Toma el control de tu ladrillo MINDSTORMS™

por Oton Ribić

Después de ensamblar nuestro primer mensaje a un ladrillo EV3 en la anterior entrega, y por fin hacer girar el motor, es hora de ampliar el concepto y fijar una secuencia de comunicación fiable.

Respuestas y sincronización

Con cada mensaje que se envíe con éxito al EV3 siempre recibirá respuesta, incluso si no se pueden seguir las instrucciones del mensaje o si contiene errores. Esta respuesta solo se envía después de que el EV3 ejecute la instrucción completamente. Esto es clave para establecer una sólida comunicación sincronizada, es decir, asegurarse de que la instrucción se ejecute completamente antes de que comience otra. En la práctica esto es esencial.

Las respuestas del EV3 se reciben a través de un puerto de serie dedicado, designado para tráfico entrante,(lee el primer artículo en la serie para más detalles). Lo que hay que conseguir mediante el código es enviar un mensaje al EV3 Smart y luego, dependiendo de la implementación, esperar hasta que el ordenador recibe una notificación de que hay datos esperando o comprobar repetidamente si hay datos para leer. Luego se obtiene el mensaje del puerto de serie y se interpreta el contenido en caso de que se solicite algún valor (por ejemplo en el caso de la lectura de un sensor). Después se puede repetir todo el proceso.

Estas respuestas del ladrillo inteligente siguen las mismas reglas que los mensajes que se le envían — los primeros dos bytes indican el tamaño del mensaje que sigue, y el resto es el mensaje propiamente dicho. Por ahora no entraremos en analizar todas las posibles respuestas, ya que convertiríamos el artículo en una enciclopedia, pero veamos cómo es un típico mensaje de “hecho y todo bien” que envía el EV3.

Nº DE BYTE	00	01	02	03	04
VALOR DE BYTE	**03**	**00**	**00**	**00**	**02**
DESCRIPCIÓN	Largo		Mensaje		OK

El mensaje tiene un tamaño total de 5 bytes, de los cuales los primeros dos indican el largo del resto (de forma inversa). Los dos siguientes indican la ID del mensaje al que hace referencia y el último (con valor 02) confirma que se ha ejecutado con éxito. Recuerda: si se envían varios mensajes de forma asincrónica, es útil comprobar que esta confirmación incluya la ID del mensaje al que hace referencia; pero si usamos un método estrictamente sincronizado, solamente puede referirse al mensaje con ID cero que acabamos de enviar. O dicho en bloques de dos bytes: 00, 00.

En resumen, a menos que pidas que tu EV3 responda con algún feedback numerado, la respuesta será 3, 0, 0, , confirmando que todo está OK, o algo diferente si ha habido un error. Si quieres saber más acerca de esos mensajes de error y de cómo interpretarlos, hay información aquí: http://ev3.fantastic.computer/doxygen-all.

Codificar valores

Antes de meternos de lleno en mensajes más complejos es necesario un inciso con relación al sistema empleado para enviar y recibir valores numéricos del EV3. Siempre que se trata de valores pequeños, como porcentajes de potencia a usar en la rotación de un motor, se codifica directamente como un valor en bytes. Un ejemplo de esto se encuentra en la anterior entrega donde convertimos el número 75 directamente en su valor en bytes y lo enviamos empaquetado en el mensaje.

El enfoque es distinto cuando tratamos con números que especifican parámetros más complejos o valores más grandes. El principal ejemplo para este caso es hacer que un motor gire unos determinados grados, es decir, un cierto ángulo. Este valor puede ser demasiado grande para caber en un solo byte. Incluso un giro completo, de 360 grados, sería demasiado grande para caber en un solo byte. Por tanto, EV3 emplea estructuras de C, específicamente, estructuras de punto flotante de 4 bytes, para representar datos numéricos.

A menos que seas un programador experto, esto probablemente no significa gran cosa, así que déjame decirlo de otra manera: hay varios sistemas para codificar valores en bloques de ceros y unos y decodificarlos en sus números originales. Los ingenieros de LEGO® eligieron uno de ellos, y a menos que trabajes con C o algún lenguaje similar donde hay soporte nativo para esto, será tu propio código el que deba ejecutar esa conversión. Afortunadamente es un asunto bastante estandarizado y todo los lenguajes populares disponen de algún mecanismo elegante para hacerlo. Si por ejemplo usas Python, existe una biblioteca struct como parte del paquete estándar. (Si quieres más detalles, necesitarás la función struct.pack(‘f’,value).)

En cualquier caso, con una búsqueda en Google con las palabras clave “C structs, conversión de números” y el lenguaje de tu elección enseguida encontrarás lo que necesitas. Sólo asegúrate de interpretar el número como punto flotante, incluso si el número que usas no necesita decimales. Si quieres probar si tu conversión funciona correctamente, he aquí unos ejemplos; 360 debería convertirse en 0, 0, 180, 67, y -360 en 0, 0, 180, 195. Cero se convierte en cuatro ceros y 12.34 en 164, 112, 69, 65. Esta conversión es esencial para poder avanzar, así que asegúrate de que funciona correctamente antes de avanzar.

Hablando de avanzar, usaremos estos valores en la siguiente entrega, en la que combinaremos el conocimiento de los anteriores artículos y de este para lanzar algunos comandos complejos, como controlar el movimiento de un motor. ¡Hasta la próxima!
#