Toma el control de tu ladrillo MINDSTORMS (3)

Texto e imágenes por Oton Ribić

Si has seguido los dos artículos anteriores, aquí es donde empieza la diversión: ensamblar el primer mensaje “real” que se puede enviar de un ladrillo a otro para ejecutar una acción. Por ahora, nos centraremos en el sistema EV3, mientras que el NXT - que es un poco más complejo - se tratará más adelante.

Como se ha explicado anteriormente, hay un “encabezado” estándar para cada mensaje, es decir, una sección que anuncia cómo de largo es el mensaje y algunos otros parámetros. Esta vez, vayamos más allá y construyamos el cuerpo de nuestro mensaje para un EV3.

Piezas Iniciales

Comencemos intentando girar un motor, la cual probablemente sea la operación más común, en cualquier caso. Hay varias maneras de controlar un motor, incluyendo pasos específicos, cuanto girar, a que velocidad y potencia, inclinación, deslizamiento, etc., pero también hay una operación mucho más simple, que implica simplemente hacer que un motor gire indefinidamente. Este será nuestro punto de partida.

Todos los cuerpos de dichos mensajes se construyen especificando un conjunto de variables, es decir, los parámetros necesarios para la operación, como la velocidad y el motor en este caso, y otros datos para otros tipos de trabajos, y luego enviando la instrucción para que actúe en consecuencia con los datos recibidos. En nuestro primer caso, necesitamos especificar tres parámetros: a qué motor intentamos girar, a qué potencia o velocidad, y en qué dirección. Probemos con el primer motor, al 75% de potencia, en dirección positiva.

Primero, debemos calcular el código de nuestro motor a medida que se va utilizando a lo largo del mensaje. Es un trabajo bastante sencillo, exponenciar al cuadrado la potencia del número de motor, siendo el primero cero. Entonces, el primer código del motor es 1, el segundo 2, luego 4 y finalmente 8 para el cuarto.

Esto se puede usar para ensamblar la primera parte de nuestro mensaje, presentando la polaridad. Si queremos activar el primer motor en la dirección positiva, nuestro mensaje lo especificará con los bytes 167, 0, 1, 1. El 167, 0 es fijo; el siguiente número es un código del motor, y finalmente 1 especifica la dirección positiva. De lo contrario, el byte final habría sido 63.

Te estarás preguntando: ¿por qué estos códigos exactamente? ¿Por qué 167 de entre todos los números? Bueno, la respuesta más sencilla y directa es que este es el “idioma” que EV3 usa para comunicarse. Aunque pueda parecer arbitrario, existe una lógica subyacente que lo respalda, aunque profundizar en ella extendería este artículo más allá de lo tolerable, así que por ahora solo sigamos con los números que tenemos.

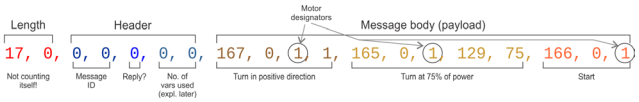
Entonces, nuestro mensaje comienza con 167, 0, 1, 1. Continuemos especificando la velocidad a la que queremos que gire. Los bytes que necesitamos son 165, 0, 1 (nuevamente el motor), 129, y finalmente, el porcentaje de potencia como un número - en nuestro caso, 75. Y luego, vamos con la instrucción final de “hacer que suceda” : 166, 0, 1.

Para el montaje!

De acuerdo, hemos recopilado todo lo que necesitamos y podemos comenzar a construir nuestro mensaje. Comencemos por concatenar estos fragmentos en el orden antes mencionado, para obtener 167, 0, 1, 1, 165, 0, 1, 129, 75, 166, 0, 1. Este es el cuerpo de nuestro mensaje que contiene tres segmentos, especificando la dirección de rotación, la velocidad deseada y las instrucciones para comenzar.

Pero, recuerda de la última vez...tenemos que agregar un encabezado a este cuerpo para construir un mensaje final y adecuado que el EV3 puede recibir e interpretar. Hay dos bytes que especifican cuál es el número de este mensaje en particular, en caso de que el EV3 pueda referirse a él al responder. Podemos mantenerlo libremente en 0, 0 para este ejercicio. Y el número de variables utilizadas en conjunto también es 0. Así que agreguemos tres ceros al frente de nuestro cuerpo de mensaje.

Y luego, está el último paso: para asegurarse de que el mensaje se haya recibido correctamente, el EV3 pBrick necesita saber cómo de largo es. Así que vamos a contar el número de bytes que contiene el mensaje final, dando la cifra de 17. Tenemos que representarlo como un número de dos bytes delante del mensaje completo, con el byte más pequeño (correctamente dicho en ingeniería: menos significativo) siendo el primero. Entonces el designador de longitud (¡que no cuenta!) resultará ser 17, 0. Esto hace que el mensaje final se vea como 17, 0, 0, 0, 0, 0, 0, 167, 0, 1, 1 , 165, 0, 1, 129, 75, 166, 0, 1.



Simplemente sigue adelante y envía estos bytes al puerto serie que has creado para este fin y al que conectaste el pBrick, de acuerdo con los datos de los artículos anteriores, ¡y el primer motor EV3 debería comenzar a girar! Por supuesto, la cuestión de cómo prefieres enviar estos bytes al pBrick depende totalmente de ti, es decir, de la elección del idioma o sistema que prefieras utilizar. C y sus derivados en su mayoría tienen soporte nativo para escribir directamente en los puertos; Python tiene una excelente biblioteca PySerial; la mayoría de los idiomas, al menos medio decentes, tienen al menos una forma de hacerlo. No te preocupes, los puertos serie son una tecnología tan estable y madura que no hay dudas sobre que tu sistema no pueda soportarlos.

Ten en cuenta que esta instrucción solo dice “comienza a girar” sin tiempo ni límite de giros. Por lo tanto, seguirá girando hasta que se indique lo contrario u, obviamente, se apague. En el futuro profundizaremos un poco más en una forma más compleja de la instrucción de girar, que permite al usuario controlar los movimientos y parámetros exactos.

Pero antes de eso, tendremos que examinar qué ha respondido realmente el pBrick al obedecer los mensajes anteriores. Aunque no siempre es obligatorio, y la clase de movimiento controlado, es impensable sin controlar las respuestas de pBrick y actuar sobre ellas. ¡Así que estad atentos!
#