

Uso de Simulink y Arduino para Prácticas de Robótica

J.E. Gil Lozano, A.J. Muñoz Ramírez, V. Torres López, y J.M. Gómez de Gabriel

Dto. Ingeniería de Sistemas y Automática

Universidad de Málaga

je.gil.lozano@gmail.com, {ajmunoz, vtorres, jmgomez}@uma.es

Resumen

En éste artículo se presenta el diseño de un equipo y un conjunto de prácticas para la docencia en asignaturas de robótica, mediante el uso de Simulink y los microcontroladores Arduino. Se han diseñado y construido un conjunto de plataformas móviles con un microcontrolador y se ha evaluado su rendimiento y la facilidad del uso de Simulink con las librerías de soporte de Arduino para su uso docente en grados de ingeniería.

Palabras clave: Docencia, Arduino, Robótica, Simulink.

1. Introducción

El aprendizaje basado en proyectos es un método de enseñanza que implica a los estudiantes en la resolución de problemas de ingeniería reales [10]. Para que sea eficaz, el profesor debe identificar un problema de una extensión adecuada y con una dificultad suficiente para estimular el proceso de aprendizaje. Si además, el proyecto está apoyado con equipos físicos, frente a los modelos de simulación, se proporciona al alumno una versión completa del problema a resolver, en lugar de limitar su trabajo a un modelo simplificado, a la vez que aumenta su motivación [11]. Desafortunadamente, muchos sistemas de desarrollo hardware son complejos y distraen al alumno y al profesor con la resolución de problemas hardware de bajo nivel que no suelen estar relacionados con el objeto de estudio.

Sin embargo, recientemente ciertas herramientas reconocidas en el ámbito académico, como *MATLAB* y *Simulink*, usadas en el diseño y simulación de sistemas de control de robots, se han visto potenciadas por la incorporación de capacidades para programar de forma transparente las plataformas de desarrollo hardware de bajo coste más populares, como son los sistemas *Arduino* [5], *Lego NXT*, *Beaglebone* y *Raspberry Pi*, entre otras [4].

La solución ideal tanto para el profesor como para el alumno es disponer de una herramienta de

alto nivel para el diseño de sistemas de control de robots, y su modelado, para simulación y generación de código que pueda ser evaluado sobre un robot físico de bajo coste [17]. En concreto, el objetivo de este trabajo es presentar la experiencia de la creación de unos equipos de prácticas de robótica de bajo coste programables con la herramienta *MATLAB/Simulink* en asignaturas de robótica. Para lo cual se ha diseñado y construido la plataforma móvil PIERO (Véase Figura 1).



Figura 1: Plataforma robótica educativa PIERO basada en *Arduino* programable mediante *Simulink*

Entre las materias impartidas por el Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga, existen diversas asignaturas relacionadas con la robótica, y el modelado y control de robots móviles y manipuladores. En éstas se incluyen prácticas de control, cinemática y dinámica, que a menudo se realizan mediante cálculos y simulación en *MATLAB* y *Simulink*.

El principal beneficio esperado es que el entorno de desarrollo elegido, al resultar familiar tanto a profesores como a alumnos, reduzca el tiempo necesario de preparación de prácticas y de aprendizaje debido a que no se requiere el aprendizaje de otros lenguajes ni herramientas de programación adicionales [8]. Asimismo se espera que la realización de las prácticas en laboratorio sea más eficiente debido a que el mismo modelo que se construye para simular el robot se utiliza para controlarlo. Por último un beneficio adicional esperado consiste en el reducido coste económico del hardware utilizado. Además, éste artículo pretende servir de ayuda o referencia a los profesores interesados en

el diseño de prácticas relacionadas.

En la sección 2 se puede encontrar una descripción de la plataforma móvil desarrollada para la realización de prácticas. A continuación se describe el diseño de un conjunto de prácticas y los principales aspectos a tener en cuenta en su diseño y realización. Posteriormente, en la sección 4 se evalúan los beneficios esperados. Por último se incluyen conclusiones sobre el uso de las herramientas propuestas para la docencia de robótica.

2. La Plataforma Móvil

El equipo de prácticas se ha diseñado en base a un robot móvil con los siguientes requerimientos:

- Coste reducido y de fácil fabricación mediante componentes comunes.
- Robusto y de tamaño medio.
- Simple y fácil de programar.
- Autónomo y que permita funcionamiento inalámbrico.
- Sistema de sensores configurable que incluya odometría.

2.1. Diseño mecánico

La plataforma posee una planta de forma circular con un diámetro de 30 cm y un sistema de locomoción diferencial con ruedas locas pasivas frontal y trasera. Las ruedas activas son de 100 mm y están accionadas por servomotores *EMG30* [16] comunes en tiendas de componentes de robótica. Puede realizar giros sobre su centro geométrico con radio de curvatura cero y sin deslizamientos.

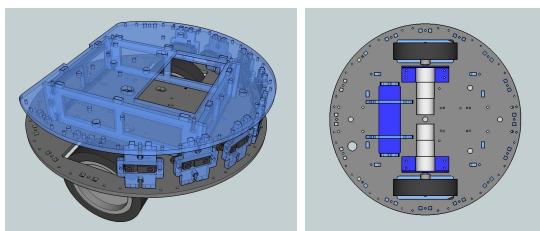


Figura 2: Diseño de la plataforma móvil PIERO realizado con *Sketchup* disponible *on-line*

En la parte inferior de la base se encuentra la electrónica de potencia y en la parte superior se ubican el microcontrolador y los sensores. Asimismo, posee un segundo nivel sobre el que se podría colocar un ordenador portátil.

Los diseños se han realizado mediante el software de diseño 3D *Sketchup* y se encuentran disponibles *on-line* en varias versiones [12]. Pueden exportarse fácilmente a formato DXF para su fabricación mediante corte por láser.

2.1.1. Módulos de sensores

PIERO posee un sistema modular de paneles laterales mediante el cual se pueden intercambiar los sensores. Actualmente se han construido módulos que soportan los clásicos sensores ópticos de distancia por infrarrojos de Sharp [19] y genéricos para añadir sensores de otros tipos.

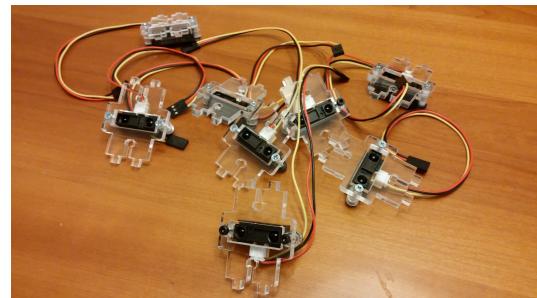


Figura 3: Paneles de sensores removibles

Los sensores utilizados [19] permiten medir distancias en base al ángulo de reflexión del haz de infrarrojos en un rango de 9 a 80 cm. Se alimentan a 5 V y proporcionan una salida analógica entre 0.4 y 2.7 V que se actualiza cada 38 ms, y que debe ser convertida para obtener las distancias.

2.2. Electrónica de potencia

Se han incorporado baterías *LiPo* de 3 celdas (11.1V) de 2400 mAh por la corriente que suministran y por su rapidez de carga. Proporcionan energía tanto a la electrónica de potencia como al microcontrolador, que reduce la tensión a 5V mediante un regulador propio. La conexión de tierra de las baterías es común a todo el sistema electrónico del robot. Están protegidas mediante un fusible y un interruptor en serie que desconectan completamente la corriente del robot.

El circuito de control de potencia para los motores es un L298 que incluye dos puentes H completos de 2 A. Si bien los motores pueden requerir más corriente (hasta 2.5 A). El disipador incluido en el módulo le permite funcionar sin calentamiento y se obtienen pares suficientes para esta aplicación. Módulos con este circuito (u otro compatible) pueden adquirirse a muy bajo coste.

Cada puente H del L298 se controla con tres líneas (Véase Figura 4): EN1(2) (Activación), conectada a una salida PWM; IN1(3) e IN2(4) que seleccionan la polaridad (dirección) y el tipo de parada (libre o con freno). Se requieren por tanto seis salidas digitales desde el *Arduino* hasta el módulo del L298. Se conectan mediante un único conector de 6 hilos etiquetado para facilitar al alumno la identificación de las señales y su programación.

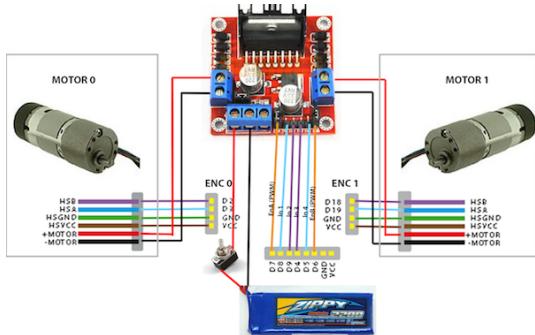


Figura 4: Esquema de la electrónica de potencia y conexiónado de los *encoders*

2.3. Electrónica de control

El control de la plataforma está basado en un microcontrolador *Arduino Mega2560* compatible con los microcontroladores soportados por *Simulink*. Se ha elegido éste modelo respecto a otros, como el *UNO*, debido a que posee un mayor número de puertos de entrada/salida, interrupciones y periféricos.

El sistema de conectores que identifica a la familia *Arduino* es adecuado para conectar otras placas de circuito impreso (denominadas *shields*) pero no resulta adecuado para la conexión de múltiples dispositivos directamente ya que sólo posee de un pin de tierra y alimentación.

Las *shields* de conexión comercial proporcionan infinidad de conexiones de tres hilos (GND, VCC y señal), si bien los dispositivos interesantes suelen requerir dos líneas de señal: módulos de comunicaciones, sensores I^2C , codificadores incrementales, etc. Por ello se ha adoptado un sistema de conexiones basado en la *GROVE Shield* [18], que usa estas conexiones de 4 hilos para todas las entradas y salidas, desarrollándose una *shield* propia.

2.3.1. PIERO Shield

Se ha diseñado la *PieroShield*. Una placa de conexiones compatible con *Arduino Mega2560* que se ha diseñado usando el software de diseño de circuitos impresos *Eagle* [6]. A esta placa de conexiones se le han añadido los siguientes puertos de conexión (todos ellos con líneas de tierra y alimentación incorporadas):

- 16 puertos digitales dobles
- 8 puertos analógicos dobles
- 2 puertos para *encoders*
- 1 puerto de mando para el módulo del L298
- 3 puertos UARTS
- 4 puertos I2C

Posee conectores de paso 0.1". Conectores hembras para los puertos digitales y conectores machos para las conexiones con el microcontrolador y resto de puertos. Esta placa puede verse en la figura 5.

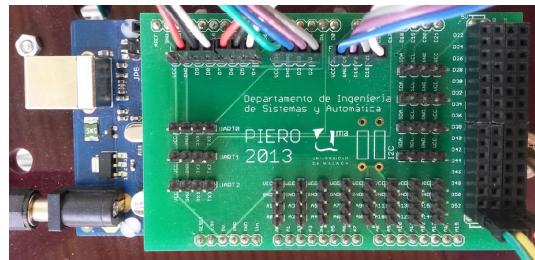


Figura 5: Placa de conexiones desarrollada para *Arduino Mega*

2.3.2. Codificadores angulares incrementales

Los codificadores angulares incrementales de los motores permiten controlar el movimiento de las ruedas del vehículo. Estos sensores permiten ejecutar trayectorias y estimar la posición mediante odometría. Esta característica es un requisito indispensable para la realización de prácticas de robótica móvil.

Los codificadores de los servomotores *EMG30* son de tipo magnético, con dos salidas digitales (A y B) de pulsos de 5 V que se conectan a entradas de interrupción del *Arduino*. Permiten obtener una resolución de 360 cuentas por vuelta si se decodifican en modo 4x (Considerando los flancos de subida y bajada de A y B). El microcontrolador posee seis líneas de interrupción externa [3]. Se han usado las líneas 2, 3, 18, 19 para las señales A y B de los *encoders* de las dos ruedas. La alimentación (5 V) se obtiene del regulador de la placa a través de conector de 4 pines mencionado en la sección 2.3.1.

2.4. Comunicaciones inalámbricas

El microcontrolador cuenta con cuatro puertos serie, de los cuales uno es utilizado por *Simulink* para la carga de programas y la ejecución en *modo externo*.

La ejecución en *modo externo* permite ejecutar el modelo de simulación simultáneamente en el hardware externo (el robot) y en el computador para hacer la simulación (ejecución) interactiva, mostrando valores y gráficas de evolución de las variables, y también para modificar valores de parámetros del modelo, lo que permite depuración y ajuste de controladores durante el funcionamiento del sistema. Esto se efectúa de manera transparente,

mediante un protocolo propio, sin necesidad de programar código para ello. Son necesarios más recursos por lo que no es compatible con los *Arduino UNO* o *nano*, debido a su reducida memoria de programa.

La alternativa a este modo consiste en programar las comunicaciones de manera explícita mediante los bloques que proporciona la biblioteca de funciones básicas de *Arduino*. Lo cual requiere de programación de código específico tanto en el microcontrolador como en el PC.

Para cumplir con el requerimiento de funcionamiento inalámbrico se ha elegido un módulo de comunicaciones *Bluetooth* [1] con capacidades especiales que permiten sustituir el cable de comunicaciones de manera transparente. Esto es posible debido a que: gestiona la señal de *reset* del microcontrolador para la carga de nuevos programas; y posee un sistema de auto-ajuste de velocidad de su *UART*. La función de reset requiere que el módulo maestro del PC sea versión 3.0 o superior.

Otras alternativas de comunicación inalámbrica consisten en adaptadores *Wifi* o *Zigbee* que presentan costes más elevados y/o no son transparentes.

2.5. Software

El soporte de *Arduino* en *MATLAB* incluye una biblioteca de bloques específicos que se encargan de gestionar las entradas/salidas. Internamente están programados en C y hacen la función de *drivers* del hardware externo. El resto de las funciones se implementan mediante los bloques estándar de *Simulink*. El conjunto de bloques proporciona acceso a: entradas/salidas digitales, entradas analógicas, salidas PWM, comunicaciones serie y entradas/salidas de pulsos para servos RC. No obstante, no proporcionan funciones que serían de gran utilidad como decodificación de codificadores incrementales, lectura de sensores ultrasónicos, comunicaciones *I²C* o SPI.

Existen unos ejemplos de programación incluidos y se pueden encontrar más recursos en MakerZone [14], una comunidad de usuarios creada por *MathWorks* para proporcionar soporte a los principales *targets*.

3. Prácticas

Se han diseñado una serie de prácticas que van desde el control de posición o velocidad de las ruedas a la ejecución de una tarea basada en una máquina de estados finitos. Estas prácticas se han planteado como las diversas etapas del desarrollo de un único proyecto de laboratorio consistente en el objetivo último de la programación de un robot

móvil completo.

3.1. Introducción a la programación de *Arduino* mediante *Simulink*

Inicialmente se realiza una práctica tutorizada para enseñar a los alumnos los aspectos relacionados con el uso y la configuración de la herramienta de desarrollo. Para ello se implementa el control de la orientación de un sensor de luz utilizando una fotocélula y un servo conectados al *Arduino*. Se implementa una estrategia básica de control proporcional y los alumnos lo ejecutan en modo remoto y externo para observar los valores de las variables del sistema y modificar los parámetros del controlador.

3.2. Control de la plataforma móvil

El objetivo de esta práctica es enseñar la electrónica de control del robot móvil y su manejo básico. En ella se estudia el esquema de conexiones y el alumno desarrolla sus propios bloques de control del vehículo. Asimismo, se instalan los módulos de comunicaciones inalámbricas y se prueba su funcionamiento. Al final de esta práctica, cada grupo ha desarrollado sus propios bloques que utilizarán en prácticas siguientes. Los bloques desarrollados son los de: envío de consignas a la electrónica de potencia de las ruedas y lectura de los sensores analógicos de distancia mencionados en la sección 2.1.1.

3.3. Control de posición

El objetivo de esta práctica consiste en aprender a controlar el movimiento de las ruedas del vehículo mediante el uso de los *encoders*. Para ello se les propone realizar un control de posición sobre uno de los motores utilizando un controlador PID. La decodificación de las señales de los codificadores incrementales no está incluida entre las funciones básicas por lo que se le proporciona al alumno el código C necesario para implementar su propio *driver* como una *S-Function* que hace uso de las interrupciones (Véase Sección 4.2). En la figura 6 se muestra un ejemplo de solución del modelo *Simulink*.

En este esquema se observa el lazo típico de control que incluye el bloque PID propio de *Simulink*. El signo de la salida del PID se utiliza para seleccionar el sentido de giro. Se ha incorporado un mecanismo para evitar la zona muerta del motor. El par deseado se envía como una señal PWM al puente H que controla el motor.

Las ganancias del PID se sintonizan mediante el método de Ziegler-Nichols [15] basado en la ganan-

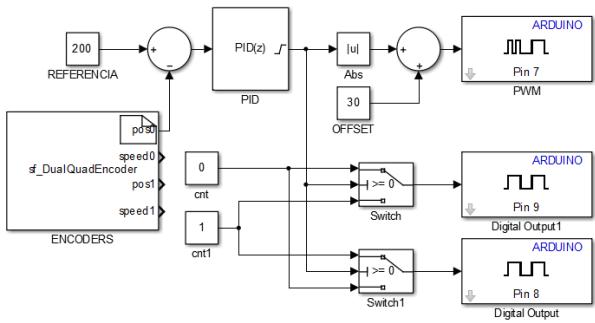


Figura 6: Esquema Control de Posición

cia crítica K_{cr} y en el periodo crítico P_{cr} . Estos valores se han obtenido gracias al uso combinado de los bloques *Scope* junto con la ejecución en modo externo.

Una vez sintonizado el PID se usa la posición de la rueda no controlada como referencia de posición de la controlada. En la figura 7 se observan la posición de la rueda no controlada (referencia) y la posición de la rueda controlada.

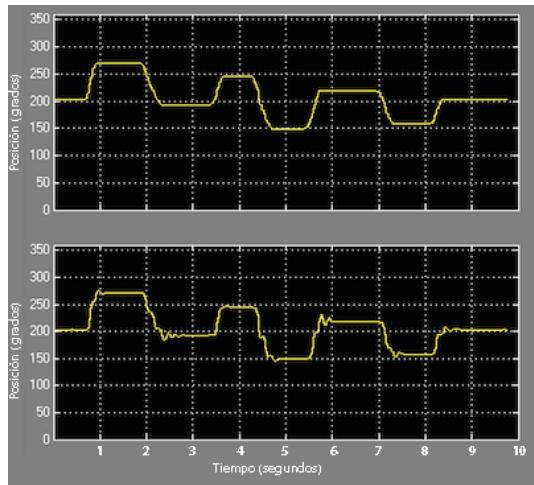


Figura 7: Resultado Control de Posición durante una ejecución de 10 s

3.4. Control de velocidad

Esta práctica consiste en controlar las velocidades de las ruedas del vehículo. Para ello se implementa un bucle de control velocidad de los motores. Ésta práctica enfrenta al alumno con el problema de la baja resolución de los *encoders*. Ya que el cálculo de la diferencia de posiciones entre ciclos aumenta el error de estimación a medida que aumentamos la frecuencia del control. Asimismo, la elevada desviación temporal sobre el periodo de muestreo *jitter* del *Arduino* introduce un gran ruido en la estimación de la velocidad.

Para obtener una mejor estimación de la veloci-

dad del vehículo se puede usar diferentes técnicas que requieren de una modificación del bloque de lectura implementado en la práctica anterior.

3.4.1. Cálculo de la velocidad basado en interrupciones

Esta técnica consiste en obtener la posición de los motores de cada cierto periodo de tiempo, mediante una rutina de interrupción asociada a un temporizador, independientemente de la frecuencia del control y calcular la velocidad utilizando la fórmula 1.

$$w (\text{rpm}) = \frac{\Delta pos \times 60}{T_m \times E} \quad (1)$$

donde T_m representa el periodo de muestreo en segundos y E el número de cuentas por rotación que ofrece el *encoder* (360 en este caso).

El periodo utilizado en esta práctica es de 180 ms, para lo que se ha configurado el *timer5* del *Arduino* con este propósito. El código necesario se ha incluido en la misma *S-Function* que lleva a cabo la decodificación de los *encoders*.

Se ha montado el mismo esquema mostrado en la figura 6 y se ha sintonizado el PID utilizando la misma técnica utilizada para control de posición. En la figura 8 se muestra el resultado de esta práctica.

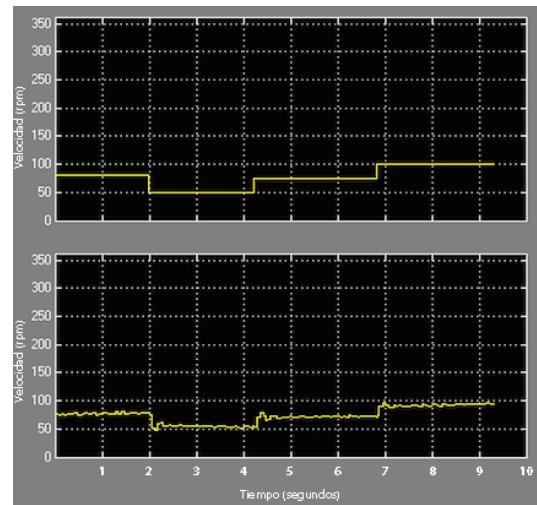


Figura 8: Resultado Control de Velocidad

3.4.2. Cálculo de velocidad basado en posición y tiempo

Una alternativa al cálculo de velocidad de los motores desarrollada en el apartado anterior, consiste en la inclusión de una marca de tiempo para el

dato de posición con ello se elimina el efecto de imprecisión de la medida debida a la desviación de exactitud de la señal del reloj, efecto conocido como *jitter*. Para el cálculo de la marca de tiempo se modifica el bloque funcional de lectura de *encoders* desarrollado en [9] de la siguiente forma:

- En la pestaña *Initialization* en *Sample mode* se cambia a *Inherited* con ello se consigue que el tiempo de muestreo del sistema de control sea el mismo que el de la lectura de la posición y poderlo cambiar sin la necesidad de recomilar el bloque funcional.
- En la pestaña *Output ports* se añade el parámetro *time* creándose así una segunda salida.
- En la pestaña *Output* se añade el código de la figura 9.

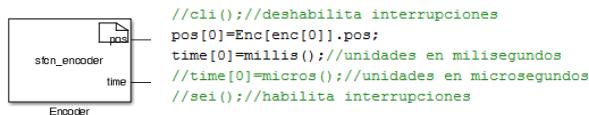


Figura 9: Nuevo Bloque *Encoder* y código modificado en la pestaña *Outputs*

Las diversas alternativas para calcular la marca de tiempo del dato posición utilizando el código nativo del *Arduino* y que se han estudiado en este apartado se encuentran como comentarios de código en la figura 9. Dichas alternativas dependen de la manera de extraer el valor del reloj del microcontrolador, usando la función *millis()* o la función *micros()*. Asimismo se ha probado a habilitar o deshabilitar las interrupciones durante esta lectura. Estas cuatro posibilidades se analizan en la sección 4.4.

3.5. Navegación reactiva

En esta práctica se implementa un sistema de control de navegación reactiva en el que la velocidad lineal y angular del vehículo es función de las lectura de los sensores delanteros de distancia. El alumno crea la función de evitación y el bloque con la cinemática inversa del vehículo para calcular las velocidades de las ruedas, que se envían a los bloques de control de velocidad.

3.6. Navegación basada en estados

El objetivo de esta última práctica es desarrollar un sistema de navegación basado en una máquina de estados finitos. En el ejemplo se muestra un sistema con dos estados. El alumno aprende a utilizar la herramienta *Stateflow* de *Simulink* y a integrarla en el bucle de control.

En uno de los estados la plataforma avanza, y en el otro, la plataforma gira debido a la presencia de un obstáculo. En la figura 10 se muestra esta máquina de estados, que cuenta con una entrada (*dist*) la distancia al obstáculo, que determina las condiciones de transición entre estados y dos salidas (*vel_izq* y *vel_der*) para determinar la velocidad de giro de cada rueda.

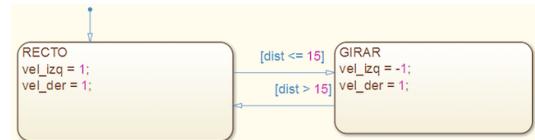


Figura 10: Diagrama de estados de la tarea de navegación programada en *Stateflow*

En el ejemplo mostrado se ha implementado la detección de los obstáculos mediante un sensor de distancias ultrasónico. Para ello se ha implementado una *S-Function* basándose en la biblioteca *NewPing* [7] de *Arduino*. En la figura 11 se muestra un ejemplo del diagrama completo.

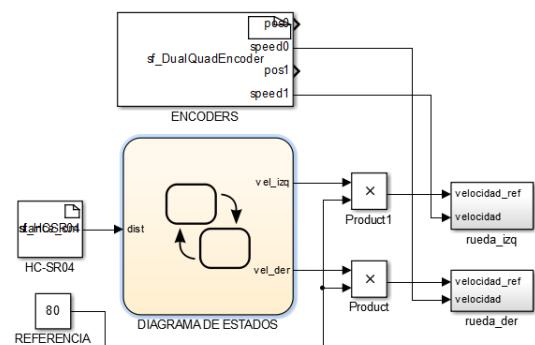


Figura 11: Diagrama *Simulink* del sistema de navegación basado en estados

4. Resultados

Durante el primer año del desarrollo de esta herramienta docente se ha trabajado en el diseño y construcción de la plataforma robótica. Pero también, en la creación del conjunto de prácticas y su evaluación mediante sesiones con alumnos y experimentales. A continuación, se incluyen algunos de los inconvenientes encontrados que condicionan el diseño e implementación del resto de las prácticas.

4.1. Pruebas

Se han realizado experimentos docentes en asignaturas de robótica con las prácticas 1 y 2 (descritas en las secciones 3.1 y 3.2). Los alumnos han aprendido a configurar el software y también han conectado sensores y actuadores al sistema *Arduino*.

Éstas prácticas también han servido para verificar el funcionamiento homogéneo de las plataformas construidas y de los sistemas de control de los motores. En estas prácticas los alumnos han tenido el primer contacto con el software de desarrollo y con el hardware, lo que les ha motivado visiblemente.

4.2. Lectura de codificadores incrementales

Para controlar el movimiento de los motores es necesario decodificar las señales de los codificadores incrementales. En el ejemplo oficial del módulo de soporte de *Arduino* se asume una única señal de pulsos que detecta la velocidad de avance de las ruedas en una dirección. Para estimar la velocidad se usa un bloque controlado por eventos como se muestra en la Figura 12 cuya entrada proviene de un bloque de entrada digital a la que se conecta el sensor óptico. No se implementa como una interrupción sino que se establece un periodo de muestreo (en el ejemplo es de 1 ms). Cada evento (subida o de bajada en la señal de disparo) provoca la ejecución del bloque que mide el tiempo transcurrido entre eventos a partir de la hora del sistema. Éste método presenta problemas de actualización con velocidades bajas y sobrecarga el microcontrolador con velocidades altas.

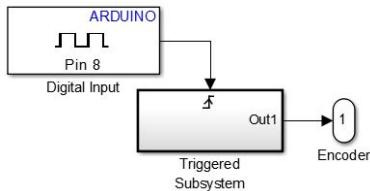


Figura 12: Bloque ejecutado por eventos para medir la velocidad de una rueda

Además, tampoco puede usarse en robots con tracción diferencial, donde las velocidades pueden cambiar de signo en los giros, y la capacidad computacional del microcontrolador limita el número de pulsos por vuelta del sensor. Por otra parte, este ejemplo menciona un control PID, que no existe, siendo a lo sumo proporcional.

No existe un medio oficial para la lectura de codificadores incrementales. Para resolver este problema se han diseñado y probado los métodos descritos en la sección 3.4 con resultados satisfactorios.

4.3. Pull-ups internos

Una entrada digital, como las que se usan para detectar los pulsos de los codificadores incrementales de las ruedas (Véase sección 2.3.2) u otros sensores con salida por transistor en colector abierto, puede poseer resistencias internas (denominadas *pull-*

up) para conseguir una lectura de un uno lógico cuando la entrada está en alta impedancia y un cero lógico cuando la entrada se conecta a tierra. Este sistema permite simplificar el hardware y el conexionado de múltiples sensores. En el lenguaje de *Arduino*, la activación de los *pull-up* internos se realiza escribiendo un uno en un puerto digital configurado como entrada, tal como se muestra en el código de la Figura 13. No obstante, las librerías de soporte de *Arduino* en *Simulink* sólo permiten especificar el número del pin de entrada a leer y no permiten usar dicho número en otro bloque de salida digital.

```

/* Configura el puerto pin como entrada */
pinMode(pin, INPUT) ;
/* Habilita resistencias pull-up interna */
digitalWrite(pin, HIGH) ;
  
```

Figura 13: Código para la activación de los *pull-up* internos

Sin embargo, las señales de los *encoders* que vienen de sensores de efecto hall requieren de *pull-up*. La soluciones pueden ser varias: incorporar las resistencia *pull-up* físicamente, programar una *S-Function* con un *driver* para activarlas o utilizar el *driver* ya programado [9] para la lectura de los *encoders*, que incluye dicho código y utiliza las interrupciones eficientemente. En nuestro caso utilizaremos esta última solución.

4.4. Rendimiento y *Jitter* en *Arduino*

Un buen determinismo temporal (bajo *jitter*) es necesario en un sistema de tiempo real para limitar el error cometido en cálculos que dependen del tiempo, como son la diferenciación (cálculo de la velocidad) y la integración (cálculo de la parte integral de un PID). Con el fin de medir el determinismo temporal del *Arduino* programado con *Simulink* se han realizado unos experimentos para registrar la desviación temporal con diferentes intervalos de ejecución. Éstos experimentos se han ejecutado en *modo externo* manteniendo el motor a la máxima potencia durante 100 segundos, las gráficas de dispersión de las mediciones se han obtenido con *MATLAB*, haciendo uso de la función *boxplot()*. Se ha marcado el valor de la media con un rombo y un asterisco.

El primer experimento sirve para estimar el *jitter*, calculándolo con ayuda de la marca de tiempo para un tiempo de muestreo del sistema de 100 ms y con el *Arduino* ejecutando un código simple (Véase figura 14). La menor dispersión del valor viene dada por el uso de la instrucción *millis()* con las interrupciones habilitadas pero sin grandes variaciones con respecto a las demás.

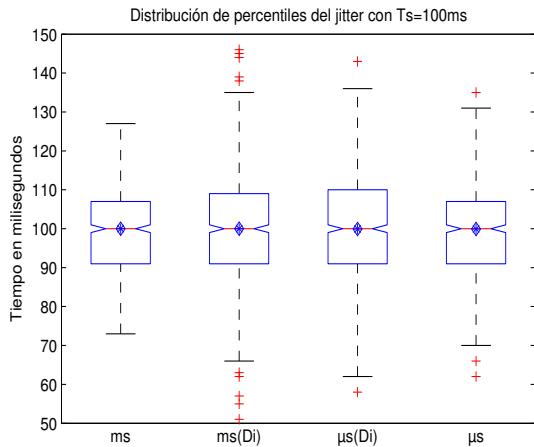


Figura 14: Desviación del valor del reloj para un tiempo de muestreo de 100 ms durante 100 seg. (*ms* indica el uso de la instrucción *millis()*, *μs* indica el uso de la instrucción *micros()*, (*D*) indica la deshabilitación de interrupciones en el *Arduino*).

La influencia del *jitter* en la estimación de la velocidad se ilustra la figura 15. Se presentan los valores de velocidad angular calculadas con y sin tener en cuenta la marca de tiempo. En la observación de la figura 15 cabe destacar cómo se pasa de unos errores de velocidad que oscilan entre del 15 al 50 % a un errores en la medida que van desde 0.5 al 3 %.

Los mejores resultados se han obtenido usando la función *micros()* con las interrupciones deshabilitadas. Los mismos resultados se obtuvieron de los experimentos realizados con un tiempo de muestreo de 50 ms, si bien todavía se acusaron más debido al aumento de la dispersión del *jitter* conforme se disminuye el tiempo de muestreo.

La utilización de un tiempo de muestreo de 10 ms con el mismo experimento obtuvo unos resultados sorprendentes en cuanto al valor del *jitter* obtenido. En lugar de centrar el valor entorno a esos 10 ms, el valor se centra próximo a los 20 ms. La gráfica de dispersión que se ilustra en la figura 16 demuestra que el *Arduino* es incapaz de alcanzar valores inferiores a 15 ms de tiempo de ejecución y eso para un programa muy simple, es decir, el rendimiento que se consigue con el código escrito en *Simulink* y en *modo externo* lo hacen altamente ineficiente y la supervisión del *jitter* muy necesaria.

4.5. Versiones de *MATLAB*

Si bien, puede utilizarse cualquier versión de *MATLAB* con soporte *Arduino*, las últimas versiones, han automatizado el proceso de incorporación del soporte de ésta y otras plataformas físicas me-

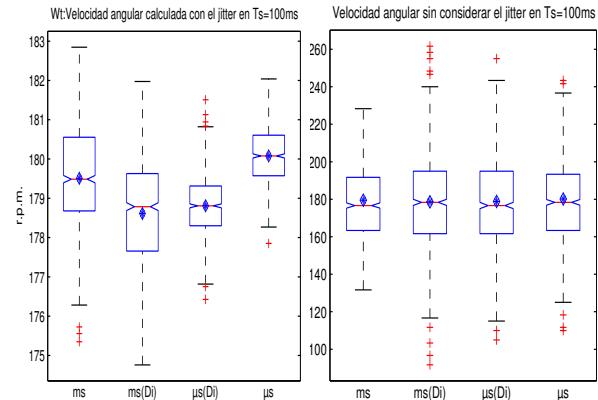


Figura 15: Dispersión en los valores de velocidad angular máxima del motor con o sin compensación de *jitter* para un tiempo de muestreo de 100 ms durante 10 seg (*ms* indica el uso de la instrucción *millis()*, *μs* indica el uso de la instrucción *micros()*, (*D*) indica la deshabilitación de interrupciones en el *Arduino*).

diente un repositorio centralizado que facilita su instalación y actualización.

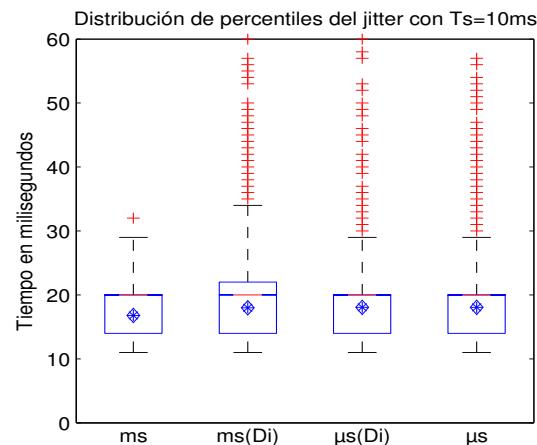


Figura 16: Desviación del valor del reloj para un tiempo de muestreo de 10 ms durante 100 seg. (*ms* indica el uso de la instrucción *millis()*, *μs* indica el uso de la instrucción *micros()*, (*D*) indica la deshabilitación de interrupciones en el *Arduino*).

Esta puesta al día, necesaria para usar las últimas características, puede resultar difícil en los actuales laboratorios docentes ya que *Mathworks* ha anunciado que a partir de la versión 2014b (2015 y posteriores) no será compatible con sistemas operativos de 32 bits.

No obstante la versión 2014a ha resultado muy inestable durante la realización de estas prácticas, por lo que, hasta nuevas versiones, se recomienda usar 2013b.

5. Conclusiones

De acuerdo a estos requerimientos planteados en la sección 2 se ha diseñado la plataforma y se han construído un total de seis unidades con un coste inferior a 200 € por robot.

Entre los objetivos iniciales de este trabajo se encuentran determinar si el uso de *Simulink* y *Arduino* permite la realización de prácticas basadas en proyectos para asignaturas de robótica de manera eficiente. En concreto se han encontrado los siguientes beneficios:

- Tiempo de adopción reducido. En términos de incremento del tiempo de preparación de prácticas por parte del profesor y de los alumnos (número de horas invertidas por el profesor para aprender la herramienta y horas de clase dedicadas a su uso).
- Adecuación. Diversidad del tipo de prácticas posibles dentro del campo de la robótica (identificación, control, cinemática, planificación, etc.).
- Bajo coste de los equipos de prácticas.
- Alto nivel de motivación. Tanto de los alumnos como del profesor.
- Generalidad. Múltiples posibilidades de aplicación de esta plataforma a otras asignaturas con lo que se reduce el coste económico y de tiempo.
- Utilidad futura. Proveen al alumno de una herramienta de diseño y desarrollo de sistemas relacionadas con la mecatrónica y la robótica que incluso puede ser replicada en su casa por su bajo coste.
- Otros beneficios. El aprendizaje basado en proyectos desarrolla las competencia de trabajo autónomo.

Asimismo se ha estudiado el determinismo temporal del sistema de desarrollo para comprobar las sus capacidades a la hora de implementar un sistema de control y se han mostrado los resultados en una tarea de estimación de la velocidad de las ruedas del robot.

Entre los beneficios que se esperan obtener durante el próximo curso, aparte de la evaluación del tiempo invertido en la realización del conjunto de prácticas completo, se encuentran la posibilidad de probar en sistemas físicos los resultados de herramientas de alto nivel como *Stateflow*, las utilidades de diseño de control borroso u otros.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Proyecto de Innovación Educativa de la Universidad de Málaga, con código PIE13-134, e Ingeniería UNO (<http://www.ingenieriauno.com/>).

Referencias

- [1] Adafruit, *Bluefruit EZ-Link*, <https://learn.adafruit.com/introducing-bluefruit-ez-link/overview>, Consultado Jun. 2014.
- [2] Apte, A. (2014), “Line Following Zumo Robot Using Simulink”, *Adafruit on-line tutorials*, <https://learn.adafruit.com/line-following-zumo-robot-programmed-with-simulink>.
- [3] Arduino, *Arduino Mega 2560*, <http://arduino.cc/en/Main/ArduinoBoardMega2560>, Consultado Jun. 2014.
- [4] Basso, M., Innocenti, G. and Rosa, A., (2012) “Simulink meets Lego: Rapid Controller Prototyping of a Stabilized Bicycle Model”, *52nd IEEE Conference on Decision and Control, Florence, Italy*.
- [5] Beni, N., Grottoli, M., Ferrise, F., Bordegoni, M. (2014), “Rapid Prototyping of Low Cost 1 DOF Haptic Interfaces, *IEEE Haptics Symposium*, pp. 479 - 483.
- [6] Cadsoft, *CadSoft EAGLE PCB Design Software*, <http://www.cadsoftusa.com/eagle-pcb-design-software>, Consultado Jun. 2014.
- [7] Eckel, T. (2012), “NewPing Library for Arduino”, *Arduino Playground*, <http://playground.arduino.cc/Code/NewPing>
- [8] Gartseev, I.B., Lee, L.F., and Krovi, V.N. (2011) “A Low-Cost Real-Time Mobile Robot Platform to support Project-Based Learning in Robotics and Mechatronics”, *Proceedings of 2nd International Conference on Robotics in Education. Vienna, Austria*, pp. 117-124.
- [9] Giampiero, C. (2013), “Writing a Simulink Device Driver block: a step by step guide”, *MATLAB File Exchange*.
- [10] Gómez-de-Gabriel, J.M., Fernández-Lozano, J.J. and García-Cerezo, A.J. (2012), “Evaluación de la contribución del aprendizaje basado en proyectos a la adquisición de competencias en estudiantes de Ingeniería en Automática y Electrónica Industrial” *Actas de las XXXIII Jornadas de Automática, Vigo*.
- [11] Gómez-de-Gabriel, J.M., Mandow, A., Fernández-Lozano, J. and García-Cerezo, A.J. (2011), “Using LEGO NXT Mobile Robots With LabVIEW for Undergraduate Courses on Mechatronics” *IEEE Transactions on Education, Vol. 54, I.1, pp. 41-47*.

- [12] Gómez-de-Gabriel, J.M. (2014) *Piero Mobile Robot Platform* <http://gomezdegabriel.com/wordpress/projects/piero-mobile-robot-platform/>
- [13] Lamár, K. and Kocsis, A. G. (2013), “Implementation of speed measurement for electrical drives equipped with quadrature encoder in LabVIEW FPGA”, *Acta Technica Corvinensis - Bulletin of Engineering*.
- [14] Mathwoks, *MakerZone Arduino, Raspberry PI and Lego Mindstorms Resources*, <http://makerzone.mathworks.com/>, Consultado Jun. 2014.
- [15] Ogata, K. (1999), “Ingenieria de Control Moderna (Spanish Edition)”, *Prentice Hall*, pp. 670 - 679.
- [16] Robot Electronics, *EMG30 mounting bracket and wheel specification*, <http://www.robot-electronics.co.uk/htm/emg30.htm>, Consultado Jun. 2014.
- [17] Rogers J.R. and McVay, R.C. (2012), “Graphical Microcontroller Programming”, *IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pp. 48 - 52.
- [18] Seeed Studio, *Grove Mega Shield* http://www.seeedstudio.com/wiki/Grove_-_Mega_Shield, Consultado Jun.2014.
- [19] Sharp, *GP2D12 Optoelectronic Device*, http://www.sharpsma.com/webfm_send/1203, Consultado Jun. 2014.