Modificaciones realizadas al Robot Multiplo N6 para permitir programación interactiva

Einar Felipe Lanfranco - LINTI (UNLP)
einar en linti.unlp.edu.ar
Joaquin Bogado - LINTI (UNLP)
jbogado en linti.unlp.edu.ar
David Vilaseca
vilaseca en gmail.com
Julián da Silva Gillig
julian.dasilva.gillig en gmail.com

LINTI (UNLP) - RobotGroup

Resumen Este trabajo describe las modificaciones realizadas al robot Multiplo N6 para que pueda ser usado como herramienta de enseñanza en un curso de programación para alumnos de escuelas de educación media.
El lenguaje utilizado es Python, un lenguaje que, al ser interpretado, tiene un alto nivel de interactividad, permitiendo a los alumnos ver inmediatamente los efectos de los algoritmos que desarrollan reflejados en el comportamiento del robot.

Keywords: Robot, Multiplo, N6, Python, Lihuen, Interactivo

1. Introducción

El modelo estándar de Multiplo N6 fabricado por RobotGroup[1] es un robot personal extensible con fines educativos de diseño y especificaciones libres. Es posible programar este robot en diversos lenguajes[1]. Sin embargo, (hasta el momento de realizar este trabajo) el N6 solo funcionaba en modo autónomo o *stand alone*, es decir que el único modo de ejecutar un programa del robot era grabándolo en la memoria del robot y luego dejando que el robot lo ejecutara.

La necesidad de interacción, que se detalla en la siguiente sección, hizo necesario modificar el robot para que acepte comandos en forma interactiva y para que pueda programarse ejecutando *scripts* de Python[2] en una computadora.

Este trabajo detalla las modificaciones que fueron necesarias, las cuales pudieron realizarse sin problemas dada la naturaleza de las licencias utilizadas en el diseño del Multiplo N6.

2. La necesidad de interactividad

El ciclo de desarrollo que se cumple habitualmente cuando se aprende a programar con lenguajes compilados es el siguiente: **programación - compilación - prueba - programación** (ver figura 1).

En la primer etapa de programación, el estudiante recibe el enunciado de un problema que debe solucionar y escribe el programa o una parte de este e intenta compilarlo. Durante el proceso de compilación, se detectan errores, generalmente de sintaxis, que deben corregirse para poder continuar con la compilación.

Una vez que la compilación es exitosa, se obtiene el o los binarios ejecutables. A partir de ellos, el estudiante realiza las pruebas. Los errores que se detectan en esta etapa dependen de los casos de prueba y son de naturaleza semántica, es decir, que el *software* desarrollado no cumple con lo que se pretendía de él.

Entonces debe volverse a modificar el código fuente, es decir vuelve a comenzar el ciclo, retornado a la etapa de programación para corregir aquellos errores detectados en el programa.

[1] Ver sección 'El robot Multiplo N6'

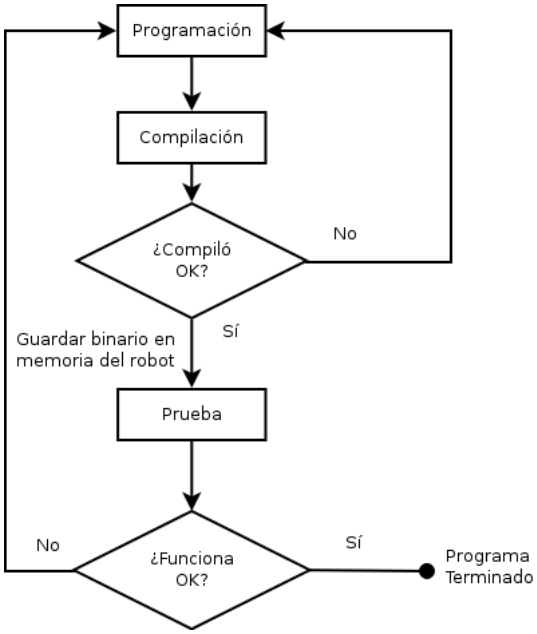

Figura 1. Ciclo de desarrollo de un programa en un lenguaje compilado

En la programación de robots autónomos, hay que agregar (al ciclo descripto) un nuevo paso que tiene que ver con la grabación en la memoria del robot del programa compilado antes de poder realizar las pruebas. Los errores de programación que no son detectados en tiempo de compilación, solamente se hacen evidentes una vez que el programa se está ejecutando en el robot.

En los lenguajes interpretados con consola interactiva, como Python, el esquema de programación es levemente diferente. Al no existir etapa de compilación, el ciclo se reduce a **programación - prueba - programación**. Este ciclo más simple no detecta errores antes de la prueba: los errores se detectan en tiempo de ejecución. Sin embargo, durante la etapa de programación, si el alumno no está seguro de la semántica de una sentencia, puede hacer una pequeña prueba en la consola interactiva. Este pequeño cambio en el esquema de programación reduce considerablemente la cantidad de errores semánticos y sintácticos en el programa final, ya que el estudiante puede ver el resultado de una determinada expresión antes de que el programa este completo.

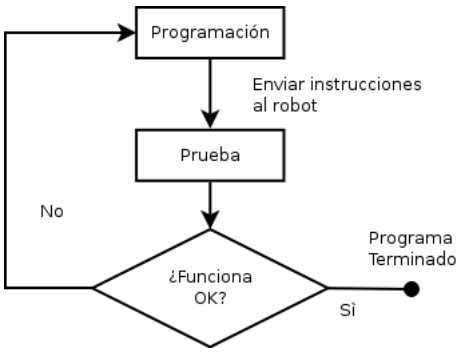

Figura 2. Ciclo de desarrollo de un programa en un lenguaje interpretado

Para que este modo de programación esté disponible en el robot Multiplo N6, fue necesario realizarle una serie de modificaciones a nivel *hardware*, *firmware* y *software*.

3. Experiencias previas

Desde el año 2009, el Laboratorio de Investigación de Nuevas Tecnologías Informáticas (LINTI)[3] de la Facultad de Informática de la Universidad Nacional de La Plata[4] trabaja con robots personales, con los fines de incentivar tanto el interés de los niños y jóvenes por aprender a utilizar la tecnología, como la curiosidad de docentes y adultos en el uso de la tecnología como herramienta para la educación. Realizando diversas charlas en escuelas de nivel medio de la zona de La Plata y alrededores se comenzaron a enseñar conceptos básicos de la programación, utilizando para ello el robot Scribbler[5].

El robot Scribbler se programa utilizando el lenguaje Python de forma interactiva, es decir, desde una consola de comandos. Permitía también la posibilidad de guardar las secuencias de pasos en archivos de manera análoga a los *scripts*.

Por dificultades en la importación de estos robots, no se han podido conseguir unidades nuevas desde principios de 2011. Sin embargo, se siguieron utilizando las unidades existentes. Un ejemplo de ello fueron las pasantías realizadas por alumnos del colegio Nacional de La Plata [5], [6],[7]. Durante estas pasantías, se comprobó que un grupo de alumnos del último año del secundario, sin experiencias previas en programación, fue capaz de construir programas sencillos con los conceptos principales de la programación, como variables y funciones, durante la primer clase, y programas más complejos con instrucciones de repetición y selección dentro de *scripts* en menos de tres clases (de tres horas cada una).

4. El robot Multiplo N6

4.1. Descripción general

El Multiplo N6 es un modelo de robot con fines educativos diseñado y ensamblado en el país por la empresa RobotGroup.

Para poder moverse, el robot cuenta con tres ruedas, dos delanteras y una trasera más pequeña que gira libremente. Las ruedas delanteras cuentan con tracción diferencial gracias a dos motores de corriente continua que se controlan por *software* independientemente uno del otro. Esto permite programar el robot para que vaya hacia adelante, hacia atrás o que gire variando la velocidad de una rueda con respecto de la otra.

Para poder "observar" el ambiente, posee varios sensores que le permiten percibir el entorno: dos sensores reflexivos y uno ultrasónico. Los reflexivos emiten luz infrarroja y detectan cambios de contraste entre las superficies claras y oscuras, permitiendo programar el robot para que siga una línea o detecte un borde, entre otras cosas. El sensor ultrasónico agrega la posibilidad de detectar obstáculos a disctancia, con precisión de centímetros.

Todos estos sensores, los motores y un *buzzer* que emite frecuencias en el espectro audible se conectan a una placa central basada en Arduino [8], con un microcontrolador programable ATmega32U4[9].

El robot, además, tiene algunos conectores libres para otros tipos de sensores, como sensores ópticos o potenciómetros, que pueden ser agregados adicionalmente en forma separada.

En la versión estándar del robot Multiplo N6, los programas se escriben en la interfaz de programación de Arduino o Minibloq [10], luego se compilan y por último se descargan al robot utilizando un cable USB. Los errores de programación que no se detectan en compilación solo se hacen evidentes cuando el programa ya fue descargado y se está ejecutando en el robot.

4.2. Especificaciones técnicas

Estas son las características más importantes del robot Multiplo N6:

- Alimentación
  - El controlador DuinoBot posee un sistema de alimentación que permite entregar hasta 12 V a los motores partiendo de sólo 3 pilas AA (o cualquier otra celda de 3.6 V, ya sea NiMH o Li-Ion). Además, eleva también la tensión de alimentación de la lógica del circuito, permitiendo la conexión de todo tipo de sensores estándar y otros accesorios Multiplo de 5V.
  - Cuando no se utilizan los motores, es posible alimentar también la placa desde el bus USB, contando la placa con selección automática de la fuente de alimentación.

- Sensores
  El robot cuenta con:
  - Dos sensores infrarrojos reflexivos. Se pueden utilizar tanto como para medir luz reflejada (por ejemplo, color), como para medir luz incidente. La aplicación más común es detectar una línea en el piso para poder seguirla.
  - Un sensor de distancia ultrasónico que permite identificar la distancia a un obstáculo con precisión de centímetros.
  - Un sensor interno de carga de la batería que permite conocer el voltaje de la fuente de alimentación.
- *Software*:
  El controlador DuinoBot es compatible con Arduino y puede ser programado de forma nativa *stand-alone* (escribiendo *software* de modo que quede residente en el microcontrolador del robot) en C++, Bitlash [11], y otros lenguajes de alto nivel. También puede ser utilizado en modo remoto desde lenguajes que corran en la PC (a partir de las modificaciones propuestas en este trabajo).
- Microcontrolador
  - Placa controladora DuinoBot basada en el microprocesador AVR ATMega32U4 y desarrollada por RobotGroup.
  - Memoria de programa: Flash de 32 KBytes.
  - Memoria de datos volátil: SRAM 2.5 KBytes.
  - Memoria de datos no volátil: EEPROM de 1 KByte.
  - Velocidad: 14-16 MIPS @ 16 MHz.
- Interfaz de usario
  - Un *buzzer* permite la generación de tonos a diferentes frecuencias.
  - LED de usuario.
  - LED indicador de 5V (alimentación lógica).
  - LED indicador de alimentación de motores (6V / 12V).
  - 4 LEDs indicadores de sentido de giro de los motores.
  - Un pulsador de Reset.
  - Un pulsador de Usario/Run.
- Comunicaciones
  - USB por *hardware* en el microcontrolador (lo cual permite utilizarlo como CDC, HID, entre otros, según el *software* que el usuario baje al microcontrolador). Esto permite utilizar la placa controladora también como *kit* de desarrollo para aplicaciones USB.
  - Puerto extra serie TTL con conector estándar para módulos de comunicaciones Multiplo (C0). En este puerto serie hay conectado un módulo XBee para comunicarse con una PC.
- Entradas y salidas
  - Seis entradas para sensores analógicos de 10 bits, con ficha estándar para los sensores Multiplo (S0 a S5). Las entradas analógicas pueden funcionar también como salidas digitales programables de hasta 40 mA (200 mA como máximo entre todos los pines).
  - Doble puente H MOSFET de alto rendimiento para motores de 2.5 a 13.5 V, corriente promedio de 1.2 A y pico de 3.2 A (M0 y M1).
  - Más entradas y salidas disponibles en los conectores estándar Arduino.

4.3. Licencia

La Licencia Pacifista[2] RobotGroup-Multiplo Pacifist License (RMPL) es esencialmente una licencia MIT modificada[3]. La principal modificación consiste en la adición de las cláusulas necesarias para impedir que la obra licenciada, así como cualquier obra o trabajo derivados de la misma, sean utilizados con fines bélicos, o con fines relacionados con la pena de muerte.

Tanto el diseño del Multiplo N6 como el *software* y el *firmware* que interviene en su desarrollo y que permiten que el robot funcione, se encuentran licenciados con RMPL.

[2] La licencia pude descargarse de multiplo.com.ar/soft/Mbq/Minibloq.Lic.v1.0.sp.pdf
[3] La licencia MIT original se puede obtener en http://www.opensource.org/licenses/mit-license

5. Modificaciones

Para poder utilizarlo de la forma que se pretendía, fue necesario modificar tanto el *hardware* como el *software* del equipo, estos cambios se describen en las dos subsecciones que siguen.

5.1. Modificaciones en el *hardware* del robot

Para que el robot pueda recibir comandos de forma interactiva fue necesario dotarlo de capacidades de comunicación inalámbrica. Para esto se agregó al robot un módulo de comunicaciones, el XBee Serie 1 compatible con el estándar de la IEEE 802.15.4 [12] que se conecta a uno de los puertos serie de la placa principal del robot, concretamente el C0. Este módulo permite a una computadora que tenga conectado un adaptador XBee enviar mensajes al robot.

Entre las nuevas posibilidades que nos brinda el XBee podemos mencionar que:

- Se permiten conexiones tanto punto a punto como punto a multipunto para controlar uno o más robots desde un mismo programa.
- El adaptador nos da un alcance efectivo para comunicarse con el robot de más de 15 metros, distancia más que suficiente para permitirnos controlarlo en todo el entorno de un aula grande.

Para el desarrollo de la API de alto nivel fue necesario acordar qué puerto pertenece a cada sensor. Por ello, se estableció el siguiente esquema:

- El sensor de línea derecho se conecta al puerto S5.
- El sensor de línea izquierdo se conecta al puerto S4.
- El sensor ultrasónico se conecta al puerto S3.
- El motor derecho se conecta al puerto M0.
- El motor izquierdo se conecta al puerto M1.
- Los puertos S0, S1 y S2 quedan disponibles para otros sensores.

5.2. Modificaciones en el *firmware* del robot

Para poder controlar el robot desde Python, fue necesario implementar un *firmware*[4] para ponerlo en modo esclavo, es decir, que esté pendiente de las órdenes enviadas a través del puerto de comunicaciones en lugar de funcionar en modo *stand-alone* como lo venía haciendo.

Esto se logró escribiendo un programa especial en la versión de C++ de la API de Arduino. Este programa es el único que se baja a la memoria del robot y queda residente. Una vez iniciado el programa, el robot queda a la espera de órdenes a través de la interfaz serial XBee. Cuando se recibe un comando, el *firmware* lo interpreta e inicia la ejecución de dicha orden.

Para enviar los comandos al robot, se adaptó el protocolo Firmata [13], extendiendo la API en Python, PyFirmata, de acuerdo a las necesidades específicas de la aplicación. Fue necesario implementar funciones para poder leer y escribir datos en los puertos de entrada/salida a los cuales se conectan los sensores reflexivos, los motores u otros sensores simples (aquellos que requieren solamente una lectura digital o analógica del valor de su entrada). Para los sensores más complejos fue necesario implementar funciones más sofisticadas que permitan utilizarlos, como es el caso del sensor ultrasónico, para el cuál se pide al dispositivo que realice una serie de operaciones a partir de las cuales se determina la distancia al obstáculo.

Cada mensaje enviado mediante el protocolo Firmata cuenta con un número de identificador que indica qué robot es el destinatario de dicho mensaje. Todos los robots con el mismo número harán caso del comando enviado. Este número identificador queda almacenado en la memoria del robot, por lo que también fue necesario implementar funciones para leerlo o escribirlo. Esto permite que los robots actúen ya sea de manera independiente o todos siguiendo las mismas órdenes.

También se implementaron comandos de reporte para determinar los números de identificación de aquellos robots que estén listos para recibir órdenes y desarrollaron comandos para reproducir tonos con el *buzzer*.

[4] El *firmware* es *software* que corre en el controlador del robot N6

6. La interfaz de programación de aplicaciones (API)

La API está dividida en dos partes. Una parte de más alto nivel está implementada en la clase **Robot** que permite enviar comandos simples al N6 para controlar los movimientos o para acceder a información de los sensores. La segunda parte es de más bajo nivel y se encuentra implementada en la clase **Board**.

Cuando se crea un objeto **Robot** (a través de la función `__init__()` o invocando a `Robot()`), se lo asocia a un objeto **Board**, como se puede observar en la figura 3. Cuando una instancia de **Robot** recibe un mensaje, por ejemplo `forward()`, este mensaje es delegado a la instancia de la clase **Board** correspondiente al objeto **Robot** que se está manejando, siendo esta la que finalmente envía la señal al robot para que avance a través protocolo Firmata.

El objeto Board es una abstracción de la placa controladora del robot y su sistema de conexión inalámbrica XBee. Los mensajes enviados a las instancias de **Board** incluyen un comando para el N6 (`forward()`, `backward()`, `getLine()`, etc), pero además incluyen un identificador numérico para la unidad al cual va dirigido dicho comando. Como cada instancia de **Robot** conoce su número de identificación y el **Board** al cual está asociado. Ese objeto **Robot** delega el envío del mensaje a su objeto **Board**. De esta manera, es posible controlar un grupo de robots Multiplo N6 utilizando un único módulo de comunicaciones XBee.

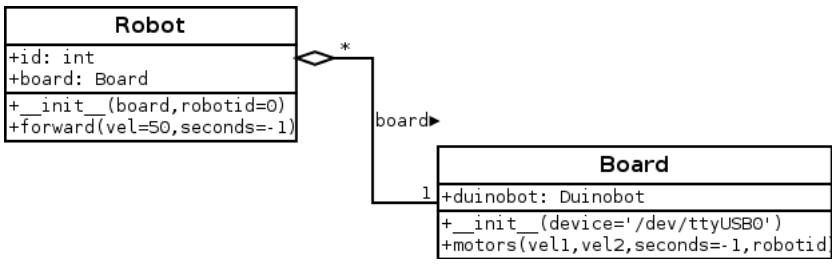

Figura 3. Diagrama sintético de clases de la API

Un ejemplo típico de una función de alto nivel de la clase Robot es `forward()` cuya implementación es como sigue:

```
def forward(self, vel=50, seconds=-1):
    '''El robot avanza con velelocidad vel durante seconds segundos.'''
    self.board.motors(vel, vel, seconds, self.robotid)
```

De esta forma, cuando un objeto de la clase **Robot** recibe el mensaje `forward()`, este delega el comportamiento en su objeto **Board** al cual está asociado. El mensaje `motors()` de la clase **Board** esta implementado como sigue.

```
def motors(self,vel1, vel2, seconds=-1, robotid=0):
    if(abs(vel1)<=100 and abs(vel2)<=100):
        self.board.send_sysex(4,[int(abs(vel1)), int(abs(vel2)),\\
              1 if vel1>0 else 0, 1 if vel2>0 else 0, robotid])
        if seconds!=-1:
            self.board.pass_time(seconds)
            self.motors(0,0,-1,robotid)
```

En este caso, el objeto de la clase **Board** delega el envío del mensaje a `self.board`, un objeto de la clase **DuinoBot**, subclase de un objeto de **PyFirmata**. El mensaje enviado es `send_sysex()`, un mensaje del protocolo Firmata y es el que finalmente es interpretado por el N6 para que las ruedas se muevan. El mensaje `send_sysex(self, sysex_cmd, data=[])` recibe dos argumentos: el primero, `sysex_cmd`, es el número del comando que se envía y varía dependiendo de a qué funcionalidad se quiere acceder. Por ejemplo, para mover el motor 0, el número de comando es 1, para mover el motor

1, el número de comando es 2 y para mover ambos a la vez, el número de comando es 4. Así también existen números de comando para acceder a los puertos en forma analógica o digital dependiendo de que tipo de sensores se hallen conectados. El segundo parámetro de `send_sysex()` es `data`, una lista de parámetros para el comando. Así, en el caso de la función `motors()`, los parámetros tienen que ver con la velocidad para ambos motores y el sentido de avance (hacia adelante o hacia atrás). En todos los casos es necesario enviar en el parámetro data el número de identificación del robot al cual se le envía el comando.

Los demás mensajes, ya sean de movimiento, de lectura de los sensores o de otros tipos, se ejecutan de manera análoga.

7. Integración con el proyecto Lihuen GNU/Linux

Desde 2005, en el laboratorio LINTI, se viene trabajando en una distribución de GNU/Linux llamada Lihuen GNU/Linux[14], orientada al ambiente educativo de niveles primario, secundario y universitario. Por este motivo y porque algunos de los involucrados en este proyecto trabajan también en el proyecto Lihuen, es que se decidió crear un paquete de instalación para la API del robot.

Lihuen GNU/Linux está basado en Debian GNU/Linux y por lo tanto utiliza el mismo sistema de paquetes en formato *.deb* [15]. Fue entonces que se decidió crear un paquete de instalación *.deb* que permitiera instalar el paquete mediante las herramientas típicas de instalación de paquetes para plataformas Debian como dpkg, apt, Gdebi o Synaptic.

A partir de mediados del mes de abril de 2012 está disponible en los repositorios de Lihuen GNU/Linux[5] en la rama experimental. También puede descargarse el binario para Debian GNU/Linux o Ubuntu o cualquier distribución basada en Debian desde `http://repo.lihuen.linti.unlp.edu.ar/lihuen/pool/experimental/main/robot_0.05_all.deb`

8. Conclusiones

Este trabajo permitió adaptar el Robot Multiplo N6 a un esquema de trabajo diferente para el que fue diseñado. La posibilidad de que el N6 trabaje en modo interactivo cuenta con las ventajas mencionadas más arriba. Sin embargo, este modo presenta una desventaja con respecto al modelo estándar, en el que el programa es guardado directamente en la memoria del robot. Esta desventaja tiene que ver con un incremento en el tiempo de respuesta del robot del orden de los milisegundos, que puede ser particularmente notable cuando se lee información de un sensor y se espera actuar de acuerdo a los valores leídos.

Sin embargo, el tiempo de repuesta es aceptable para tareas que no son críticas y no tienen un impacto significativo para el objetivo planteado.

Todas las modificaciones fueron realizadas en conjunto por los fabricantes de Robot Multiplo N6, RobotGroup y por las personas involucradas en el proyecto Lihuen GNU/Linux de la Facultad de Informática de la UNLP, y solo fueron posibles gracias a la licencia abierta del producto.

9. Trabajo a futuro

- Integrar la nueva API de Python a entornos orientados a niños más pequeños, como por ejemplo Minibloq, con el objetivo de sumar la programación gráfica basada en bloques a las ventajas de la programación interactiva.
- Explorar proyectos donde se pueda aplicar cierto nivel de colaboración entre robots, dado que tanto la plataforma de *hardware* como el *software* implementado aquí soportan este tipo de actividades.
- Ampliar las capacidades sensoriales de los robots, incluyendo sistemas de visión global basados en cámaras de bajo costo a través de la utilización de librerías como OpenCV [16] o SimpleCV [17].

[5] `http://repo.lihuen.linti.unlp.edu.ar/lihuen`

Bibliografía

1. ROBOTGROUP - Robótica para la acción. http://www.robotgroup.com.ar/web/.
2. Python Programming Language - Official Website. http://www.python.org/.
3. LINTI. http://www.linti.unlp.edu.ar/linti.
4. Facultad de Informática - Inicio. http://www.info.unlp.edu.ar/.
5. Página principal - robots. http://robots.linti.unlp.edu.ar/index.php?title=P %C3 %A1gina_principal.
6. Joaquín Bogado - Pasantes del colegio nacional. Programando con robots en el secundario, October 2011.
7. Programando con robots en el secundario. MAH04002.mp4 (Objeto video/mp4).
8. Arduino - HomePage. http://arduino.cc/.
9. ATmega32U4- atmel corporation. http://www.atmel.com/devices/atmega32u4.aspx.
10. Minibloq. http://blog.minibloq.org/.
11. Bitlash Online. http://bitlash.net/wiki/start.
12. IEEE 802.15.4. http://www.ieee802.org/15/pub/TG4.html.
13. Main Page - Firmata. http://firmata.org/wiki/Main_Page.
14. Sitio oficial de Lihuen. http://lihuen.linti.unlp.edu.ar/index.php?title=P %C3 %A1gina_principal.
15. The Debian GNU/Linux FAQ - basics of the debian package management system. http://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.
16. OpenCV Wiki. http://opencv.willowgarage.com/wiki/.
17. Machine Vision Made Easy - SimpleCV. http://simplecv.org/.