Proyecto Prácticas Procesadores Integrados: Self-Balancing Robot basado en Arduino

Carlos Bailón[1] and Antonio F. Díaz[1]

Departamento de Arquitectura y Tecnología de Computadores.
Universidad de Granada
cbailon37@correo.ugr.es, afdiaz@ugr.es

Resumen Procesadores Integrados es una asignatura que se imparte en el Grado en Ingeniería Electrónica Industrial donde los estudiantes adquieren la capacidad de desarrollar pequeños proyectos basados en microcontrolador. En este artículo se presenta uno de estos proyectos: un robot auto-balanceado basado en Arduino. Se describen los elementos principales a considerar en el diseño hardware y software. La ventaja de esta configuración es que permite evaluar un sistema de control PID en tiempo real a partir de la información obtenida por un acelerómetro y un giróscopo implementados en un sensor MEMS.

Palabras clave: Control PID, microcontrolador, Arduino, Sensor MEMS.

Abstract Integrated Processors is a course in the Electronic Engineering Degree where students acquire the ability to develop small proyects based on microcontrollers. In this paper we present one such projects: a self-balanced robot based on Arduino. The main elements to consider in the hardware and software design is described. The advantage of this configuration is that allows evaluating a PID control system in real time from information obtained by a accelerometer and a gyroscope implemented on a MEMS sensor.

Keywords: PID control, microcontroller, Arduino, MEMS sensor.

1. Introdución

La asignatura Procesadores Integrados se imparte en 3° del Grado en Ingeniería Electrónica Industrial de la Universidad de Granada y se centra en el estudio de microprocesadores, microcontroladores y procesadores de señales digitales (DSP). En la introducción se muestran las características principales de éstos y a continuación se empieza con el bloque relacionado con los microcontroladores, evaluando las posibilidades que ofrecen. Se estudian diversos microcontroladores para definir criterios de selección óptimos en función de las características del diseño y se analiza la información que ofrecen los fabricantes de microcontroladores de sus productos. También se analizan las

etapas para el diseño y se revisan buses y distintos métodos de interconexión tanto con otros sistemas como con dispositivos, evaluando las posibilidades de conexión de elementos de entrada y salida así como las tecnologías de memorias empleadas con microcontroladores. La ventaja de comenzar con el bloque de microcontroladores es que el estudiante adquiere la base suficiente para empezar a realizar prácticas en un corto de periodo de tiempo debido a la rápida curva de aprendizaje, por lo que el resto de la asignatura puede comprenderse desde un punto de vista más práctico.

Además, en la asignatura se estudian: Los elementos internos de los procesadores para incrementar sus prestaciones basadas en el paralelismo interno del procesador, las características de los procesadores segmentados y superescalares, cómo el tratamiento de las dependencias influye en el diseño del cauce de instrucciones, cómo la emisión de instrucciones desordenada en los procesadores superescalares reduce el número de ciclos que como media requiere una instrucción para su ejecución, técnicas de ejecución especulativa e identificar ciertos elementos necesarios en la implementación de los procesadores modernos, como por ejemplo las estaciones de reserva, o el buffer de reordenación, las limitaciones en cuanto al rendimiento de los procesadores segmentados y superescalares y elementos que penalizan su rendimiento, las extensiones multimedia incorporadas en los procesadores actuales, por qué los procesadores multihebra mejoran las prestaciones, qué elementos internos incorporan los procesadores para dar soporte al sistema operativo, la diferencia entre paralelismo de datos y paralelismo funcional.

También se estudian otros elementos directamente relacionados como son: cómo se implementan los buses en los sistemas basados en microprocesadores y el uso de los chipsets, diferencias entre los distintos buses y conocer sus prestaciones y forma de funcionamiento, buses de alto rendimiento para el procesador, la necesidad de disponer de una jerarquía de memoria así como el concepto de memoria virtual como solución al problema de la capacidad de almacenamiento, distintos tipos de memoria empleados para la memoria principal así como los principios básicos de las memorias caché, la importancia del diseño de las E/S en las prestaciones globales del sistema, la estructura y principios de funcionamiento de los controladores de E/S, el impacto de las interrupciones y como se gestionan, las distinta técnicas de E/S.

Finalmente se analiza la arquitectura interna de los DSPs y cómo está orientada al tratamiento de señales, las posibilidades de aplicación de los DSPs así como las diferencias de arquitectura frente a los procesadores de propósito general.

Partiendo de estos conocimientos, en la parte práctica los estudiantes deben desarrollar como proyecto un sistema electrónico basado en microcontrolador. En este sentido las plataformas basadas en Arduino permiten que los estudiantes adquieran rápidamente dicha capacidad ya que existen gran cantidad de documentación y recursos. En este artículo se presenta uno de los trabajos desarrollados para la evaluación de la parte práctica de la asignatura. En particular el desarrollo de un robot autobalanceado basado en Arduino.

Básicamente, este robot se mantiene en equilibrio sobre dos ruedas. El microcontrolador monitoriza su posición vertical mediante un acelerómetro de 3 ejes y un giróscopo de 3 ejes. Dicha información es la que se utiliza en el PID que controla los motores en tiempo real, permitiendo mantenerse vertical. En la Figura 1 se muestra el aspecto final que tiene el sistema propuesto.

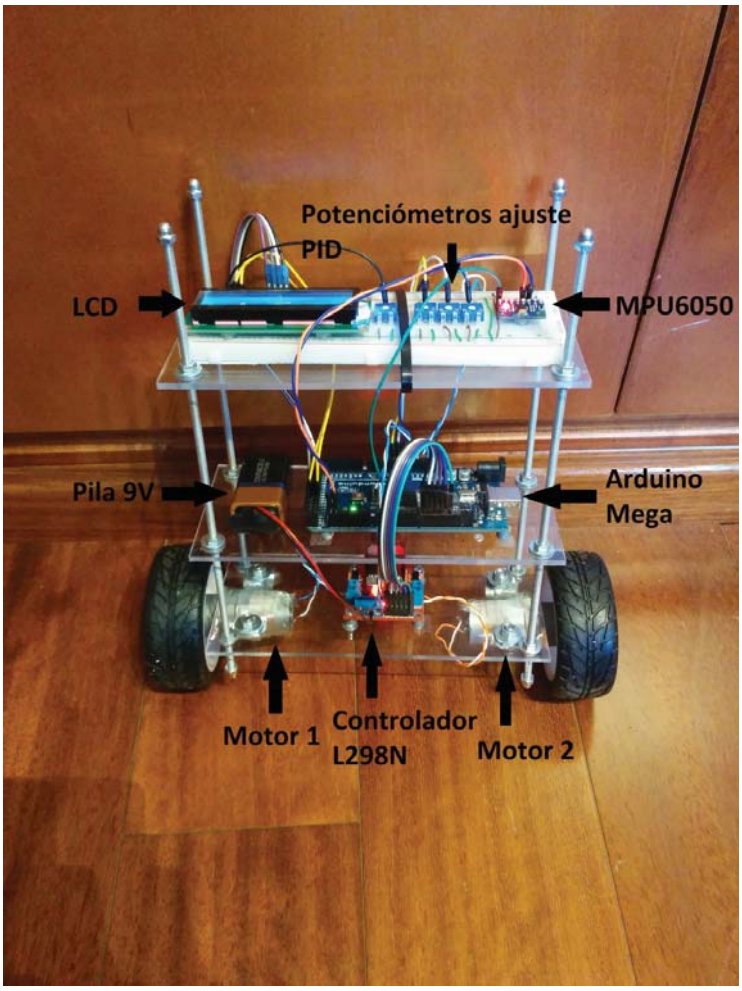

Figura 1. Aspecto del robot auto-balanceado.

2. Descripción hardware del sistema

Los elementos principales del sistema son:

- Microcontrolador Arduino Mega 2560
- Sensor MEMS MPU6050
- Motores y puente H
- Pantalla y calibración

La plataforma Arduino

Para este diseño se ha utilizado el Arduino Mega 2560 que está basado en el microcontrolador ATmega2560 con una frecuencia de 16 MHz, aunque se podía haber utilizado otras placas de Arduino más reducidas.

MPU6050

El MPU6050 es un circuito MEMS (Micro-Electro-Mechanical System) que combina un acelerómetro de 3 ejes y un giróscopo de 3 ejes para medir las inclinaciones y rotaciones e incluye un DMP (Digital Motion Processor). Esta unidad permite realizar complejos cálculos con las medidas captadas por el chip, ahorrando tiempo y trabajo al microcontrolador. Este elemento se encarga de la captación de datos así como de procesar información de posibles magnetómetros externos. La Figura 3 muestra una pequeña placa con el MPU6050.

El chip va orientado de forma que su eje Y sea perpendicular al robot, de modo que sólo es necesaria medir la inclinación en ese eje. La conexión se realiza mediante I2C, los pines SDA (datos) y SCL (reloj), más los de alimentación y un pin de interrupción (INT0). Si se desea modificar la dirección I2C del dispositivo se dispone de otro pin (AD0), que modifica el último bit de dicha dirección, B110100X, donde X indica el estado de este pin.

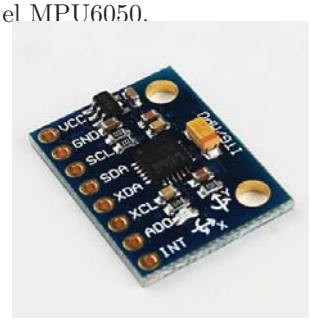

Figura 2. MPU6050

En nuestro sistema, captamos los datos de entrada del acelerómetro en unas coordenadas conocidas como Yaw-Pitch-Roll (figura 3), que son usadas para medir rotaciones en aeronáutica. De este modo, solo utilizamos la segunda coordenada (eje Y), que será nuestra señal de entrada. Este componente se encuentra en el nivel superior del robot, insertado en una protoboard con más componentes, de manera que se reduce el cableado ya que la alimentación a 5V es común.

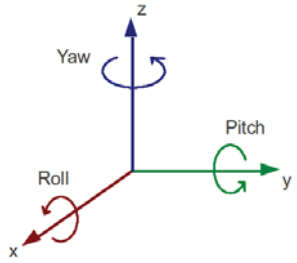

Figura 3. Sistema de coordenadas.

Motores

Permiten el movimiento de las ruedas del robot para su equilibrio y se controlan mediante salidas del microcontrolador. En este diseño se utilizan dos motores con sistema de reducción incorporado. Trabajan entre 3

y 9 V y dan una velocidad de salida nominal de 281 rpm (a 6V). Es fundamental que estos motores ofrezcan un gran torque, ya que nuestro objetivo es que el robot ofrezca una respuesta rápida y robusta ante cambios en la inclinación, para lo que es necesaria bastante fuerza. Las ruedas utilizadas son de 8 cm de diámetro.

Puente H

Para el control de los motores se utiliza una placa con el circuito integrado L298N (Figura 4). Este chip contiene puente H, que mediante una estructura simétrica de pares darlington y diodos permite el control de la velocidad y el sentido de giro de los motores. La placa necesita alimentación externa acorde con la necesaria para los motores, a través de la cual se extrae también la alimentación de 5V para toda la electrónica (mediante un regulador linear LM7805). Podemos controlar ambos motores mediante 3 pines cada uno, 1 para la regulación de velocidad (se utiliza el PWM de Arduino) y 2 para el control del sentido de giro (ambos conectados, el motor se enclava, ninguno el motor se mueve libremente en punto muerto, y uno solo determina el sentido de giro). Podemos utilizar esta alimentación para alimentar la placa de Arduino. Los motores y el controlador están en el nivel más bajo del robot.

Figura 4. Puente H basado en L298N.

Alimentación

Para la alimentación se utiliza una pila de 9V que se conecta directamente a la placa controladora de motores, de la cual se extrae la alimentación regulada a 5V para el resto de la electrónica del robot. La pila está situada en el nivel medio junto a la placa de Arduino.

La Tabla 1 muestra el presupuesto de los componentes utilizados en el proyecto.

3. Descripción software del sistema

Recursos utilizados

El MPU6050 se comunica por I2C, y su utilización es compleja, por lo que se han usado dos bibliotecas, *I2Cdevlib* y *MPU6050_6Axis_MotionApps20*, desarrolladas por Jeff Rowberg, que facilitan enormemente la captura de datos válidos para procesamiento. La biblioteca para I2C *Wire* también es necesaria para utilizar las dos anteriores.

Por otro lado, para la monitorización del proceso se envían datos por puerto serie, tanto el ángulo de inclinación del robot como las constantes del control PID mientras se está calibrando, siempre y cuando se seleccione la opción correspondiente al inicio del código. Además, cada 5 segundos se guardan los

Elemento	Cantidad	Precio unidad (€)	Precio total (€)
Chasis y elementos mecánicos			
Ruedas (Ø 55 mm)	2	4,90	9,80
Ejes ruedas (5 cm)	2	0,15	0,30
Juntas ejes motor - rueda	2	0,30	0,60
Motor CC con reductora 6V 281 rpm	2	6,39	12,78
Abrazaderas fijación motores	2	0,32	0,64
Planchas metacrilato (180x80x3 mm)	3	1,58	4,74
Varilla roscada (20 cm)	4	0,32	1,28
Tuercas varillas roscadas	24	0,03	0,72
Tornillos fijación placas	8	0,03	0,24
Topes varillas roscadas	4	0,06	0,24
Total chasis			**31,34**
Electrónica			
Arduino Mega 2560 Rev3	1	35,00	35,00
Acelerómetro MPU6050	1	4,20	4,20
Motor Driver Board L298N	1	5,70	5,70
Protoboard MB-102	1	3,09	3,09
Cables macho-hembra	15	0,03	0,45
Pila 9V	1	3,28	3,28
Total electrónica			**51,72**
Precio Final			**83,06**

Tabla 1. Presupuesto del proyecto.

datos de inclinación del robot en la memoria EEPROM externa, por si es necesario hacer una gráfica a largo plazo de los niveles de inclinación. Para ello se han utilizado interrupciones, mediante el Timer 3 de Arduino Mega (evitando así bloquear funcionalidades de otro Timer). Se han utilizado las bibliotecas *EEPROM* y *TimerThree*.

Para el control de motores se ha creado una biblioteca propia, que se ha llamado *LMotorController*. Esta dispone de más funcionalidades que las que usa el robot (ya que éste sólo hace un movimiento rectilíneo, y la biblioteca permite girar y mover ambas ruedas con velocidades diferentes), pero está pensada para poder usarla de forma más genérica. Por último, se ha incorporado la configuración del watchdog, para lo que se utiliza la biblioteca *wdt*, incorporada en la IDE de Arduino, y que con simples órdenes permite configurar el watchdog del sistema. En este caso se ha configurado para lanzar un reset si no se reinicia en 8 segundos.

Control PID

La placa de Arduino Mega 2560 se encuentra en el segundo nivel del robot, y es la encargada de hacer los cálculos para el procesamiento de la señal de entrada.

Este procesamiento se hace mediante un control PID, para lo que recurrimos a la biblioteca *PIDv1*, que se encuentra en la página oficial de Arduino. Para ajustar el control, se utilizan 3 potenciómetros de 10k (uno para cada una de las

constantes), situados en la placa de prototipado. Se monitorizan los valores de las constantes por puerto serie, de forma que se pueda conocer cuál es el valor óptimo.

Para realizar el primer ajuste, se ponen a cero los potenciómetros, por lo que el robot no hace nada (constantes cero, no hay control PID), y se va aumentando el que corresponde a la constante proporcional *Kp*, hasta que alcanzamos un valor para el cual los motores del robot responden a los datos del acelerómetro. En nuestro caso ha sido $Kp = 70$. Una vez fijada, pasamos a variar el potenciómetro correspondiente a la constante integral *Ki*. Ésta se encarga de reducir el error en estado estacionario, por lo que la aumentamos hasta el valor para el cual la oscilación de los motores es pequeña. En nuestro caso es $Ki = 240$. Finalmente ajustamos la constante diferencial *Kd*, que reduce el sobredisparo, hasta el punto en el que el movimiento del robot es suave y no hay picos de fuerza. $Kd = 1.9$. Una vez ajustadas manualmente se introducen en el código las constantes obtenidas para simplificar la ejecución del programa. En cada bucle se llama a un método que implementa el algoritmo PID y nos da la señal de salida.

Descripción del código

Se comienza incluyendo las bibliotecas necesarias y definiendo una serie de variables que nos permiten hacer una monitorización del sistema. Son 3 variables que dependiendo de si valen 1 ó 0 habilitan ciertas partes del código (para ver los valores del acelerómetro por puerto serie, para ver las constantes PID si se están sintonizando de forma manual y para el propio ajuste manual). A continuación se declaran los objetos *mpu*, *pid* y *controlMotores* para las bibliotecas correspondientes, y se declaran las variables necesarias para el funcionamiento del sistema. Respecto al MPU6050, se incluyen variables para detectar interrupciones mediante el pin de interrupción asignado, otras para comprobar el estado del dispositivo tras cada operación y otras más para trabajar con la memoria FIFO que incorpora el sensor, la cual nos permite hacer un almacenamiento de los datos para su lectura. Se ha configurado también un búfer para almacenar los datos. El resto de variables corresponden a los vectores necesarios para obtener los datos y procesarlos.

Después se definen las variables utilizadas para el PID y el controlador de motores y una variable de temporización para almacenar valores de tiempo y ejecutar la captación de datos para sintonización manual en ciertos intervalos (si ha sido seleccionada previamente). En concreto compara constantemente el tiempo mediante *millis()* y ejecuta la rutina cada segundo.

En el *setup*, lo primero que se hace es iniciar el bus I2C y el puerto serie, y después se realizan todas las inicializaciones y conexiones necesarias en el sensor MPU6050. También se ha incluido una rutina para detectar errores en el inicio, con su correspondiente indicación en el monitor serie. Después se configuran el control PID, el Timer para el envío de datos a la EEPROM y el watchdog. En el *loop* lo primero que se hace es el cálculo de los valores de salida del control PID y la actuación de los motores, así como el bucle correspondiente a la variable de temporización señalada anteriormente. Tras ello se efectúa la obtención de los datos, con una rutina para el desbordamiento de la memoria

FIFO (lo cual no debería ocurrir). Para obtener los datos, primero se obtienen en *cuaterniones*, que son los valores primarios (conocidos como *raw values*) que obtiene el sensor. Estos valores son un tipo de coordenadas complejas para medidas de rotaciones, previas a la invención de otras medidas como los ángulos de Euler o las coordenadas Yaw-Pitch-Roll.

Por último se transforman a coordenadas YPR y se muestran en tiempo real por el monitor serie si se ha seleccionado dicha opción al comienzo del código. Tras ello se actualiza el valor de entrada del control PID y se reinicia el watchdog. Una vez fuera del *loop*, se define el método *envíoDatos*, que se ejecuta en cada interrupción, y que cada 5 s guarda el valor del ángulo en la memoria EEPROM. El hecho de que se configure cada 5 s es porque la EEPROM tiene un número limitado de escrituras por celda, que no queremos rebasar. También se declara el método *sintonizacionManual* (correspondiente al bucle temporizado anterior, y solo se ejecuta si hemos puesto a 1 la variable correspondiente al principio), que nos permite captar los datos de las constantes PID a través de los potenciómetros en cada iteración del bucle.

4. Conclusiones

En este artículo se ha presentado un trabajo de prácticas de la asignatura Procesadores Integrados. Este trabajo consiste en un robot autobalanceado basado en Arduino. Uno de los principales problemas que plantea este diseño es la posición del acelerómetro. Al estar insertado en una placa de prototipado, la fijación no es tan exacta como si estuviera atornillado, por lo que hay que asegurarse de que no se varía la inclinación del mismo si el robot sufre alguna caída, ya que cualquier variación de la inclinación supone el fin del equilibrio. En general, el comportamiento del robot una vez calibrado es bastante estable.

Referencias

1. Asignatura Procesadores Integrados. Grado Ingeniería Electrónica Industrial. Universidad de Granada. https://swad.ugr.es/es?crs=7267
2. Ogata K. Ingeniera de control moderna. Pearson-Prentice Hall, 5a ed. (2010)
3. Arduino: http://www.arduino.cc/
4. Jeff Rowberg.*MPU6050_6Axis_MotionApps20*. https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
5. Biblioteca PID Arduino. http://playground.arduino.cc/Code/PIDLibrary