, 2014.

Códigos

1 Código Arduino

```
#include <Wire.h>
#include <math.h>
#include <SoftwareSerial.h>
#define DIRECCION_RPI 0x05 //ESTABLE LA DIRECCIÓN I2C

// Bluetooth
char comando;
int dtheta_r;
char NOMBRE[21] = "JOSE";
char BPS = '8'; // 1=1200, 2=2400, 3=4800, 4=9600, 5=19200, 6=38400, 7=57600, 8=115200
char PASS[5] = "1234"; // PIN O CLAVE de 4 caracteres numericos
SoftwareSerial mySerial(0, 1);
#define STX 0x02
#define ETX 0x03
#define ledPin 13
#define SLOW 750
#define FAST 250
byte cmd[8] = {0, 0, 0, 0, 0, 0, 0, 0};
byte buttonStatus = 0;
long previousMillis = 0;
long sendInterval = SLOW;
String displayStatus = "xxxx";

//DEFINO LA PREESCALA DE LOS TIMER 1 Y 2
#define PREESCALA_1 8
#define PREESCALA_2 256
//TIEMPO DE PULSO DE CADA TIMER
float T_PULS_1=0.0000000625*PREESCALA_1; //T=(1/16000000)*65535*Escala
float T_PULS_2=0.0000159375*PREESCALA_2; //T=(1/16000000)*255*Escala

//DEFINO FUNCIONES PARA ACTIVAR O DESACTIVAR PINES
#define CLR(x, y) (x&=(~(1<<y)))
#define CLRR(x, y, z) (x&=(~((1<<y)|(1<<z))))
#define SET(x, y) (x|=(1<<y))
#define SETT(x, y, z) (x|=(1<<y)|(1<<z))
#define SETTT(x, y, z, v) (x|=(1<<y)|(1<<z)|(1<<v))

//VARIABLES GLOBALES
float alfa=0; // VARIABLE QUE ALMACENA LA ACELERACIÓN QUE RECIBE DE LA RPI
float vec[2]={0,0}; // VECTOR QUE ALMACENA LOS VALORES DE LA VELOCIDAD
```

```
41 float alfa_M1=0; // VARIABLE QUE ALMACENA LA ACELERACIÓN A APLICAR, DESPUÉS DE
       FILTRAR SI ESTA ENTRE LOS LIMITES ESTABLECIDOS EN +- xx RAD/S^2. EVITANDO ASI EL
       OVF AL ENVIARSE UN DATO CORRUPTO
42 float flag=0;      // BANDERA QUE SE PONE A CERO CUANDO SE RECIBE UN DATO alfa DESDE LA
       RPI, UNA VEZ PROCESADO ESTE DATO SE PONE A UNO A LA ESPERA DE RECIBIR OTRO DATO
43 int comprueba;
44 int j=0;
45 float vel;   // VALOR DE LA VELOCIDAD MEDIA QUE LE ENVÍO A LA PI
46 float vel_M1, vel_M2;
47
48 // DECLARACION PINES MOTOR 1
49 #define STEP_MOTOR_1 2   // PORTD 4
50 #define DIR_MOTOR_1 6    // PORTD 5
51 int MS1_MOTOR_1 = 8 - 8 ; // PORTB 1
52 int MS2_MOTOR_1 = 9 - 8; // PORTB 2
53
54 // DECLARACION PINES MOTOR 2
55 #define STEP_MOTOR_2 4      // PORTD 7
56 #define DIR_MOTOR_2 5       // PORTD 6
57 int MS1_MOTOR_2 = 11 - 8;   // PORTB 3
58 int MS2_MOTOR_2 = 10 - 8;   // PORTB 2
59
60 // PARÁMETROS DEL LED
61 #define PIN 3
62
63 // CONEXIÓN BLUETOOTH
64 float giro_M1=0, giro_M2;
65
66 // DECLARACIÓN DE LA CANTIDAD DE PASOS POR REVOLUCIÓN DEPENDIENDO DE LA
       CONFIGURACIÓN
67 uint16_t npuls_min[]={700, 600, 600, 600};
68 uint16_t npuls_max[]={65535, 65535, 65535, 65535};
69 // NUMERO MAX Y MIN DE PASOS/SEGUNDO
70 float f_timer_min[]={1/(npuls_max[0]*T_PULS_1),1/(npuls_max[1]*T_PULS_1),1/(npuls_max[2]*T_PULS_1
       ),1/(npuls_max[3]*T_PULS_1)};
71 float f_timer_max[]={1/(npuls_min[0]*T_PULS_1),1/(npuls_min[1]*T_PULS_1),1/(npuls_min[2]*T_PULS_1)
       ,1/(npuls_min[3]*T_PULS_1)};
72 // NUMERO DE PASOS PARA DAR UNA VUELTA Y AVANCE DE CADA PASO
73 float avance_grados [4]={1.8,0.9,0.145,0.225};
74 float pasos[]={200,400,800,1600};
75 // FRECUENCIAS MAX Y MINIMAS, Nº MAX Y MINIMOS DE VUELTAS POR SEGUNDO.
       EXPERIMENTALMENTE LA VEL MAX 3000º/S
76 float f_motor_max[]={f_timer_max[0]/pasos[0],f_timer_max[1]/pasos[1], f_timer_max[2]/pasos[2],
       f_timer_max[3]/pasos[3]};
77 float f_motor_min[]={f_timer_min[0]/pasos[0], f_timer_min[1]/pasos[1], f_timer_min[2]/pasos[2],
       f_timer_min[3]/pasos[3]};
78 // VELOCIDAD MAX Y MIN DEL MOTOR EN RAD/S.. W=2*PI*f
79 float omega_motor_max[]={2*PI*f_motor_max[0],2*PI*f_motor_max[1],2*PI*f_motor_max[2],2*PI*
       f_motor_max[3]};
80 float omega_motor_min[]={2*PI*f_motor_min[0],2*PI*f_motor_min[1],2*PI*f_motor_min[2],2*PI*
       f_motor_min[3]};
81 // NUMERO DE PULSOS DE CADA INTERRUPCION DEL TIMER 1
82 uint16_t npuls_M1=65535, npuls_M2=65535;
83 // DIRECCIÓN DE CADA MOTOR
84 int dir_M1=0, dir_M2=0;
85 // TRADUCE DE º/S A PULSOS/S
86 float omega2npuls[]={omega_motor_max[0]/(T_PULS_1*f_motor_max[0]*pasos[0]),omega_motor_max[1]/(
       T_PULS_1*f_motor_max[1]*pasos[1]),omega_motor_max[2]/(T_PULS_1*f_motor_max[2]*pasos[2]),
       omega_motor_max[3]/(T_PULS_1*f_motor_max[3]*pasos[3])};
87
88 // FUNCIÓN SETUP PARA INICIAR VALORES INICIALES//
89 void setup() {
```

```
90 // COMUNICACIÓN I2C CON RPi
91 Wire.begin(DIRECCION_RPI); //ESTABLECE COMUNICACIÓN I2C CON LA RPI EN LA DIRECCIÓN
       DIRECCION_RPI QUE SERÁ LA 0X05
92 Wire.onReceive( dato_recibido ); // FUNCIÓN QUE ACTÚA COMO UN MANEJADOR SALTANDO
       CUANDO RECIBE UN ENVÍO DE DATO DE LA RPI
93 Wire.onRequest(dato_enviado); // FUNCIÓN QUE ACTÚA COMO UN MANEJADOR SALTANDO
       CUANDO RECIBE UNA PETICIÓN DE DATO DE LA RPI
94
95 // CONFIGURACIÓN DE LA VELOCIDAD DE TRANSMISIÓN I2C
96 TWBR=158;
97 TWBR|=bit(TWPS0);
98
99 // CONFIGURACIÓN DEL MONITOR SERIE Y VELOCIDAD BUETOOTH // 1=1200 , 2=2400, 3=4800,
       4=9600, 5=19200, 6=38400, 7=57600, 8=115200
100 mySerial.begin(38400);
101 Serial.begin(38400);
102
103 // CONFIGURACIÓN MOTOR 1
104 pinMode(MS1_MOTOR_1 + 8 , OUTPUT); //MS1
105 pinMode(MS2_MOTOR_1 + 8, OUTPUT); //MS2
106 pinMode(STEP_MOTOR_1 , OUTPUT); //STEP
107 pinMode(DIR_MOTOR_1, OUTPUT); //DIR
108
109 // CONFIGURACIÓN DEL MOTOR 2
110 pinMode(MS1_MOTOR_2 + 8 , OUTPUT); //MS1
111 pinMode(MS2_MOTOR_2 + 8 , OUTPUT); //MS2
112 pinMode(STEP_MOTOR_2, OUTPUT); //STEP
113 pinMode(DIR_MOTOR_2, OUTPUT); //DIR
114
115 // CONFIGURACIÓN DEL PIN
116 pinMode(PIN, OUTPUT);
117  digitalWrite (PIN, HIGH);
118
119 // CONFIGURACIÓN DE LAS INTERRUPCIONES DEL TIMER 1 PARA EL CONTROL DE LA
       VELOCIDAD
120 TCCR1A=0; //INICIAMOS EN MODO NORMAL
121 TCCR1B=0; //INICIAMOS EN MODO NORMAL
122 switch (PREESCALA_1){
123 case 1: // 4.095ms
124 SET(TCCR1B,CS10);
125 break;
126 case 8: // 32.7675ms
127 SET(TCCR1B,CS11);
128 break;
129 case 64: // 0.2621s
130 SETT(TCCR1B,CS11,CS10);
131 break;
132 case 256: // 1.048s
133 SETT(TCCR1B,WGM12,CS12);
134 break;
135 case 1024: // 4.1942
136 SETT(TCCR1B,CS12,CS10);
137 break;
138 dir_M1=0;
139 dir_M2=0;
140 };
141 OCR1A =65535; //ACTIVA INTERRUPCIÓN DEL TIMER 1A
142 OCR1B =65535; //ACTIVA INTERRUPCION DEL TIMER 1B
143 TIMSK1 =0; // LIMPIA LA MASCARA DEL TIMER
144 SETT(TIMSK1,OCIE1A,OCIE1B); //ACTIVA LAS INTERRUPCIONES A Y B DEL TIMER 1
145
```

```
146 // CONFIGURACIÓN DE LAS INTERRUPCIONES DEL TIMER 2 PARA IMPLEMENTAR LA ACELERACI
        ÓN
147 TCCR2A=0;    // INICIAMOS EN MODO NORMAL
148 TCCR2B=0;    // INICIAMOS EN MODO NORMAL
149 TCNT2=0;
150 switch (PREESCALA_2){
151 case 1:                    // 19.94us
152 SET(TCCR2B,CS20);
153 break;
154 case 8:                    // 127us
155 SET(TCCR2B,CS21);
156 break;
157 case 32:                   // 510us
158 SETT(TCCR2B,CS21,CS20);
159 break;
160 case 64:                   //  1.02ms
161 SET(TCCR2B,CS22);
162 break;
163 case 128:                  // 2.04ms
164 SETT(TCCR2B,CS20,CS22);
165 break;
166 case 256:                  // 4.08ms
167 SETT(TCCR2B,CS21,CS22);
168 break;
169 case 1024:                 // 16.32ms
170 SETTT(TCCR2B,CS20,CS21,CS22);
171 break;
172 };
173 OCR2A=250;   // ACTIVO LA INTERRUPCIÓN DEL TIMER PARA QUE SALTE CADA 4 MS
174 SET(TCCR2A,WGM21);
175 SET(TIMSK2,OCIE2A); //ACTIVO LA INTERURPCIÓN A DEL TIMER 2
176
177 }
178 // BUCLE LOOP QUE RECIBE EL DATO DEL BLUETOOTH//
179 void loop() {
180 blue();
181 delay(1);
182 }
183 // INTERRUPCION A DEL TIMER 1 PARA MOVER EL MOTOR 1//
184
185 ISR(TIMER1_COMPA_vect)
186 {
187 // ACTIVA LA INTERRUPCIÓN A DEL TIMER 1 PARA DAR VELOCIDAD AL MOTOR 1
188 if (dir_M1==0) // SI EL MOTOR NO SE ENCUENTRA GIRANDO EN NINGÚN SENTICO (CONDICIÓN
        INICIAL), EL MOTOR NO SE MOVERÁ
189 return;
190 SET(PORTD,STEP_MOTOR_1); //ACTIVA EL PIN STEEP
191 OCR1A+=npuls_M1;            // MANTIENE EL PIN EN ALTO DURANTE EL TIEMPO ESTABLECIDO
        POR NPULS_M1
192 CLR(PORTD,STEP_MOTOR_1); //DESACTIVA EL PIN STEEP
193 }
194 // INTERRUPCION B DEL TIMER 1 PARA MOVER EL MOTOR 2//
195 ISR(TIMER1_COMPB_vect)
196 {
197 // FUNCIÓN ANÁLOGA A LA ANTERIOR
198 if (dir_M2==0)
199 return;
200 SET(PORTD,STEP_MOTOR_2);
201 OCR1B+=npuls_M2;
202 CLR(PORTD,STEP_MOTOR_2);
203 }
204 // INTERRUPCION A DEL TIMER 2 PARA IMPLEMENTAR LA ACELERACIÓN//
```

```
205 //IMPLEMENTA LA VELOCIDAD ACELERADA CADA 4 MS. IMPLEMENTA UNA NUEVA ACELERACI
        ÓN CADA XX MS QUE RECIBE DATO DE LA NUEVA ACELERAICÓN DE LA RPI
206 ISR(TIMER2_COMPA_vect)
207 {
208
209  interrupts ();     //FUNCIÓN QUE EVITA QUE EL PROGRAMA SE COLAPSE
210 if (flag==0)            //BANDERA QUE PONO A CERO CUANDO SE RECIBE UNA NUEVA ACELERACIÓ
        N DESDE LA RPI
211 {
212 flag=1;                 //BANDERA A UNO PARA TRABAJR CON LA ACELERACIÓN RECIBIDA Y NO
        VOLVER A CAMBIAR LA ACELERACIÓN HASTA QUE SE VUELVA A RECIBIR VALOR POR I2C
213 comprueba=isnan(alfa);  //COMPRUEBA SI EL NUMERO QUE SE RECIBE ES REAL, EN CASO
        CONTRARIO DEVUELVE UN 1
214 if  (alfa>100 || alfa<-100 || comprueba==1) //FILTRO PARA OBVIAR LOS VALORES DE ACELERACIÓ
        N +- 100 RAD/S^2, EVITANDO QUE SE PRODUZCA OVF AL RECIBIR DATO FUERA DE RANGO
215 {
216 alfa_M1=alfa_M1;
217 }
218 else
219 {
220 alfa_M1=alfa;     //EN CASO DE RECIBIR DATO DENTRO DEL RANGO, ACTUALIZO LA ACELERACIÓ
        N QUE APLICO CON EL VALOR RECIBIDO
221 }
222 }
223
224
225 // Serial.println(alfa_M1);     //IMPRIMO LA ACELERACIÓN APLICADA PARA VER QUE LOS
        VALORES SON CORRECTOS
226 vel_M1=vel_M1+alfa_M1*T_PULS_2+giro_M1*0.05; //ACTUALIZO LA VELOCIDAD DEL MOTOR 1
        CADA T_PULS_2 MS
227 vel_M2=vel_M2-alfa_M1*T_PULS_2+giro_M1*0.05; //ACTUALIZO LA VELOCIDAD 0DEL MOTOR 2
        CADA T_PULS_2 MS
228 vel_M1=constrain(vel_M1,-60,60);    //SATURO LA VELOCIDAD ENTRE LOS LÍMITES ALCANZABLES
229 vel_M2=constrain(vel_M2,-60,60);    //SATURO LA VELOCIDAD ENTRE LOS LÍMITES ALCANZABLES
230 comprueba_vel_1(vel_M1); //COMPRUEBO LA DIRECCIÓN DE GIRO DEL MOTOR 1 CON LA NUEVA
        VELOCIDAD
231 comprueba_vel_2(vel_M2); //COMPRUBEO LA DIRECCIÓN DE GIRO DEL MOTOR 2 CON LA NUEVA
        VELOCIDAD
232 vel_M1-=giro_M1*0.05;
233 vel_M2-=giro_M1*0.05;
234 vec[0]=(vel_M1-vel_M2)/2.0; //CÁLCULO EL VALOR DE LA VELOCIDAD MEDIA QUE LE ENVÍO A LA
        PI
235 }
236
237 //FUCIÓN QUE RECIBE DATOS POR I2C//
238 //ESTA FUNCIÓN ACTÚA COMO UN MANEJADOR, SALTANDO CUANDO LA RPI QUE ACTÚA COMO
        MAESTRO ENVIA UN DATO, TRABAJA CON NUMETROS VOLATILES
239 void  dato_recibido(int howMany) {
240  interrupts ();
241  volatile  float fnum;
242  volatile  byte * p = (byte*) &fnum;  //CONVIERTE EL VALOR DE UN FLOTANTE EN BITS, YA QUE EL
         ENVÍO POR I2C SE PRODUCE EN BITS
243 while(Wire.available())    //BUCLE PARA LEER MIENTRAS QUEDEN DATOS DISPONIBLES
244 {
245 for  (int i = 0; i < 4; i++)      //BUCLE FOR PARA LEER DATOS CON UN TAMANO DE 4 BITS, UN
        FLOTANTE OCUPARÍA 8, CON CUATRO LOS RESULTADOS SON SUFICIENTEMENTE
        ACEPTABLES Y REDUCE EL COSTE COMPUTANCIONAL
246 *p++=Wire.read();           //GUARDA EL VALOR LEIDO EN UN PUNTERO
247 }
248 alfa=(float)fnum;       //GUARDA EN ALFA EL VALOR RECIBIDO COMO FLOTANTE
249 j=0;           //PONE A 0 EL VALOR DEL VECTOR DE VELOCIDAD PARA QUE CUANDO LA RPI PIDA
        UN VALOR SE LE ENVÍE PRIMERO LA ACELERACIÓN DEL MOTOR 1
```

```
250 flag=0;       // PONE LA BANDERA A CERO PARA QUE SE PROCESE EL DATO RECIBIDO EN LAS
        INTERRUPCIONES DEL TIMER 2
251 }
252 //  FUNCIÓN QUE SE ENCARGA DE ENVIAR UN DATO FLOTANTE EN FORMATO BYTE. EL DATO
        ENVIADO SERÁ LA COMPONENETE DE LA
253 // VELOCIDAD DE CADA MOTOR, PRIMERO SE ENVÍA LA COMPONENTE 0 QUE EQUIVALE A LA
        VELOCIDAD DEL MOTOR 1 Y SE IMPLEMENTA
254 //  EL VALOR DE J PARA QUE EN LA SIGUIENTE PETICIÓN DE DATO, SE ENVÍE LA COMPONENTE 1
        QUE EQUIVALE A LA VELOCIDAD
255 // DEL MOTOR2. ESTA FUNCIÓN VA SINCRONIZADA CON EL EJECUTIVO CICLICO APLICADO EN LA
         RPI.
256 void  dato_enviado()  {
257 Wire.write((byte*)  &vec[j],   sizeof(float));
258 j++;
259 }
260 //  ESTA FUNCIÓN SE ENCARGA DE COMPROBAR EL VALOR DE LA VELOCIDAD DEL MOTOR 1 Y
        GIRAR EN EL SENTIDO CORRECTO CON EL PASO QUE HACE EL MOVIMIENTO MÁS SUAVE.
        FUNCIONES DESARROLLADAS POR J.A BORJA CONDE.
261 void  comprueba_vel_1(float  vel)
262 {
263 if  (vel  >0)
264 {
265 if(dir_M1 !=1)
266 {
267 SET(PORTD,DIR_MOTOR_1);
268 dir_M1=1;
269 }
270 if  (vel<=omega_motor_max[3] && vel>=omega_motor_min[3])
271 {
272 SETT(PORTB, MS1_MOTOR_1, MS2_MOTOR_1);
273 npuls_M1=omega2npuls[3]/vel;
274 }
275 else  if  (vel<=omega_motor_max[2]&& vel>=omega_motor_min[2])
276 {
277 CLR(PORTB,MS1_MOTOR_1);
278 SET(PORTB,MS2_MOTOR_1);
279 npuls_M1=omega2npuls[2]/vel;
280 }
281 else  if  (vel<=omega_motor_max[1]&& vel>=omega_motor_min[1])
282 {
283 CLR(PORTB,MS2_MOTOR_1);
284 SET(PORTB,MS1_MOTOR_1);
285 npuls_M1=omega2npuls[1]/vel;
286 }
287 else  if  (vel<=omega_motor_max[0]&& vel>=omega_motor_min[0])
288 {
289 CLRR(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
290 npuls_M1=omega2npuls[0]/vel;
291 }
292 else
293 {
294 SETT(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
295 npuls_M1=npuls_max[3];
296 }
297 }
298 if  (vel  <0)
299 {
300 if  (dir_M1 !=-1)
301 {
302 CLR(PORTD,DIR_MOTOR_1);
303 dir_M1=-1;
304 }
```

```
305 if (-vel<=omega_motor_max[3] && -vel>=omega_motor_min[3])
306 {
307 SETT(PORTB, MS1_MOTOR_1, MS2_MOTOR_1);
308 npuls_M1=-omega2npuls[3]/vel;
309 }
310 else if (-vel<=omega_motor_max[2]&& -vel>=omega_motor_min[2])
311 {
312 CLR(PORTB,MS1_MOTOR_1);
313 SET(PORTB,MS2_MOTOR_1);
314 npuls_M1=-omega2npuls[2]/vel;
315 }
316 else if (-vel<=omega_motor_max[1]&& -vel>=omega_motor_min[1])
317 {
318 CLR(PORTB,MS2_MOTOR_1);
319 SET(PORTB,MS1_MOTOR_1);
320 npuls_M1=-omega2npuls[1]/vel;
321 }
322 else if (-vel<=omega_motor_max[0]&& -vel>=omega_motor_min[0])
323 {
324 CLRR(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
325 npuls_M1=-omega2npuls[0]/vel;
326 }
327 else
328 {
329 SETT(PORTB,MS1_MOTOR_1,MS2_MOTOR_1);
330 npuls_M1=npuls_max[3];
331 }
332 }
333 }
334
335 //FUNCIÓNA ANALOGA A LA ANTERIOR PARA EL MOTOR 2
336 void comprueba_vel_2(float vel)
337 {
338
339 if (vel >0)
340 {
341 if(dir_M2 !=1)
342 {
343 SET(PORTD,DIR_MOTOR_2);
344 dir_M2=1;
345 }
346 if (vel<=omega_motor_max[3] && vel>=omega_motor_min[3])
347 {
348 SETT(PORTB, MS1_MOTOR_2, MS2_MOTOR_2);
349 npuls_M2=omega2npuls[3]/vel;
350 }
351 else if (vel<=omega_motor_max[2]&& vel>=omega_motor_min[2])
352 {
353 CLR(PORTB,MS1_MOTOR_2);
354 SET(PORTB,MS2_MOTOR_2);
355 npuls_M2=omega2npuls[2]/vel;
356 }
357 else if (vel<=omega_motor_max[1]&& vel>=omega_motor_min[1])
358 {
359 CLR(PORTB,MS2_MOTOR_2);
360 SET(PORTB,MS1_MOTOR_2);
361 npuls_M2=omega2npuls[1]/vel;
362 }
363 else if (vel<=omega_motor_max[0]&& vel>=omega_motor_min[0])
364 {
365 CLRR(PORTB,MS1_MOTOR_2,MS2_MOTOR_2);
366 npuls_M2=omega2npuls[0]/vel;
```

```
367 }
368 else
369 {
370 SETT(PORTB,MS1_MOTOR_2,MS2_MOTOR_2);
371 npuls_M2=npuls_max[3];
372 }
373 }
374 if (vel <0)
375 {
376 if (dir_M2 !=-1)
377 {
378 CLR(PORTD,DIR_MOTOR_2);
379 dir_M2=-1;
380 }
381 if (-vel<=omega_motor_max[3] && -vel>=omega_motor_min[3])
382 {
383 SETT(PORTB, MS1_MOTOR_2, MS2_MOTOR_2);
384 npuls_M2=-omega2npuls[3]/vel;
385 }
386 else if (-vel<=omega_motor_max[2]&& -vel>=omega_motor_min[2])
387 {
388 CLR(PORTB,MS1_MOTOR_2);
389 SET(PORTB,MS2_MOTOR_2);
390 npuls_M2=-omega2npuls[2]/vel;
391 }
392 else if (-vel<=omega_motor_max[1]&& -vel>=omega_motor_min[1])
393 {
394 CLR(PORTB,MS2_MOTOR_2);
395 SET(PORTB,MS1_MOTOR_2);
396 npuls_M2=-omega2npuls[1]/vel;
397 }
398 else if (-vel<=omega_motor_max[0]&& -vel>=omega_motor_min[0])
399 {
400 CLRR(PORTB,MS1_MOTOR_2,MS2_MOTOR_2);
401 npuls_M2=-omega2npuls[0]/vel;
402 }
403 else
404 {
405 SETT(PORTB,MS1_MOTOR_2,MS2_MOTOR_2);
406 npuls_M2=npuls_max[3];
407 }
408 }
409
410
411 }
412
413 void blue()
414 {
415 if(mySerial.available()) {
416
417 cmd[0] = mySerial.read();
418 if(cmd[0] == STX) {
419 int i=1;
420 while(mySerial.available()) {
421 delay(1);
422 cmd[i] = mySerial.read();
423 if(cmd[i]>127 || i>7) break;
424 if((cmd[i]==ETX) && (i==2 || i==7)) break;
425 i++;
426 }
427
428 if(i==7) getJoystickState(cmd);
```

```
429 }
430 }
431 }
432 int   jjj ;
433
434 void   getJoystickState (byte data[8])      {
435 int joyX = (data[1]-48)*100 + (data[2]-48)*10 + (data[3]-48);
436 int joyY = (data[4]-48)*100 + (data[5]-48)*10 + (data[6]-48);
437 joyX = joyX - 200;
438 joyY = joyY - 200;
439 if (joyX<-100 || joyX>100 || joyY<-100 || joyY>100)       return ;
440
441 giro_M1=joyX;
442 vec[1]=joyY*0.35;
443 }
```

2 Biblioteca MPU6050.h

```
1
2  //selecciona el rango del giroscopio, por defecto será 0
3  #define GYRO_RANGE 0
4  // Gyroscope Range
5  // 0 +/- 250 degrees/second
6  // 1 +/- 500 degrees/second
7  // 2 +/- 1000 degrees/second
8  // 3 +/- 2000 degrees/second
9
10 //selecciona el rango del acelerómetro, por defecto será 0
11 #define ACCEL_RANGE 0
12 // 0 +/- 2g
13 // 1 +/- 4g
14 // 2 +/- 8g
15 // 3 +/- 16g
16
17 //ofset calculado con su funcion correspondiente
18 //   Accelerometer
19 #define A_OFF_X 8977.75
20 #define A_OFF_Y 1870.60
21 #define A_OFF_Z -16697.05
22 //  Gyroscope
23 #define G_OFF_X -3.55
24 #define G_OFF_Y 0.65
25 #define G_OFF_Z 3.7
26
27 //-----------------------FIN MODIFICACION PARAMETROS
      ------------------------
28 // cabeceras necesarias
29 #include <stdio.h>
30 #include <unistd.h>
31 #include <fcntl.h>
32 #include <sys/ioctl.h>
33 #include <sys/types.h>
34 #include <time.h>
35 #include <sys/time.h>
36 #include <linux/i2c-dev.h>
```

```
37 #include <sys/fcntl.h>
38 #include <sys/stat.h>
39 #include <i2c/smbus.h>
40 #include <math.h>
41 #include <pthread.h>
42
43 //normas posix para calcular el tiempo
44 #define _POSIX_C_SOURCE 200809L
45
46 //cambio de radianes a grados
47 #define RAD_T_DEG 57.29577951308
48
49 //posibilidades de rango giroscopio, por defecto usaremos 0
50 #if GYRO_RANGE == 1
51 #define GYRO_SENS 65.5
52 #define GYRO_CONFIG 0b00001000
53 #elif GYRO_RANGE == 2
54 #define GYRO_SENS 32.8
55 #define GYRO_CONFIG 0b00010000
56 #elif GYRO_RANGE == 3
57 #define GYRO_SENS 16.4
58 #define GYRO_CONFIG 0b00011000
59 #else
60 #define GYRO_SENS 131.0
61 #define GYRO_CONFIG 0b00000000
62 #endif
63 #undef GYRO_RANGE
64
65 //posibilidades de rango giroscopio, por defecto usaremos 0
66 #if GYRO_RANGE == 1
67 #define ACCEL_SENS 8192.0
68 #define ACCEL_CONFIG 0b00001000
69 #elif GYRO_RANGE == 2
70 #define ACCEL_SENS 4096.0
71 #define ACCEL_CONFIG 0b00010000
72 #elif GYRO_RANGE == 3
73 #define ACCEL_SENS 2048.0
74 #define ACCEL_CONFIG 0b00011000
75 #else
76 #define ACCEL_SENS 16384.0
77 #define ACCEL_CONFIG 0b00000000
78 #endif
79 #undef GYRO_RANGE
80
81 //variables globales
82 float _accel_angle[3];
83 float _gyro_angle[3];
84 int MPU6050_addr;
85 int f_dev;
86
87
88
89 //función para iniciar el sensor, recibe como argumento la dirección
      i2c
90 void inicio_mpu (int addr)
91 {
```

```
92  int status;
93  MPU6050_addr = addr; //guarda la dirección recibida en la variable
       MPU6050_addr
94  //abre la comuicacion i2c
95  f_dev = open("/dev/i2c-1", O_RDWR);
96  if (f_dev < 0) {
97  printf("error abriendo la conexión");
98  }
99  status = ioctl(f_dev, I2C_SLAVE, MPU6050_addr);
100 if (status < 0) {
101 printf("error");
102
103
104 }
105 i2c_smbus_write_byte_data(f_dev, 0x6b, 0b00000000); //desabilita el
       modo sleep del dispositivo
106
107 i2c_smbus_write_byte_data(f_dev, 0x1a, 0b00000011); //ajusta un
       filtro paso bajo a 44Hz
108
109 i2c_smbus_write_byte_data(f_dev, 0x19, 0b00000100); //Establece el
       divisor de frecuencia de muestreo
110
111 i2c_smbus_write_byte_data(f_dev, 0x1b, GYRO_CONFIG); //configura los
       agustes del giroscopio
112
113 i2c_smbus_write_byte_data(f_dev, 0x1c, ACCEL_CONFIG); //configura los
        ajustes del acelerómetro
114
115 //pone el offset a ceto
116 i2c_smbus_write_byte_data(f_dev, 0x06, 0b00000000),
       i2c_smbus_write_byte_data(f_dev, 0x07, 0b00000000),
       i2c_smbus_write_byte_data(f_dev, 0x08, 0b00000000),
       i2c_smbus_write_byte_data(f_dev, 0x09, 0b00000000),
       i2c_smbus_write_byte_data(f_dev, 0x0A, 0b00000000),
       i2c_smbus_write_byte_data(f_dev, 0x0B, 0b00000000),
       i2c_smbus_write_byte_data(f_dev, 0x00, 0b10000001),
       i2c_smbus_write_byte_data(f_dev, 0x01, 0b00000001),
       i2c_smbus_write_byte_data(f_dev, 0x02, 0b10000001);
117
118 }
119
120 //obtiene los valores en bruto del acelerómetro
121 void getAccelRaw(float *x, float *y, float *z)
122 {
123 int16_t X = i2c_smbus_read_byte_data(f_dev, 0x3b) << 8 |
       i2c_smbus_read_byte_data(f_dev, 0x3c);
124 int16_t Y = i2c_smbus_read_byte_data(f_dev, 0x3d) << 8 |
       i2c_smbus_read_byte_data(f_dev, 0x3e);
125 int16_t Z = i2c_smbus_read_byte_data(f_dev, 0x3f) << 8 |
       i2c_smbus_read_byte_data(f_dev, 0x40);
126 *x = (float)X;
127 *y = (float)Y;
128 *z = (float)Z;
129 }
130
```

```
131 //obtiene los valores en bruto del giroscopio
132 void getGyroRaw(float *roll, float *pitch, float *yaw)
133 {
134 int16_t X = i2c_smbus_read_byte_data(f_dev, 0x43) << 8 |
       i2c_smbus_read_byte_data(f_dev, 0x44);
135 int16_t Y = i2c_smbus_read_byte_data(f_dev, 0x45) << 8 |
       i2c_smbus_read_byte_data(f_dev, 0x46);
136 int16_t Z = i2c_smbus_read_byte_data(f_dev, 0x47) << 8 |
       i2c_smbus_read_byte_data(f_dev, 0x48);
137 *roll = (float)X; //giro en el eje x
138 *pitch = (float)Y; //giro en el eje y
139 *yaw = (float)Z; //giro en el eje z
140
141 }
142
143 //obtiene los valores del acelerómetro en g con 3 cifras decimales
144 void getAccel(float *x, float *y, float *z)
145 {
146 getAccelRaw(x, y, z);
147 *x =round((*x - A_OFF_X)*1000.0 / ACCEL_SENS)/1000.0 ;
148 *y =round((*y - A_OFF_Y)*1000.0 /ACCEL_SENS)/1000.0;
149 *z = round((*z - A_OFF_Z)*1000.0 / ACCEL_SENS)/1000.0 ;
150 }
151
152
153 //obtiene los valores del giroscopio en g/s con 3 cifras decimales
154 void getGyro(float *roll, float *pitch, float *yaw)
155 {
156 getGyroRaw(roll, pitch, yaw);
157 *roll = round((*roll - G_OFF_X) * 1000.0 / GYRO_SENS) / 1000.0;
158 *pitch = round((*pitch - G_OFF_Y) * 1000.0 / GYRO_SENS) / 1000.0;
159 *yaw = round((*yaw - G_OFF_Z) * 1000.0 / GYRO_SENS) / 1000.0;
160 }
161
162 //calcula el offset
163 void getOffsets(float *ax_off, float *ay_off, float *az_off, float *
       gr_off, float *gp_off, float *gy_off)
164 {
165 float gyro_off[3];
166 float accel_off[3];
167 float suma_x=0, suma_y=0, suma_z=0, suma_gx=0, suma_gy=0, suma_gz=0;
168 //bucle para calcular los ofset como sumatorio y luego hacer el valor
        medio
169 for (int i = 0; i < 10000; i++) {
170 getGyroRaw(&gyro_off[0], &gyro_off[1], &gyro_off[2]); //
171 suma_gx=suma_gx+ gyro_off[0], suma_gy=suma_gy+ gyro_off[1], suma_gz=
       suma_gz+ gyro_off[2];
172 getAccelRaw(&accel_off[0], &accel_off[1], &accel_off[2]);
173 suma_x=suma_x+accel_off[0], suma_y=suma_y+accel_off[1], suma_z=suma_z
       + accel_off[2];
174 }
175
176 *gr_off = suma_gx / 10000, *gp_off = suma_gy / 10000, *gy_off =
       suma_gz / 10000;
177 *ax_off = suma_x / 10000, *ay_off = suma_y / 10000, *az_off = suma_z
       / 10000;
```

```
178 *az_off = *az_off - ACCEL_SENS ;
179 }
```

3 Biblioteca ARDUINO.h

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <fcntl.h>
4  #include <sys/ioctl.h>
5  #include <sys/types.h>
6  #include <time.h>
7  #include <sys/time.h>
8  #include <linux/i2c-dev.h>
9  #include <sys/fcntl.h>
10 #include <sys/stat.h>
11 #include <i2c/smbus.h>
12 #include <math.h>
13 #include <pthread.h>
14
15 //VARIABLES PARA LA COMUNICACIÓN CON ARDUINO
16 int ADRESS_ARDUINO;
17 int file;
18
19 void inicio_arduino(int ddr)
20 {
21 ADRESS_ARDUINO=ddr;
22 file = open("/dev/i2c-1", O_RDWR);
23 if (file < 0) {
24 printf("error abriendo la conexión");
25 }
26 if (ioctl (file, I2C_SLAVE, ADRESS_ARDUINO) < 0) {
27 printf("I2C: error al acceder al esclavo en 0x%x", ADRESS_ARDUINO);
28 }
29 }
```

4 Código LQR y LQRI

```
1
2  //DECLARACIÓN DE CABECERAS
3  #include "MPU6050.h" //dentro de dicha libreria va declarada todas
       las cabeceras necesarias
4  #include <signal.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include "ARDUINO.h"
8
9
10 //Para implementar el control LQRI se varía la cuarta componente del
       vector K a 0.3
11 float K[]={-350.22,-47.23,7.42,0};
12 float offset=3.5;
```

```
13
14 //FUNCIONES
15 float calculo_lqr(float A, float B, float gir);
16 float filtro(float accel, float gxy);
17
18 //CAMBIO DE RADIANES A GRADOS
19 #define DEG_TO_RAD 0.01745329252
20 float PI=3.1415;
21 int cont;
22
23 //VARIABLES PARA EL MPU
24 float ax,ay,az, ax_escalado, ay_escalado, az_escalado;
25 float accel_ang_y;
26 float gx,gy,gz, gx_escalado, gy_escalado, gz_escalado;
27 float giros_ang_y, giros_ang_y_prev=3;
28 float TAU=0.99;
29
30 //VARIABLES PARA COMUNICAR CON ARDUINO
31 float omega, omega_r;
32 float alpha;
33
34 //VARIABLES PARA CONTROL LQR
35
36 float ivr=0, u;
37 float dtheta_r, phi,dphi,dtheta;
38
39 //VARIABLES PARA TEMPORIZADOR Y SENAL
40 timer_t temporizador;//declaración del nombre del temporizador
41 struct itimerspec periodo;//estructura para indicar cada cuanto
      tiempo se lanza el temporizador
42 struct timespec inicial;
43 struct timespec tiempo; //estructura para definirla duración del
      temporizador
44 struct sigevent aviso; //establezco una senal que avise que se ha
      cumplido el temporizador
45 sigset_t sigset; //conjunto de senales
46 struct sigaction act;
47
48 //VARIABLE EJECUTIVO CICLICO
49 int ciclo=0;
50 double dtt;
51 clock_t iniciall;
52 int flag=0;
53
54 //FUNCIÓN MANEJADOR
55 void Manejador(int signo, siginfo_t *info, void*context)
56 {
57 ciclo++;
58 //CADA VEZ QUE SE ACTIVA LA SENAL, CICLO AUMENTA EN UNA UNIDAD Y HACE
       LO QUE LE CORRESPONDA EN EL SWITCH-CASE
59 //printf("ciclo %d \n", ciclo);
60 switch(ciclo)
61 {
62 case 1 :
63 getGyro(&gx_escalado, &gy_escalado, &gz_escalado);
```

```
64  getAccel(&ax_escalado, &ay_escalado, &az_escalado); //lee valores en
       el sistema internacional g
65  accel_ang_y = atan2(az_escalado , (sqrt((ay_escalado*ay_escalado) + (
       ax_escalado*ax_escalado))))*(180/PI); //da el angulo en grados
66  giros_ang_y =filtro(accel_ang_y,gy_escalado);
67
68  //printf("gx %f \n", gx_escalado);
69  //printf("gy %f \n", gy_escalado);
70  //printf("gz %f \n", gz_escalado);
71  //printf("ay %f \n", ay_escalado);
72  //printf("az %f \n", accel_ang_y);
73  //printf("dphi %f \n", gy_escalado);
74  //printf ("angulo %f \n", (giros_ang_y+offset));
75  break;
76
77  case 2:
78  read(file, &omega, sizeof(omega));
79  //printf("omega %f \n", omega_A);
80  break;
81
82  case 3:
83  read(file, &omega_r, sizeof(omega_r));
84  //printf ("ref %f \n", omega_r);
85  break;
86
87  case 4:
88  alpha=calculo_lqr(omega,omega_r, giros_ang_y);
89  //printf ("alfa %f \n", alpha);
90  break;
91
92  case 5:
93  write(file, &alpha, sizeof(alpha));
94  ciclo = 0;
95  break;
96
97  }
98  }
99
100
101 //FUNCIÓN PRINCIPAL MAIN
102 int main(void )
103 {
104 inicio_mpu(0x68); //inicio la conexión del MPU6050
105 inicio_arduino(0x05);//inicio la conexión con ARDUINO
106 if (flag==0){
107 flag=1;
108 iniciall=clock();
109 }
110 //TRATAMIENTO DE SENALES
111 sigemptyset(&sigset); //crea una mascara vacía
112 sigaddset(&sigset, SIGUSR1); //anade la senal SIGALARM al conjunto
113 pthread_sigmask(SIG_UNBLOCK,&sigset, NULL); //bloqueo la senal para
       el proceso
114 /* programo la senar SIGVTALRM para que salte la funcion manejador
       cuando se cumpla el temporizador */
115 act.sa_sigaction=Manejador;
```

```
116 sigemptyset(&(act.sa_mask));
117 act.sa_flags=SA_SIGINFO;
118 sigaction(SIGUSR1, &act, NULL);
119 //configuro el temporizador
120 inicial.tv_sec=1;
121 tiempo.tv_nsec=4000000; //cada 4 ms salta el manejador
122 periodo.it_value=inicial; //indico que el primer valor lo lance 1
       segundo despues de llamarlo
123 periodo.it_interval=tiempo; //indico que los peridos sean cada 4 ms
124 aviso.sigev_notify=SIGEV_SIGNAL;
125 aviso.sigev_signo=SIGUSR1;
126 timer_create(CLOCK_REALTIME, &aviso, &temporizador);//creo el
       temporizador llamado tempo, de tiempo real y que avise con la
       senal configurada en el segundo argumento de entrada
127 //inicio el temporizador
128 timer_settime(&temporizador, TIMER_ABSTIME, &periodo , NULL);
129 while(1){}
130 exit(0);
131 }
132
133 //función que calcula la acción de control
134 float calculo_lqr(float A, float B,float gir)
135 {
136 phi=(gir+offset)*DEG_TO_RAD;
137 dphi=gy_escalado*DEG_TO_RAD;
138 dtheta=A;
139 ivr=ivr+B-A;
140 u=K[0]*(phi)+K[1]*(dphi)+K[2]*(B+A)+K[3]*ivr;
141 return (u);
142 }
143
144 //función que calcula el filtro complementario
145 float filtro(float accel, float gxy)
146 {
147 dtt=(double)(clock()-iniciall)/CLOCKS_PER_SEC;
148 giros_ang_y_prev=TAU*(giros_ang_y_prev+dtt*gxy)+(1-TAU)*accel;
149 iniciall=clock();
150 return (giros_ang_y_prev);
151 }
```

5 Código MPC

```
1  //DECLARACIÓN DE CABECERAS
2  #include "MPU6050.h" //dentro de dicha libreria va declarada todas
       las cabeceras necesarias
3  #include <signal.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include "ARDUINO.h"
7  #include <stdbool.h>
8
9  //FUNCIONES
10 void MPCT_EADMM(float *pointer_u, float *pointer_x0, float *
       pointer_ref);
```

```
float filtro (float accel, float gxy);

//VARIABLES PARA EL MPC
#define n 3
#define m 1
#define nm 4
#define N 12
#define k_max 500
#define tol 0.0010000000
const float beta[4][13] = { {1000.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 1000.0000000000}, {1000.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 1000.0000000000},
    {1000.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    1000.0000000000}, {5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 5.0000000000, 5.0000000000, 5.0000000000,
    5.0000000000, 200.0000000000} };
const float LB[4] = { -0.8707963268, -1000.0000000000, -3.0000000000,
     -4.50000000 };
const float UB[4] = { 0.8707963268, 1000.0000000000,
    3.0000000000,4.50000000 };
const float AB[3][4] = { {1.0131085393, 0.0200873141, 0.0000000000,
    -0.0029194134}, {1.3137103440, 1.0131085393, 0.0000000000,
    -0.2925774981}, {0.0000000000, 0.0000000000, 1.0000000000,
    0.0200000000} };
const float TS[4] = { -1000.0000000000, -1000.0000000000,
    -1000.0000000000, -0.1250000000 };
const float Alpha[11][3][3] = { {{-0.3187462091, -0.4133221424,
    0.0000000000}, {-0.0014156360, -0.2870717845, 0.0000000000},
    {-0.0000516370, -0.0030857846, -0.3145598227}}, {{-0.3186572629,
    -0.4132068048, 0.0000000000}, {0.0030727422, -0.2608568047,
    0.0000000000}, {-0.0000324881, -0.0048119945, -0.3145044137}},
    {{-0.3185796468, -0.4131061590, 0.0000000000}, {0.0062006954,
    -0.2440971015, 0.0000000000}, {0.0000645421, -0.0053381803,
    -0.3144719197}}, {{-0.3185474408, -0.4130643971, 0.0000000000},
    {0.0079683437, -0.2351160162, 0.0000000000}, {0.0002045042,
    -0.0051923599, -0.3144451799}}, {{-0.3185546280, -0.4130737169,
    0.0000000000}, {0.0088170295, -0.2308171255, 0.0000000000},
    {0.0003564351, -0.0047721595, -0.3144147698}}, {{-0.3185844102,
    -0.4131123358, 0.0000000000}, {0.0091495440, -0.2289528124,
    0.0000000000}, {0.0005037833, -0.0042795787, -0.3143777503}},
    {{-0.3186239173, -0.4131635652, 0.0000000000}, {0.0092173022,
    -0.2282754245, 0.0000000000}, {0.0006398447, -0.0037983638,
    -0.3143341756}}, {{-0.3186658007, -0.4132178759, 0.0000000000},
    {0.0091572370, -0.2281587293, 0.0000000000}, {0.0007627965,
    -0.0033572915, -0.3142850806}}, {{-0.3187064173, -0.4132705440,
    0.0000000000}, {0.0090402573, -0.2282983200, 0.0000000000},
    {0.0008728902, -0.0029626145, -0.3142316592}}, {{-0.3187441762,
    -0.4133195064, 0.0000000000}, {0.0089023348, -0.2285468779,
    0.0000000000}, {0.0009711361, -0.0026122911, -0.3141749933}},
```

```
    {{-0.3187785344, -0.4133640591, 0.0000000000}, {0.0087613840,
    -0.2288329869, 0.0000000000}, {0.0010587384, -0.0023017687,
    -0.3141159920}} };
26  const float Beta[12][3][3] = { {{3.1462197455, 0.0047650758,
    -0.0000365577}, {0.0000000000, 2.8897514064, -0.0033645762},
    {0.0000000000, 0.0000000000, 3.1455982272}}, {{3.1453417925,
    0.0098975925, -0.0000876370}, {0.0000000000, 2.6831294958,
    -0.0057265604}, {0.0000000000, 0.0000000000, 3.1450441366}},
    {{3.1445756740, 0.0139248122, -0.0000686687}, {0.0000000000,
    2.5544277560, -0.0068380111}, {0.0000000000, 0.0000000000,
    3.1447191968}}, {{3.1442577812, 0.0163654304, 0.0000272831},
    {0.0000000000, 2.4866613927, -0.0070711744}, {0.0000000000,
    0.0000000000, 3.1444517987}}, {{3.1443287236, 0.0175834046,
    0.0001656615}, {0.0000000000, 2.4542582365, -0.0068722004},
    {0.0000000000, 0.0000000000, 3.1441476981}}, {{3.1446226917,
    0.0180766713, 0.0003158739}, {0.0000000000, 2.4397389987,
    -0.0065156522}, {0.0000000000, 0.0000000000, 3.1437775033}},
    {{3.1450126511, 0.0181906925, 0.0004615580}, {0.0000000000,
    2.4337664715, -0.0061289247}, {0.0000000000, 0.0000000000,
    3.1433417558}}, {{3.1454260658, 0.0181204255, 0.0005960755},
    {0.0000000000, 2.4317951393, -0.0057615166}, {0.0000000000,
    0.0000000000, 3.1428508060}}, {{3.1458269770, 0.0179677681,
    0.0007176134}, {0.0000000000, 2.4316725701, -0.0054287773},
    {0.0000000000, 0.0000000000, 3.1423165923}}, {{3.1461996797,
    0.0177843589, 0.0008264128}, {0.0000000000, 2.4323788300,
    -0.0051326406}, {0.0000000000, 0.0000000000, 3.1417499335}},
    {{3.1465388162, 0.0175958003, 0.0009234686}, {0.0000000000,
    2.4334257379, -0.0048703879}, {0.0000000000, 0.0000000000,
    3.1411599197}}, {{22.4828378165, 0.1244178314, 0.0072158015},
    {0.0000000000, 4.2822303782, -0.0119272289}, {0.0000000000,
    0.0000000000, 21.2648360907}} };
27  int i = 0;
28  int j = 0;
29  int k = 0;
30  int l = 0;
31  float z1[4][13] = { {0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000}, {0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000}, {0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000},
    {0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000} };
32  float z2[4] = { 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000 };
33  float z3[4][13] = { {0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000}, {0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
```

```
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000}, {0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000},
    {0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000} };
34  float lambda[4][15] = { {0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000},
    {0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000}, {0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000}, {0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000, 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000} };
35  float q2[4];
36  float q3[4][13];
37  float res[4][15];
38  float mu3[3][12];
39  float x0[4] = { 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000 };
40  float res_1;
41  float res_2;
42  float ref[4] = { 0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000 };
43  bool e_flag;
44  const float H1i[4][13] = { {0.0005000000, 0.2000000000, 0.2000000000,
     0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.0005000000}, {0.0005000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.0005000000}, {0.0005000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.0005000000},
    {0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.2000000000, 0.2000000000, 0.2000000000, 0.2000000000,
    0.0025000000} };
45  const float W2[4][4] = { {0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000}, {0.0000000000, -0.0000000000, 0.0000000000,
    0.0000000000}, {0.0000000000, 0.0000000000, -0.0002466091,
    0.0000000000}, {-0.0000000000, 0.0000000000, 0.0000000000,
    0.0000000000} };
```

```
46 const float H3i[4][13] = { {0.0009950249, 0.1000000000, 0.1000000000,
       0.1000000000, 0.1000000000, 0.1000000000, 0.1000000000,
   0.1000000000, 0.1000000000, 0.1000000000, 0.1000000000,
   0.1000000000, 0.0009950249}, {0.0009950249, 0.1000000000,
   0.1000000000, 0.1000000000, 0.1000000000, 0.1000000000,
   0.1000000000, 0.1000000000, 0.1000000000, 0.1000000000,
   0.1000000000, 0.1000000000, 0.0009950249}, {0.0009950249,
   0.1000000000, 0.1000000000, 0.1000000000, 0.1000000000,
   0.1000000000, 0.1000000000, 0.1000000000, 0.1000000000,
   0.1000000000, 0.1000000000, 0.1000000000, 0.0009950249},
   {0.1990049751, 0.1990049751, 0.1990049751, 0.1990049751,
   0.1990049751, 0.1990049751, 0.1990049751, 0.1990049751,
   0.1990049751, 0.1990049751, 0.1990049751, 0.1990049751,
   0.0049993751} };
47 float scaling[4] = {1.0, 1.0, 0.05, 0.05};
48 float scaling_inv[4] = {1.0, 1.0, 20, 20};
49 // Other varaibles
50 bool first_cicle = true; // This variable is used to be able to do
      things only the first loop
51 float refEng[4] = {0.0, 0.0, 0.0, 0.0}; // This is the reference in
      engineering (real) units. ref = (xr, ur)
52 float x0Eng[4] = {0.0, 0.0,0.0, 0.0}; // This is the current system
      state, in engineering (real) units. This vector has the current
      state in its first n components. After that, it must have zeros.
53 float u_ctrl[1]; // Control action
54
55 //CAMBIO DE RADIANES A GRADOS
56 #define DEG_TO_RAD 0.01745329252
57 float PI=3.1415;
58
59 //VARIABLES PARA EL MPU
60 float ax,ay,az, ax_escalado, ay_escalado, az_escalado;
61 float accel_ang_y;
62 float gx,gy,gz, gx_escalado, gy_escalado, gz_escalado;
63 float giros_ang_y, giros_ang_y_prev=3;
64 float TAU=0.99;
65 float offset=3.5;
66
67 //VARIABLES PARA COMUNICAR CON ARDUINO
68 float omega, omega_r;
69 float alpha;
70 float dtheta_r, phi,dphi,dtheta;
71 clock_t iniciall;
72 double tim;
73 double dtt;
74 clock_t inicial_ang;
75 int flag=0;
76
77 //VARIABLES PARA TEMPORIZADOR Y SEnAL
78 timer_t temporizador;//declaración del nombre del temporizador
79 struct itimerspec periodo;//estructura para indicar cada cuanto
      tiempo se lanza el temporizador
80 struct timespec inicial;
81 struct timespec tiempo; //estructura para definirla duración del
      temporizador
```

```
82  struct sigevent aviso; //establezco una senal que avise que se ha
        cumplido el temporizador
83  sigset_t sigset; //conjunto de senales
84  struct sigaction act;
85
86  //VARIABLE EJECUTIVO CICLICO
87  int ciclo=0;
88
89  //FUNCIÓN MANEJADOR
90  void Manejador(int signo, siginfo_t *info, void*context)
91  {
92  write(file, &alpha_A, sizeof(alpha_A));
93  usleep(1000);
94
95  getGyro(&gx_escalado, &gy_escalado, &gz_escalado);
96  getAccel(&ax_escalado, &ay_escalado, &az_escalado); //lee valores en
        el sistema internacional g
97  accel_ang_y = atan2(az_escalado , (sqrt((ay_escalado*ay_escalado) + (
        ax_escalado*ax_escalado))))*(180/PI); //da el angulo en grados
98  giros_ang_y =filtro(accel_ang_y,gy_escalado);
99
100 read(file, &omega, sizeof(omega));
101 usleep(1000);
102
103 read(file, &omega_r, sizeof(omega_r));
104 usleep(1000);
105
106 x0Eng[0]=-(giros_ang_y+offset)*0.01745329252;
107 x0Eng[1]=-gy_escalado*0.01745329252;
108 x0Eng[2]=omega_A;
109 refEng[2]=omega_r;
110
111 iniciall=clock();
112 MPCT_EADMM(u_ctrl, x0Eng, refEng);
113 double tim=(double)(clock()-iniciall)/CLOCKS_PER_SEC;
114 alpha_A=u_ctrl[0];
115
116 }
117
118 //FUNCIÓN PRINCIPAL MAIN
119 int main(void )
120 {
121 inicio_mpu(0x68); //inicio la conexión del MPU6050
122 inicio_arduino(0x05);//inicio la conexión con ARDUINO
123 if (flag==0)
124 {
125 flag=1;
126 inicial_ang=clock();
127 }
128 //TRATAMIENTO DE SENALES
129 sigemptyset(&sigset); //crea una mascara vacía
130 sigaddset(&sigset, SIGUSR1); //anade la senal SIGALARM al conjunto
131 pthread_sigmask(SIG_UNBLOCK,&sigset, NULL); //bloqueo la senal para
        el proceso
132 /* programo la senar SIGVTALRM para que salte la funcion manejador
        cuando se cumpla el temporizador */
```

```
133 act.sa_sigaction=Manejador;
134 sigemptyset(&(act.sa_mask));
135 act.sa_flags=SA_SIGINFO;
136 sigaction(SIGUSR1, &act, NULL);
137 //configuro el temporizador
138 inicial.tv_sec=1;
139 tiempo.tv_nsec=16000000; //cada 2 ms salta el manejador
140 periodo.it_value=inicial; //indico que el primer valor lo lance 1
       segundo despues de llamarlo
141 periodo.it_interval=tiempo; //indico que los peridos sean cada 20 ms
142 aviso.sigev_notify=SIGEV_SIGNAL;
143 aviso.sigev_signo=SIGUSR1;
144 timer_create(CLOCK_REALTIME, &aviso, &temporizador);//creo el
       temporizador llamado tempo, de tiempo real y que avise con la
       senal configurada en el segundo argumento de entrada
145 //inicio el temporizador
146 timer_settime(&temporizador, TIMER_ABSTIME, &periodo , NULL);
147 while(1){}
148 exit(0);
149 }
150
151 float filtro(float accel, float gxy)
152 {
153 dtt=(double)(clock()-inicial_ang)/CLOCKS_PER_SEC;
154 giros_ang_y_prev=TAU*(giros_ang_y_prev+dtt*gxy)+(1-TAU)*accel;
155 inicial_ang=clock();
156 return (giros_ang_y_prev);
157 }
158
159 void MPCT_EADMM(float *pointer_u, float *pointer_x0, float *
       pointer_ref)
160 {
161
162 // Obtain variables in scaled units
163 for(i = 0; i < nm; i++){
164 ref[i] = pointer_ref[i]*scaling[i];
165 x0[i] = pointer_x0[i]*scaling[i];
166 }
167
168 // Initialize EADMM variables
169 bool done = false;
170 k = 0;
171
172 // Algorithm
173 while( done == false){
174
175 k += 1;
176
177 // Problem QP1
178 for(i = 0; i < nm; i++) {
179 z1[i][0] = fmax( fmin( (beta[i][0]*(x0[i] + z3[i][0] + z2[i]) +
       lambda[i][1] - lambda[i][0] )*H1i[i][0], UB[i]), LB[i]);
180
181 }
182 for(j = 1; j < N; j++) {
183 for(i = 0; i < nm; i++){
```

```
184 z1[i][j] = fmax( fmin( (beta[i][j]*(z2[i] + z3[i][j]) + lambda[i][j
        +1] )*H1i[i][j], UB[i]), LB[i]);
185 }
186 }
187 for(i = 0; i < nm; i++){
188 z1[i][N] = fmax( fmin( (beta[i][N]*(2*z2[i] + z3[i][N]) + lambda[i][N
        +1] + lambda[i][N+2] )*H1i[i][N], UB[i]), LB[i]);
189 }
190
191 // Problem QP2
192
193 // Compute q2
194 for(i = 0; i < nm; i++){
195 q2[i] = TS[i]*ref[i] + beta[i][N]*(z3[i][N] - 2*z1[i][N]);
196 }
197 for(j = 0; j < N; j++){
198 for(i = 0; i< nm; i++){
199 q2[i] = q2[i] + beta[i][j]*(z3[i][j] - z1[i][j]);
200 }
201 }
202 for(j = 1; j < N+3; j++){
203 for(i = 0; i< nm; i++){
204 q2[i] = q2[i] + lambda[i][j];
205 }
206 }
207
208 // Compute z2
209 for(i = 0; i < nm; i++){
210 z2[i] = 0;
211 for(j = 0; j < nm; j++){
212 z2[i] = z2[i] + W2[i][j]*q2[j];
213 }
214 }
215
216 // Problem QP3
217
218 // Compute q3
219 for(i = 0; i < nm; i++){
220 q3[i][0] = beta[i][0]*(z2[i] - z1[i][0]) + lambda[i][1];
221 }
222 for(j = 1; j< N; j++){
223 for(i = 0; i< nm; i++){
224 q3[i][j] = beta[i][j]*(z2[i] - z1[i][j]) + lambda[i][j+1];
225 }
226 }
227 for(i = 0; i < nm; i++){
228 q3[i][N] = beta[i][N]*(z2[i] - z1[i][N]) + lambda[i][N+1];
229 }
230
231 // Compute mu3: Forward substitution
232
233
234 // Compute first n elements
235 for(j = 0; j < n; j++){
236 // Compute r.h.s. vector
237 mu3[j][0] = H3i[j][1]*q3[j][1];
```

```
238 for(i = 0; i < nm; i++){
239 mu3[j][0] = mu3[j][0] - AB[j][i]*H3i[i][0]*q3[i][0];
240 }
241 // Forward substitution
242 for(i = 0; i <= j-1; i++){
243 mu3[j][0] = mu3[j][0] - Beta[0][i][j]*mu3[i][0];
244 }
245 mu3[j][0] = Beta[0][j][j]*mu3[j][0]; // Divide by the diagonal
        element
246 }
247 // Compute all other grout except for the last n elements
248 for(l = 1; l < N-1; l++){
249 for(j = 0; j < n; j++){
250 // Compute r.h.s. vector
251 mu3[j][l] = H3i[j][l+1]*q3[j][l+1];
252 for(i = 0; i < nm; i++){
253 mu3[j][l] = mu3[j][l] - AB[j][i]*H3i[i][l]*q3[i][l];
254 }
255 // Forward substitution
256 for(i = 0; i < n; i++){
257 mu3[j][l] = mu3[j][l] - Alpha[l-1][i][j]*mu3[i][l-1];
258 }
259 for(i = 0; i <= j-1; i++){
260 mu3[j][l] = mu3[j][l] - Beta[l][i][j]*mu3[i][l];
261 }
262 mu3[j][l] = Beta[l][j][j]*mu3[j][l];
263 }
264 }
265 // Compute the last n elements
266 for(j = 0; j < n; j++){
267 //Compute r.h.h. vector
268 mu3[j][N-1] = H3i[j][N]*q3[j][N];
269 for(i = 0; i < nm; i++){
270 mu3[j][N-1] = mu3[j][N-1] - AB[j][i]*H3i[i][N-1]*q3[i][N-1];
271 }
272 // Forward substitution
273 for(i = 0; i < n; i++){
274 mu3[j][N-1] = mu3[j][N-1] - Alpha[N-2][i][j]*mu3[i][N-2];
275 }
276 for(i = 0; i <= j-1; i++){
277 mu3[j][N-1] = mu3[j][N-1] - Beta[N-1][i][j]*mu3[i][N-1];
278 }
279 mu3[j][N-1] = Beta[N-1][j][j]*mu3[j][N-1];
280 }
281
282 // Compute mu3: Backward substitution
283
284 // Compute last n elements
285 for(j = n-1; j >= 0; j--){
286 for(i = n-1; i >= j+1; i--){
287 mu3[j][N-1] = mu3[j][N-1] - Beta[N-1][j][i]*mu3[i][N-1];
288 }
289 mu3[j][N-1] = Beta[N-1][j][j]*mu3[j][N-1];
290 }
291 // Compute all other groups except for the first n elements
292 for(l = N-2; l >=1; l--){
```

```
for(j = n-1; j >= 0; j--){
for(i = n-1; i >= 0; i--){
mu3[j][l] = mu3[j][l] - Alpha[l][j][i]*mu3[i][l+1];
}
for(i = n-1; i >= j+1; i--){
mu3[j][l] = mu3[j][l] - Beta[l][j][i]*mu3[i][l];
}
mu3[j][l] = Beta[l][j][j]*mu3[j][l];
}
}
// Compute the first n elements
for(j = n-1; j >=0; j--){
for(i = n-1; i >=0; i--){
mu3[j][0] = mu3[j][0] - Alpha[0][j][i]*mu3[i][1];
}
for(i = n-1; i >= j+1; i--){
mu3[j][0] = mu3[j][0] - Beta[0][j][i]*mu3[i][0];
}
mu3[j][0] = Beta[0][j][j]*mu3[j][0];
}

// Compute z3

// Compute first n+m elements
for(j = 0; j < nm; j++){
z3[j][0] = q3[j][0];
for(i = 0; i < n; i++){
z3[j][0] = z3[j][0] + AB[i][j]*mu3[i][0];
}
z3[j][0] = -H3i[j][0]*z3[j][0];
}
// Compute all other groups except for the last n+m
for(l = 1; l < N; l++){
for(j = 0; j < n; j++){
z3[j][l] = q3[j][l] - mu3[j][l-1];
}
for(j = n; j < nm; j++){
z3[j][l] = q3[j][l];
}
for(j = 0; j < nm; j++){
for(i = 0; i < n; i++){
z3[j][l] = z3[j][l] + AB[i][j]*mu3[i][l];
}
z3[j][l] = -H3i[j][l]*z3[j][l];
}
}
// Compute the last n+m elements
for(j = 0; j < n; j++){
z3[j][N] = H3i[j][N]*(mu3[j][N-1] - q3[j][N]);
}
for(j = n; j < nm; j++){
z3[j][N] = -H3i[j][N]*q3[j][N];
}

// Compute res
for(j = 0; j < n; j++){
```

```
349 res[j][0] = z1[j][0] - x0[j];
350
351 }
352 for(l = 0; l < N+1; l++){
353 for(j = 0; j < nm; j++){
354 res[j][l+1] = z2[j] + z3[j][l] - z1[j][l];
355 }
356 }
357 for(j = 0; j < nm; j++){
358 res[j][N+2] = z2[j] - z1[j][N];
359 }
360
361 // Update lambda
362 for(j = 0; j < n; j++){
363 lambda[j][0] = lambda[j][0] + beta[j][0]*res[j][0];
364 }
365 for(l = 1; l < N+2; l++){
366 for(j = 0; j < nm; j++){
367 lambda[j][l] = lambda[j][l] + beta[j][l-1]*res[j][l];
368 }
369 }
370 for(j = 0; j < nm; j++){
371 lambda[j][N+2] = lambda[j][N+2] + beta[j][N]*res[j][N+2];
372 }
373
374 // Exit condition
375 res_1 = 0;
376 for(l = 0; l < N+3; l++){
377 for(j = 0; j < nm; j++){
378 res_1 = fmax( res_1, fabs(res[j][l]));
379
380 }
381 }
382
383 res_2 = 0;
384 for(j = 0; j < n; j++){
385 res_2 = fmax(res_2, fabs(x0[j] - z1[j][0]));
386 res_2 = fmax(res_2, fabs(z2[j] - z1[j][N]));
387 }
388
389 if(res_1 < tol){
390
391 for(j = 0; j < m; j++){
392 pointer_u[j] = z1[n+j][0]*scaling_inv[j+n];
393 }
394 e_flag = true;
395 done = true;
396 }
397 else if(k >= k_max){
398 e_flag = false;
399 done = true;
400 }
401
402 }
403
404 }
```