LABORATORIO DE CONTROL (4º GITI)

Práctica 3

Microcontroladores programables

Departamento de Ingeniería de Sistemas y Automática
Universidad de Sevilla

Índice

1 Objetivos

En esta práctica se pretende que el alumno conozca cómo se puede llevar a cabo el control de un sistema a través de microcontroladores programables industriales.

Los controladores industriales basados en circuitos electrónicos son de gran utilidad dado su bajo coste, su confiabilidad y la gran cantidad de herramientas informáticas y de desarrollo que proporcionan una implementación fácil de algoritmos de control. Los circuitos microcontroladores son cada vez más rápidos, pequeños y con mayor potencia de cálculo.

La práctica consiste en controlar la posición de un servomotor mediante un controlador PID. Tanto el servomotor como la implementación del controlador PID se conocen por prácticas anteriores, por lo que esta práctica se centrará en el conocimiento y uso del microcontrolador (Arduino DUE), incluyendo el conexionado y las herramientas para su programación.

Los pasos a efectuar en la práctica son:

- Obtener los conocimientos básicos del nuevo microcontrolador Arduino DUE.
- Conexión del microcontrolador Arduino DUE al PC.
- Programación de un controlador PID con las herramientas de programación del microcontrolador.
- Conexión del microcontrolador al servomotor y puesta en marcha.
- Descripción de los resultados prácticos obtenidos.

A continuación se describen cada uno de estos puntos. Una vez realizada la práctica el alumno dispone de dos semanas para entregar por EV una memoria individual de la misma, respondiendo a las cuestiones planteadas en el formato de memoria proporcionada para esta práctica.

2 El microcontrolador Arduino DUE

La placa Arduino DUE es distinta a la mayor parte de placas compatibles con Arduino porque no lleva el habitual microcontrolador ATMEGA, sino uno bastante más potente basado en ARM; por otra parte tiene dos salidas analógicas (nivel de tensión, no solamente salidas PWM, que también tiene). Debido al cambio de arquitectura, el software básico que se utiliza (compilador, bibliotecas) es completamente distinto al de las otras placas, aunque desde la interfaz de programación de Arduino no se aprecie normalmente tal diferencia. La última versión estable del software de desarrollo se puede descargar e instalar desde http://arduino.cc/en/Main/Software, donde hay soporte para ambas arquitecturas.

La placa microcontroladora Arduino Due está basada en una CPU con arquitectura ARM, en concreto la Atmel SAM3X8E ARM Cortex-M3. Es la primera placa de Arduino que trabaja con un microcontrolador ARM de 32 bits. La placa tiene 54 entradas/salidas digitales (de las cuales 12 pueden usarse como salidas PWM), 12 entradas analógicas, 4 UARTs (puertos series) y dos convertidores de digital a analógico.

El núcleo ARM de la placa mejora las prestaciones de otras estructuras típicas de microcontroladores de 8 bits. Especialmente relevantes son sus siguientes características:

- Núcleo de 32 bits.
- Reloj de la CPU a 84 Mhz.
- 96 KB de SRAM.
- 512 KB de memoria Flash para código.

En esta práctica se ha elegido esta placa Arduino frente a otras, además de por sus mejores prestaciones, porque permite la lectura y escritura de señales analógicas. En concreto, dispone de 12 entradas analógicas y dos salidas analógicas. Todas ellas permiten digitalizar valores analógicos con 12 bits de resolución (4096 niveles distintos). Por defecto, la resolución está fijada a 10 bits, por lo que hay que inicializarla a 12 bits para poder utilizar el rango completo.

La placa dispone de dos conexiones USB externas. El "Programming port" provee un puerto COM virtual, que puede conectarse a un PC y sirve para programar el Arduino y comunicarse con él. Además, existe otro puerto USB, el "Native USB port", que provee de comunicación serie y que puede actuar como USB host, permitiendo la conexión de periféricos como ratones, teclados o “smartphones”.

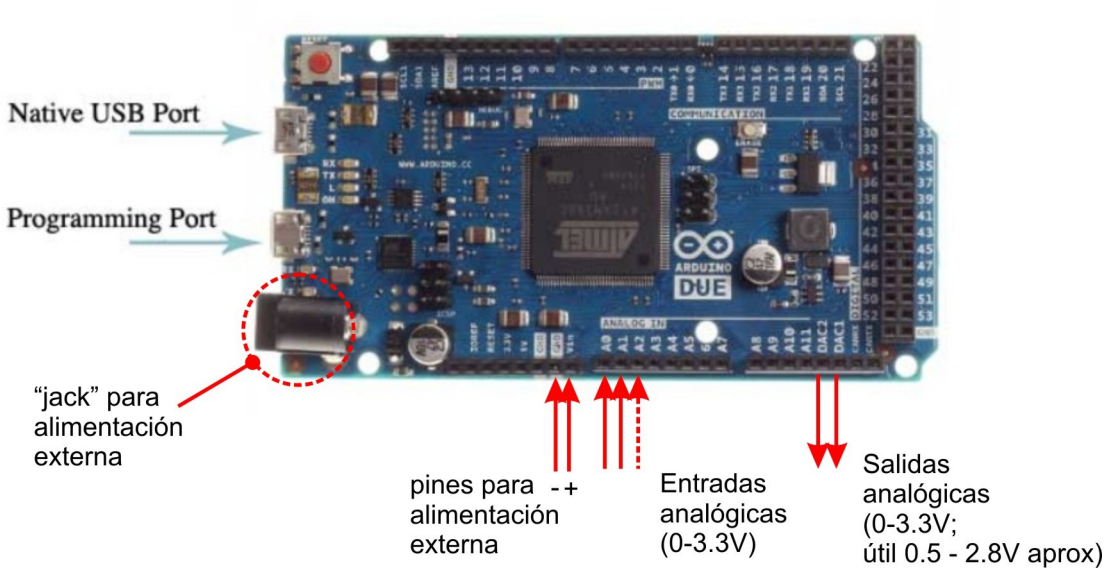

Figura 1. Placa Arduino DUE

2.1 Entradas y salidas. Conexión con el servomotor

Las entradas y salidas analógicas de la placa Arduino DUE se muestran en la Figura 1. Todas tienen en común un rango reducido de tensión, entre 0 y 3.3V, por lo que nunca deben conectarse a tensiones de 5 Voltios, por ejemplo. Las salidas además no aprovechan todo el rango. La salida “0” no es 0 Voltios, y la salida “4095” (máxima) no llega a 3.3 Voltios. Por otra parte las entradas y salidas de los servomotores de prácticas, es de -10 a +10 Voltios, por lo que es necesaria una adaptación de niveles y rangos de tensiones. Para ello, se han integrado las placas Arduino DUE en una placa que adapta los niveles de tensión para que estén en el

rango -10-10 V.

Puesto que la placa de desarrollo que se utiliza en las prácticas ya tiene la adaptación de tensiones integrada para que el rango de tensión del Arduino sea transformado al del servomotor, la conexión se puede realizar directamente. Habrá que conectar con latiguillos los bornes de la placa de adaptación del Arduino con los terminales de entrada y salida del servomotor, con el objetivo de leer la posición y velocidad desde el Arduino, y fijar la señal de control para el servomotor.

3 Conexión al PC del Arduino DUE

Para programar el microcontrolador Arduino DUE se utilizará el software de desarrollo de Arduino, que se comunicará a través del “Programming Port” con el PC. Por tanto, una vez conectado el “Programming Port” del Arduino al PC, podemos probar el microcontrolador con un programa simple de prueba llamado “Blink.ino”. Para ello, realizar los siguientes pasos:

- Conectar la placa al PC por el “Programming Port” si no lo está ya.
- Abrir el software de desarrollo de Arduino.
- Pulsar en “Herramientas” y en “Placa”; seleccionar “Arduino Due (Programming Port)”.
- Pulsar en “Herramientas” y en “Port”; seleccionar el puerto que indique “Arduino DUE Programming Port” – por ejemplo, COM14 (el número puede variar de un PC a otro).
- Pulsar en “Archivo” y luego en “Ejemplos” para acceder a los ejemplos de programación. Seleccionar “Basics” y luego “Blink”. Se abrirá el código del programa.
- Para compilar el programa y grabarlo en la memoria de la placa, pulsar en “->”.
- Si todo va bien, aparecerán trazas en la pantallas y la última será “CPU reset”. En ese momento debería observarse como uno de los LEDs integrados en la placa de color ámbar parpadea a intervalos de un segundo.

No es necesario compilar y cargar el código cada vez que se prueba la placa; basta con encenderla y se ejecutará el último programa grabado en la memoria no volátil.

4 Programación del Arduino

Los programas en Arduino se llaman "sketch" y se pueden escribir en el editor del software de desarrollo de Arduino como ficheros con extensión .ino. La interfaz de dicho software también tiene un botón para comprobar errores de compilación en el sketch.

Los sketchs se escriben en lenguaje C. Normalmente consisten en dos bloques principales: el bloque `setup()`, que sólo se ejecuta una vez cuando se carga el programa y sirve para inicializaciones; y el bloque `loop()`, que sirve para implementar una iteración del programa y se ejecuta de manera cíclica.

Desde el menú del software de Arduino se pueden importar bibliotecas para funcionalidades especiales. Con esta función, se insertan automáticamente los `#include` necesarios en el fichero de código. El software de Arduino viene con muchas bibliotecas disponibles, pero las que se incluyan se cargan en la placa con el programa de usuario, por lo que es recomendable no incluir bibliotecas que no se vayan a utilizar.

En general, para implementar una actividad cíclica en Arduino habrá que incluir una función `delay()` en el bloque `loop()`, la cual hará al microcontrolador esperar un cierto tiempo de muestreo. Esta espera debe ser igual a la diferencia entre el ciclo deseado y el coste de ejecución de las operaciones a realizar. Este coste es el tiempo mínimo de muestreo, ya que el sistema no podría realizar las operaciones a mayor tasa. Para determinar ese tiempo mínimo de muestreo (T_{min}), se debe calcular el tiempo máximo que tarda en ejecutarse el bucle programado. Una forma sencilla de calcularlo es estimando la ejecución del bucle durante un tiempo largo comparado con el tiempo de muestreo deseado y contabilizar el número de iteraciones que da tiempo a ejecutar. Dividiendo obtenemos una estimación de T_{min}; si T_m es el periodo de muestreo, la espera será de $T_m - T_{min}$ unidades de tiempo para el `delay()`. Puede medirse directamente T_{min} y también el tiempo de ciclo resultante para comprobar que realmente es el deseado.

Para medir tiempos con precisión de microsegundos se puede utilizar la función `micros()`, y `millis()` para medir con precisión de milisegundos. Probablemente, la duración del bucle de control para esta práctica sea de microsegundos, por lo que necesitaremos esa precisión para calcular el retraso. La función `delayMicroseconds()` se puede utilizar para especificar un retardo con precisión de microsegundos.

Para una descripición detallada de las bibliotecas y funciones disponibles para el desarrollo de programas Arduino, se recomienda la web de Arduino [2].

4.1 Planificador cooperativo (“Scheduler”)

Para la placa Arduino DUE existe una biblioteca que permite crear varios bucles concurrentes, aunque la conmutación entre uno y otro debe hacerse cediendo voluntariamente el procesador. Hay que incluir la cabecera `<Scheduler.h>`; y para crear un nuevo “hilo” o bucle concurrente es preciso llamar a la función:

```
Scheduler.startLoop(func1);
```

Esto se realiza normalmente en el bloque `setup()`, que es el que inicializará todos los bucles concurrentes que queramos tener en nuestro programa. Durante la ejecución del programa, el bloque `loop()` actuará en paralelo con la función `func1`, que será un segundo bucle. Para ello bastará incluir algún “delay” tanto en el bucle principal como en `func1`, o utilizar la función `yield()`, que cede el control a la CPU para que se ejecuten otras tareas. La función `delay()` ya contiene un “yield” implementado internamente.

5 Visualización y captura de trazas

Los resultados que se imprimen por el “Programming Port” del Arduino pueden verse con la consola virtual incluida en el software de Arduino (pulsar “Heramientas” y “Monitor serie”), y también con dicha consola pueden enviarse datos al Arduino. Sin embargo, no es necesario utilizar el software de Arduino para monitorizar el puerto serie virtual que se crea con la conexión USB; también puede utilizarse cualquier otra aplicación que sea capaz de abrir y manejar un puerto serie, por ejemplo la herramienta PuTTY. Para ello debe cerrarse la consola

de Arduino si es que está abierta, ya que el puerto de serie debe quedar libre para PuTTY. No hace falta cerrar el software de Arduino porque sólo utiliza el puerto de programación para cargar el programa, el resto del tiempo lo deja libre.

Por tanto, la herramienta PuTTY puede utilizarse para leer y escribir datos en el puerto serie del Arduino, exactamente igual que se hace con la consola virtual del software de desarrollo de Arduino. La ventaja es que con PuTTY puede además capturarse en un fichero la salida de la placa:

- Abrir PuTTY; seleccionar como “Connection type”, “serial”, y como “Serial line” la que corresponda a la placa (COM14, por ejemplo. El número es variable de un PC a otro).
- Ir a “Session” y luego a “Logging”. Seleccionar “All session output” en “Session logging” y poner como “Log file name” el que se estime oportuno.
- Dar “Open” para abrir el terminal. Se podrá ver la salida de la placa.
- Cuando se haya capturado la traza basta teclear Alt-F4 y confirmar la salida de la consola.
- El fichero de “logging” contiene la traza, que puede abrirse con cualquier editor de texto como Wordpad. Si estos datos fueron escritos con el formato adecuado, también podrían cargarse directamente en Matlab.

Esta opción es bastante útil para capturar las señales de entrada y salida del servomotor y luego dibujar las gráficas correspondientes en Matlab. No obstante, también podrían digitalizarse las señales del servomotor con la tarjeta de adquisición de prácticas anteriores para monitorizarlas directamente en Labview.

6 Características del programa a realizar

El programa de control que debe realizarse en la práctica tendrá al menos tres bloques concurrentes:

- Bloque del PID, que incluye el código del controlador y se desbloquea cíclicamente. El tiempo de espera se calculará de manera que el bucle se ejecute con el tiempo de muestreo deseado. En este bloque **no debe aparecer ninguna llamada de impresión** (`Serial.println()`, por ejemplo), salvo de manera provisional para depuración del código, debido a que consumen un tiempo de cálculo muy grande.

- Bloque de entrada/salida de datos, que se desbloquea mediante la pulsación de una tecla. Este bloque permitirá el cambio de la referencia y los parámetros del controlador, y también conocer el error actual de seguimiento (diferencia entre la referencia y la posición). Para leer datos de teclado se pueden utilizar funciones como `Serial.parseFloat()`. La función `Serial.available()` comprueba si ha llegado algún carácter por el puerto serie.

- Bloque de cambio automático de referencia, que actúa cada varios segundos (suficientes para observar todo el transitorio). Este bloque actuará después de introducir nuevas referencias desde el bloque de entrada/salida. Mantendrá la referencia introducida durante varios segundos y luego la modificará a otro valor suficientemente diferente del primero, para permitir observar un transitorio controlado entre ambos

valores. Puede utilizarse por ejemplo su simétrico con respecto al valor central de la posición, que es aproximadamente 2047 en la escala del microcontrolador. Esta funcionalidad es conveniente, porque mientras se leen datos en el bloque de entrada/salida el controlador se desactiva y el cambio de referencia no puede controlarse adecuadamente.

El tiempo de cálculo o de ciclo que se necesita medir para programar el retraso del bloque del PID también puede imprimirse desde el bloque de entrada/salida, contando el número de iteraciones ejecutadas durante ese tiempo. En la implementación del PID hay que tener en cuenta adecuadamente que tanto las entradas como las salidas **no están centradas en cero**, sino aproximadamente en la mitad de la escala que va de 0 a 4095.

Se adjunta con la práctica un fichero de ejemplo llamado “ejemplo_due.ino” con la estructura que debería tener el sketch final de la práctica. Se puede partir de ese ejemplo para implementar los tres bloques concurrentes que se piden.

Para ajustar el PID se seguirá un método heurístico, como por ejemplo el siguiente:

- Desactivar los términos integral y derivativo utilizando valores adecuados de T_i y T_d.
- Aumentar K_p hasta conseguir una respuesta con poca sobreoscilación y oscilaciones moderadas.
- Utilizar T_i para anular el error en régimen permanente.
- Utilizar T_d para mejorar el transitorio.

Puede utilizarse LabView separadamente para capturar datos de la respuesta del sistema y así poder generar las gráficas correspondientes. También pueden volcarse los datos en un fichero de texto con PuTTY (como se ha explicado anteriormente) para posteriormente representar las gráficas en Matlab.

7 Referencias:

1. *Página de la placa:* http://arduino.cc/en/Main/ArduinoBoardDue

2. *Guía de iniciación:* http://arduino.cc/en/Guide/ArduinoDue

3. *Foro de la placa:* http://arduino.cc/forum/index.php/board,87.0.html

4. *Scheduler:* http://arduino.cc/en/Reference/Scheduler