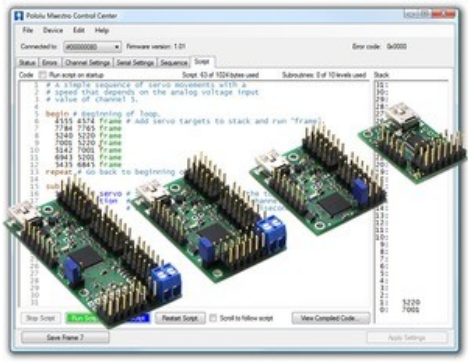
Pololu Maestro servocontrol desde USB



Guía de usuario

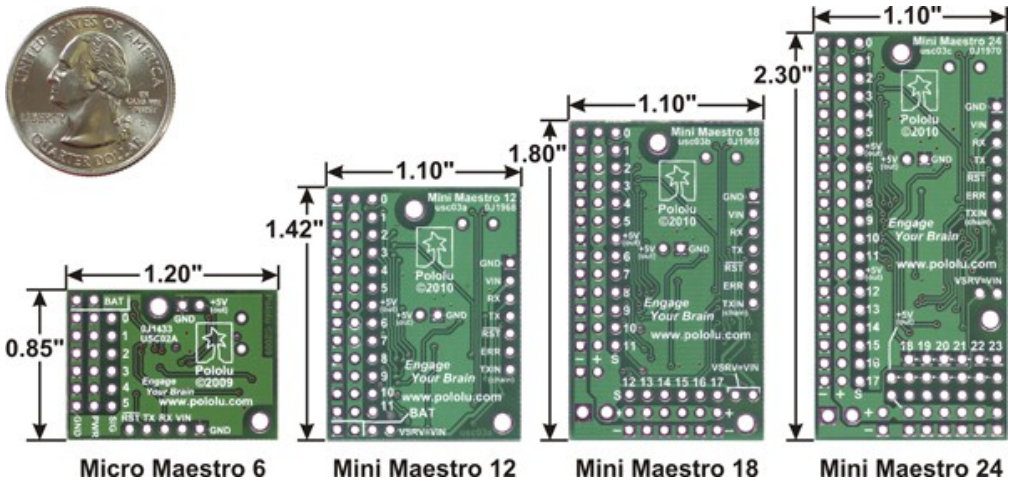
1. Información general

Los controladores Maestro son la segunda generación Pololu de controladores USB servo. La familia maestro consiste en cuatro controladores disponibles de manera ensamblado o en kit parcial:

Micro Maestro 6 Mini Maestro 12
Mini Maestro 18 Mini Maestro 24

Con tres modos de control: USB para conexión directa al PC, TTL serie para usar con sistemas embebidos y mediante scripts internos para su funcionamiento autónomo, control host de aplicaciones libres y canales que pueden configurar las salidas para servos de uso en radio control o control electrónico de velocidad (ESCs), salidas digitales o entradas analógico-digitales. Los Maestro son dispositivos versátiles para control de servos y con entradas y salidas analógico-digitales dentro de una placa muy compacta. Su gran precisión y alta resolución de pulsos en un margen de menos de 200ns, hacen que la familia Maestro sea adecuada para un alto rendimiento mecánico y electrónico, con un exacto control de aceleración y velocidad pueden hacer fácil la consecución de movimientos suaves, sin golpes, sin necesidad del control desde el código fuente para calcular constantes intermedias de actualización.

Los Maestro, con velocidades de pulso configurables (hasta de 333 Hz para Mini Maestro) pueden generar una amplia gama que alcance la máxima capacidad de respuesta para servos modernos. Las unidades se pueden conectar en cadena con otros controladores Pololu y en una sola línea.

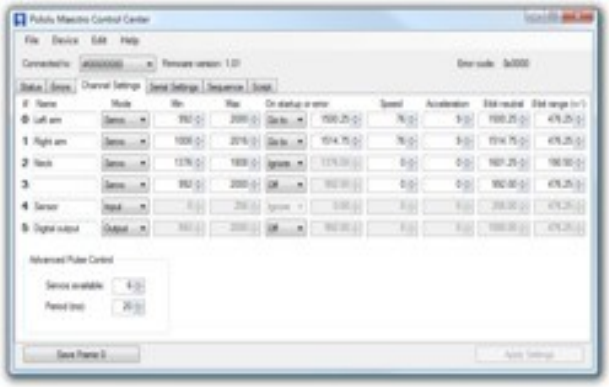


La libre configuración y su control es programable tanto en Windows como en Linux (ver sección 4), haciendo simples los ajustes y pruebas de la placa sobre USB, creado secuencias de movimientos para robótica escribiendo por pasos y corriendo los scripts almacenados en el servo controlador. La memoria interna de la placa permite el almacenamiento de las posiciones del servo que pueden iniciarse automáticamente sin necesidad de estar conectado al PC o a un micro externo. (Ver sección 6).

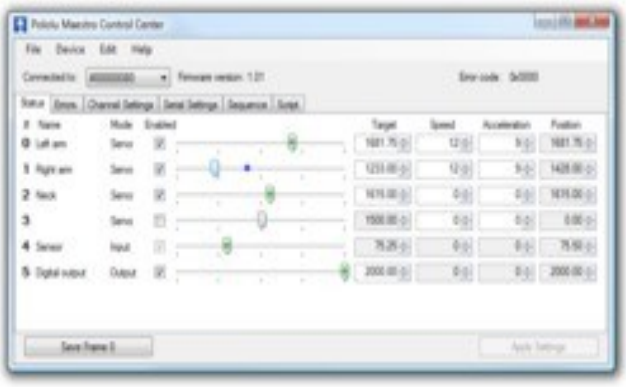
Los canales pueden usarse también como entradas analógicas-digitales o salidas digitales permitiendo de manera fácil la lectura de sensores y el control de periféricos directamente desde el PC a través del USB.

Estas entradas pueden usarse desde scripts o en forma autónoma para responder a estímulos externos. Un cable USB-A a mini-B (no incluido) se necesario para la conexión entre el dispositivo y el PC.

Características
Tres métodos de control: USB, TTL (5 V) serie y con scripts internos.
0.25µs de resolución en la salida de ancho de pulso (corresponde a unos 0.025° para un servo típico, que es más de lo que el servo puede resolver).
Configuración de pulso alto y ancho del pulso (ver la tabla comparativa).
Control de velocidad y aceleración individual para cada canal.
Los canales pueden configurarse de manera opcional para ir a una posición específica, pararse o iniciarse o dar error.
Funciones alternas del canal para usarse como:
Salidas digitales de propósito general (0 o 5 V).
Entradas analógico/digitales (canales 0 – 11 pueden ser analógicas; canales 12+ pueden ser entradas digitales).
Un canal puede sacar PWM con una frecuencia de entre 2.93 KHz a 12 MHz y hasta 10 bits de resolución (ver sección 4.a para detalles).
Un lenguaje simple de scripts permite programar el controlador para realizar complejas acciones después de haber desconectado el USB y desactivar la comunicación serie).



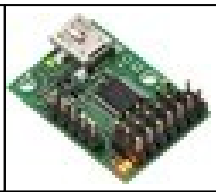
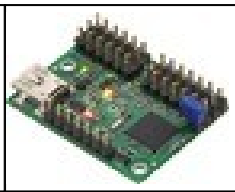
Pestaña Channel Settings en Maestro Control Center.



Pestaña Status en Maestro Control Center.

La libertad de configuración y su aplicación de control en Windows o en Linux, hacen fácil:

- Configurar y probar el controlador.
- Crear, correr y guardar secuencias de movimientos para servos o robots andantes.
- Escribir y correr scripts paso a paso, almacenados en el servo controlador
- Dos maneras de escribir software para Maestro desde el PC:
  - Puerto Virtual COM que hace fácil el envío de comandos desde cualquier programa que soporte comunicación serie.
  - Pololu USB Kit de desarrollo de software que te permite usar comandos avanzados USB e incluye ejemplos de código en C#, Visual Basic .NET y Visual C++
- TTL serie características:
  - Soporta 300 – 200,000 bps en modo fijo y 300 – 115,200 bps en modo auto detección.
  - Simultáneamente soporta protocolo Pololu, que da acceso a funciones avanzadas y protocolo simple de Scott Edwards MiniSSC II (que no precisa ninguna configuración del dispositivo para este protocolo particular)
  - Puede encadenarse con otros controladores Pololu para servos y motores usando una simple línea de transmisión serie.
  - La entrada encadenada permite la recepción de datos de múltiples Mini Maestros desde una sola línea de recepción, sin modulos adicionales (no en los Micro Maestros).
  - Can es una función general para USB-a-TTL adaptada para proyectos controlados desde PC.
- La placa se alimenta desde USB o con batería de 5-16V, y dispone de 5V regulados para el usuario
- Firmware actualizable

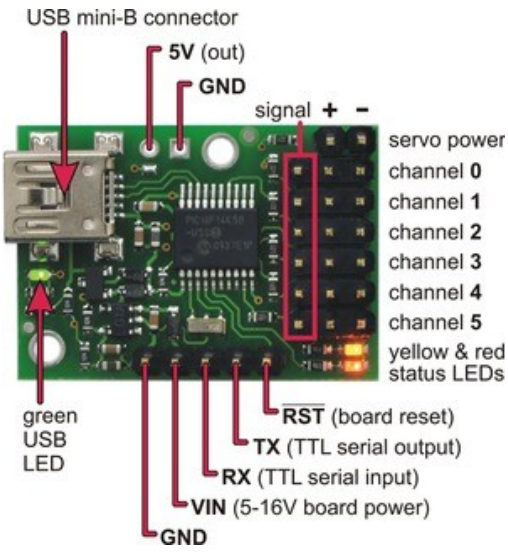
Tabla comparativa de la familia Maestro				
	Micro Maestro	Mini Maestro 12	Mini Maestro 18	Mini Maestro 24
Canales:	6	12	18	24
Entradas analógicas	6	12	12	12
Entradas digitales	0	0	6	12
Ancho:	0.85" (2.16 cm)	1.10" (2.79 cm)	1.10" (2.79 cm)	1.10" (2.79 cm)
Largo:	1.20" (3.05 cm)	1.42" (3.61 cm)	1.80" (4.57 cm)	2.30" (5.84 cm)
Peso[(1)]:	3.0 g	4.2 g	4.9 g	6.0 g
Pulso alto[(2)]:	33–100 Hz	1–333 Hz	1–333 Hz	1–333 Hz
Pulso rango[(2)]:	64–3280 µs	64–4080 µs	64–4080 µs	64–4080 µs
Script tamaño[(3)]:	1 KB	8 KB	8 KB	8 KB

1.- Peso de la placa sin pins.
2.- El pulso alto y el ancho de pulso disponible depende entre otros factores de los baudios y del número de canales usados (ver sección 9 para detalles).
3.- El sistema de scripts de usuario es más potente en el Mini Maestro que en el Micro Maestro (ver sección 6.d para detalles).

Ejemplos de aplicación

Controlador multiservo serie (Ej.. armas robóticas, mecatrónica, iluminación con leds) basados en placas micro controladas como BASIC Stamp, Orangután robot, o plataformas Arduino.
Servo-control desde puerto USB del PC
Interfaz entre sensores y otra electrónica:
Lectura de giro o acelerómetros desde PC para nuevas presentaciones
Control de cadena de ShiftBrites (leds) desde PC para iluminación
Expansión de I/O en el área proyectos de micros
Programación de efectos luminosos que respondan a diferentes sensores
Test de servos

1.a. Micro Maestro pins y componentes



Nota: Esta sección es aplicable para el controlador Micro Maestro.
Ver sección 1.b para más información del Mini Maestro.
El controlador Pololu Micro Maestro 6-canales puede conectarse a un PC vía puerto USB con un cables USB A a mini-B (no incluido). La conexión USB se usa para configurar el controlador y también para enviar comandos al controlador y recoger información del estado del mismo, enviando y recibiendo vía TTL bytes de datos por las líneas RX y TX.
El procesador y los servos pueden tener alimentación separada.
La alimentación del procesador se puede tomar desde la conexión USB o externamente con valores de 5 a 16V conectados a las entradas VIN y GND.
Se puede tener una fuente de alimentación externa conectada al mismo tiempo que el USB, en cuyo caso el procesador se alimenta de la fuente de alimentación externa. Tenga en cuenta que si la fuente de alimentación externa cae por debajo de 5V, no está garantizado el funcionamiento correcto incluso estando conectado desde el USB.

La energía para el funcionamiento de los servos se realiza desde la esquina superior derecha de la placa Micro Maestro. La alimentación del servo se recibe directamente sin tener que pasar a través de un regulador por lo que las únicas restricciones para la fuente de alimentación del servo son que

debe estar dentro del rango de funcionamiento del mismo y proporcionar suficiente corriente para su aplicación. Por favor, consulta las fichas técnicas para determinar la alimentación adecuada, tenga en cuenta que una cifra aproximada de consumo de corriente da un promedio de 1A.

Micro Maestro configurado para usar una alimentación común placa y servos.

Puedes alimentar los servos y el procesador desde una sola fuente de alimentación mediante la conexión de PWR a VIN soldando un cable en la parte inferior de la placa como se muestra en la imagen. La conexión a tierra no es necesaria porque todos los pins del tablero ya están conectados.

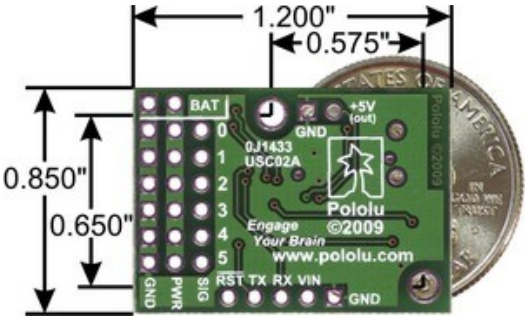
La salida de 5V (disponible) te permite alimentar dispositivos a 5V desde la placa hasta 50mA sin regulador o desde USB. El regulador de la placa funciona siempre que VIN este alimentado; en este caso el Maestro requiere 30 mA, por lo que quedan disponibles 20 mA para otros dispositivos.

Las líneas **SIG** (**0**, **1**, **2**, …) se usan para enviar las señales de pulsos a los servos, control de las salidas digitales y medición de voltajes analógicos. Estas líneas van protegidas por resistencias de 220Ω. El limite total de corriente (entrada o salida) para estos pins es de 60 mA, pero si usamos el regulador de voltaje de la placa la salidas quedan limitadas a 20 mA.

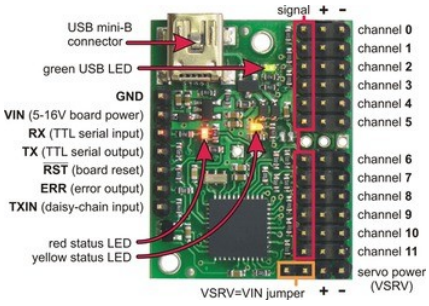
La línea **RX** se usa como receptora no invertida TTL (0–5 V) serie, como en las UARTS de los microcontroladores. Estos bytes pueden ser comandos, bytes transmitidos entre PC vía USB o ambos. Para más información mira la sección 5.a. Puede que el Maestro tenga posibilidad de recibir bytes a 3.3V, pero no garantizamos que lea 3.3V como alto en el pin RX a no ser que los fuerces a 4V si quieres asegurarte la operación.

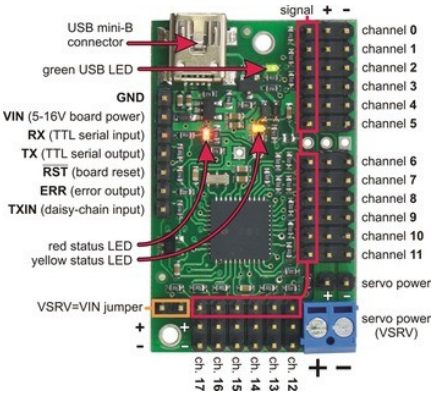
La línea **TX** transmite bytes serie no invertidos (0–5 V). Estos bytes pueden ser respuestas a los comandos enviados o bytes enviados desde el PC a través de la conexión USB.

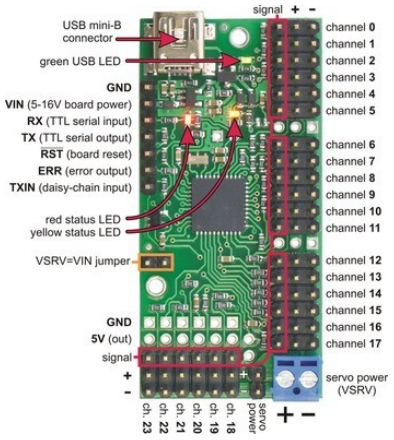
El pin RST debe estar bajo para que resetee el micro del Maestro, pero no es necesario en aplicaciones típicas. La línea, internamente está en alto con resistencia pull-up, por lo que puede dejarse desconectada. En la imagen puedes ver las medidas de la placa controladora de servos.



1.b. Mini Maestro pins y componentes

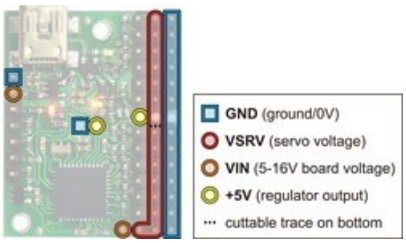


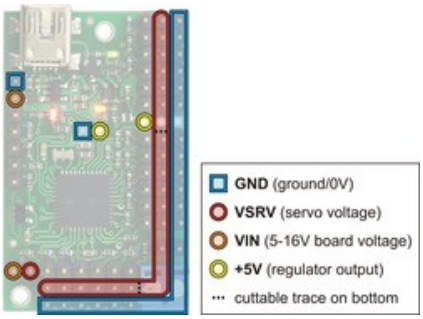


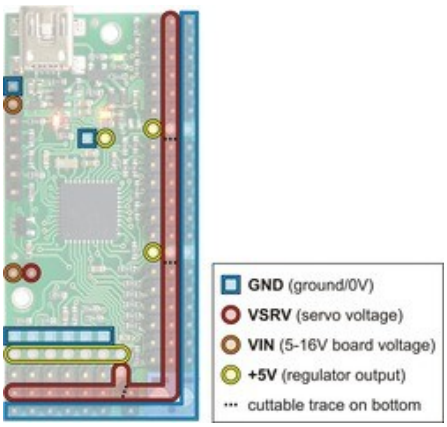


Mini Maestro 12-canales USB Mini Maestro 18-canales USB Mini Maestro 24-canales USB

Nota: Esta sección solo se aplica a los controladores Mini Maestro de 12, 18 y 24. Ver sección 1.a para Micro Maestro.
Las Pololu Mini Maestro de 12,18 y 24 canales pueden conectarse al PC vía puerto USB con un cable USB A a mini-B (no incluido). La conexión USB se usa para la configurar la controladora y poder enviar comandos y recoger información acerca del estado de la misma mediante el envío y recepción de bytes a través de las líneas RX y TX.
El procesador y los servos pueden tener alimentación separada.
La alimentación del procesador puede hacerse desde USB o desde una fuente externa con voltaje de 5–16V conectada a las patillas **VIN** y **GND** en el borde izquierdo de la placa. Se puede tener una fuente de alimentación externa conectada al mismo tiempo que USB, en cuyo caso el procesador se alimenta de la fuente externa. Tenga en cuenta que si la fuente de alimentación externa cae por debajo de 5 V, no está garantizado el correcto funcionamiento , incluso estando conectado al USB.
Las conexiones de alimentación de los servos se encuentran en la parte inferior derecha de las placas Mini Maestro. En las Mini Maestro 18 y 24, puedes hacer la alimentación de los servos a través del bloque azul de dos vías o del conector de dos pins; en el Mini Maestro 12 solo por el conector de 2-pin 0.1" destinado a la alimentación de servos. La alimentación del servo pasa directamente sin tener que regularse por lo que las únicas restricciones a la fuente de alimentación del servo son que debe estar dentro del rango de funcionamiento y proporcionar suficiente corriente para su aplicación. Por favor, consulta las fichas técnicas de los servos para determinar la fuente adecuada y ten en cuenta que el consumo de corriente de un servo es de 1 A.
Puedes alimentar los servos del Maestro desde el procesador y con una sola fuente realizando la conexión del positivo y el VIN de los puertos de alimentación de los servos (la conexión a tierra no es necesaria porque todas los pins de la placa están conectados). Recomendamos que para hacer esto al conectar la fuente de alimentación a los pins de alimentación de la esquina, utilice el bloque azul incluyendo un jumper para conectar los pins de "VSRV = VIN".
La salida de 5V (disponible**)** puede dar corriente a dispositivos con 5V desde el regulador en la placa de 150mA o directamente desde USB. El regulador se utiliza cada vez que VIN se alimenta, el Maestro requiere 50 mA, luego quedan 50 mA disponibles para alimentar otros dispositivos.
Las líneas de señal (**0**, **1**, **2**, …) se utilizan para los pulsos a los servos, el control de salidas digitales o la medida de voltajes. El limite total para estos pins es de 150 mA, pero si usamos el regulador se queda en 50 mA (ver detalle.)







Mini Maestro 12 pins alimentación. *Mini Maestro 18 pins alimentación.* *Mini Maestro 24 pins alimentación.*

La línea **RX** se usa como receptora no invertida TTL (0–5 V) serie, como en las UARTS de los microcontroladores. Estos bytes pueden ser comandos, bytes transmitidos entre PC vía conexión USB o ambos. Para más información mira la sección 5.a. Puede que el Maestro tenga posibilidad de recibir bytes a 3.3V, pero no garantizamos que lea 3.3V como alto en el pin RX a no ser que fuerces a 4V si quieres asegurarte la operación.
La línea **TX** transmite bytes serie no invertidos (0–5 V). Estos bytes pueden ser respuestas a los comandos enviados o bytes enviados desde el PC a través de la conexión USB, o también desde la línea TXIN.

El pin RST debe estar bajo para que resetee el micro del Maestro, pero no es necesario para aplicaciones típicas. Internamente la línea está en alto con resistencia pull-up, por lo que puede ser desconectada.
La línea **ERR** es una salida que control del led rojo para cuando hay errores. Se pone en alto cuando el led rojo está encendido y en bajo cuando está apagado. El led rojo se enciende cuando ocurre algún error y se apaga en cuanto el bit de error se pone a cero. Puede controlarse a través de un script de usuario. Desde la línea de ERR en bajo se puede conectar con la línea de ERR de múltiples Mini Maestros juntos. Tenga en cuenta, sin embargo, que hacer esto hará que el LED rojo de todos los Mini Maestros conectados a su vez se enciendan en cuando uno de los Maestros Mini tenga el suyo. Para más información sobre las condiciones de error posible y las opciones de respuesta, por favor, ver Sección 4.b.
La línea TXIN es una línea de entrada que hace que sea fácil encadenar varios Mini Maestros.
Cualquier serie de bytes recibidos por esta línea se van a almacenar en el buffer de transmisión y a ser transmitidos por la línea TX. Mira la Sección 5, letra g para tener más información sobre la conexión encadenada.
Las dimensiones en vertical y horizontal en distancias entre los agujeros son: 1.20" y 0.50" para el Mini Maestro 12, 1.58" y 0.50" para el Mini Maestro 18, y 1.50" y 0.50" para el Mini Maestro 24.

1.c. Indicadores LEDs

El Maestro tiene tres indicadores LEDs:
El `led verde` indica el estado del dispositivo USB. Cuando no está conectado a USB estará apagado. Al conectarlo parpadeará lentamente. El parpadeo continua mientras reciba un mensaje particular desde el PC indicando al Maestro que los controladores USB están correctamente instalados. Después de leer el mensaje el led verde permanece encendido. Pero parpadea si hay actividad en la línea USB. La aplicación de control envía constantemente datos cuando está conectado al Maestro por lo que habrá actividad en la línea.
El `led rojo` de error indica que hay errores en la transmisión. Se encenderá cuando ocurra un error y se apagará al limpiar el bit de error. Mira la sección 4.b para información de errores. El led rojo también puede ser controlado por un script de usuario, el LED rojo se enciende si hay algún error o si el comando script para encenderlo se ha ejecutado.
El `led amarillo` indica el control del estado de la trasmisión. Cuando el Maestro está en modo auto-baudio (por defecto) y aún no se ha detectado la velocidad, el LED amarillo parpadeará lentamente. Durante este tiempo el Maestro no transmite pulsos a los servos. Una vez que el Maestro está listo para controlar los servos, el LED amarillo se enciende brevemente de forma periódica. La frecuencia de los destellos es proporcional al período de servo (la cantidad de tiempo entre los pulsos en un solo canal), con un período de 20 ms el parpadeo se produce aproximadamente una vez por segundo. En Micro Maestro el número de parpadeos indica el estado: un destello indica que ningún servo está habilitado (no se envían pulsos) y todos los canales de salida están bajos, mientras que un doble flash indica que al menos uno de los servos está habilitado o uno de los canales de salida está siendo impulsado en alto. Los Mini Maestros sólo emiten un destello. Además, cuando un comando serie válido es recibido, el LED amarillo emite un breve destello, tenue, que termina cuando el siguiente comando serie válido se recibe, o cuando el parpadeo principal se produzca (lo que ocurra primero). Cuando se resetea por alguna razón que no sea la alimentación, el led rojo y/o el amarillo parpadean cuatro veces para indicar la condición de reset.
Amarillo apagado, **rojo** parpadeando: Un reset de alimentación. Ocurre cuando los 5V de corriente del Maestro descienden por debajo de los 3.0 V, seguramente por la descarga de las baterías o por una alimentación inadecuada..
Amarillo pardeando, **rojo** apagado: Reset del Maestro por bajo voltaje en la línea RST. Un reset.
Amarillo y **rojo** parpadean a la vez: Un fallo de firmware hace que el "watchdog" salte. Esto también ocurre inmediatamente después de una actualización de firmware, como parte normal del proceso de actualización.

Amarillo parpadeando, **rojo** fijo: Un error de firmware provoca un reset de software. Esto nunca debería ocurrir durante el uso normal.

1.d. Sistemas operativos soportados

Los controladores Maestro y su configuración de software USB trabajan igual bajo Microsoft Windows XP, Windows Vista, Windows 7 y Linux. No es compatible con Mac Os.

2. Contactando con Pololu

Puedes consultar la página del producto de tu modelo Maestro para obtener información adicional. Nos gustaría conocer tu opinión sobre alguno de tus proyectos y sobre tu experiencia con Maestro. Puedes contactar con nosotros directamente o enviando mensaje al foro.
Cuéntanos lo que hicimos bien, lo que podría mejorar, lo que te gustaría ver en el futuro, o cualquier otra cosa que quieras decir!

3. Primeros pasos

3.a. Instalación de los drivers y software en Windows

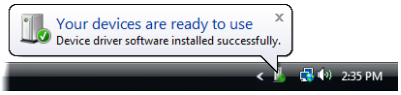
Si estás usando Windows XP necesitas tener instalado el Service Pack 3 antes usar los drivers para Maestro. Sigue los detalles. Antes de conectar Maestro al PC debes instalar los drivers para Microsoft Windows de esta manera:

1.- Descarga Maestro Servo Controller Windows Drivers y Software (6MB zip)

2.- Abre el fichero ZIP y corre `setup.exe`. El instalador te guiará a través de pasos en la instalación de del Maestro Control Center, la utilidad de Maestro Command-line (UscCmd) y los drivers para el PC. Si el instalador falla debes de extraer los ficheros a un directorio temporal y luego darle al `setup.exe` como “administrador”.

3.- Durante la instalación Windows te mostrará una pantalla de alarma diciendo que los drivers no están testados por Microsoft y no son recomendables. Ni caso, clic en “Continue” (Windows XP) o “Instalar el driver siempre” (Windows 7 y Vista).

4.- Una vez finalizada la instalación inicia el menú desde el enlace de Maestro Control Center (en el directorio de Pololu). Esta es una aplicación windows que permite configurar, controlar y depurar en tiempo real con Maestro. Tiene una utilidad de Comando-línea llamada UscCmd que permite comandos rápidos.

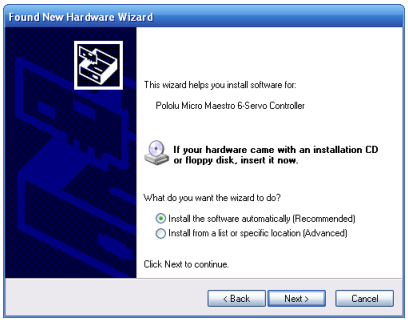


Usuarios de Windows 7 y Windows Vista: El PC debe iniciar automáticamente e instalar los drivers necesarios al conectarse con Maestro. No se requiere ninguna acción.

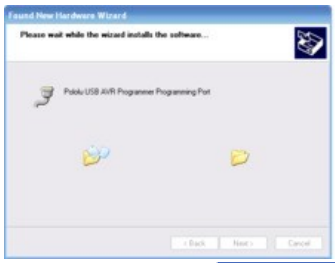
Usuarios de Windows XP: Sigue los pasos 5–9 para cada nuevo Maestro que conectes al PC.

5.- Conecta el dispositivo al USB del PC. Maestro muestra tres dispositivos en el PC con XP y detecta tres nuevos dispositivos mostrando “Nuevo hardware encontrado” hasta tres veces, recuérdalo. Cada vez que salga este mensaje sigue los pasos 6-9.

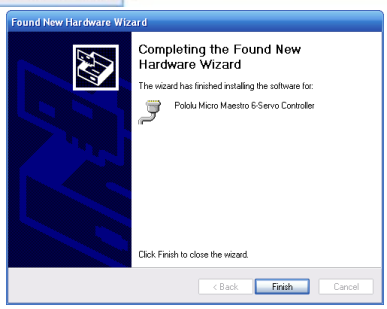
6.- Cuando se muestra “Nuevo hardware encontrado” la primera vez selecciona “No, por esta vez” y clic “Siguiente”.
7.- En la segunda pantalla de “Nuevo hardware encontrado” haz “Instala el software automáticamente” y clic “Siguiente”.



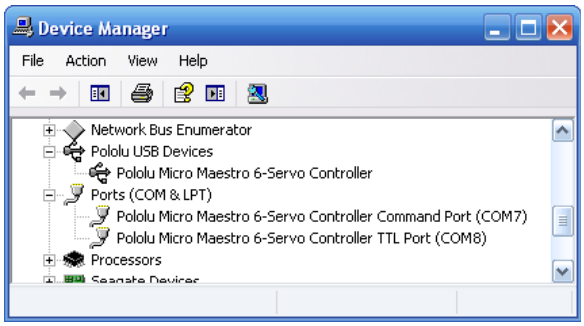
8.- Windows XP volverá a mostrar la pantalla de software no testado pero sigue con clic “Continuar la instalación”.
9.- Cuando hayas finalizado clic en “Terminar”. Después aparecerá otra pantalla y estarás en la tercera cuando insertes el Maestro. Sigue los pasos 6-9
Si tienes problemas con Windows XP con la instalación o usando los puertos la causa puede estar en algún fallo o en la antigüedad del controlador de Microsoft’s usb-to-serial driver `usbser.sys`. Los ficheros anteriores a la versión 5.1.2600.2930 no funcionan con Maestro.



Puedes comprobar la versión del driver pulsando en “Detalles” de la ventana “Propiedades” del fichero `usbser.sys` en la carpeta `C:\Windows\System32 \drivers`. Para encontrar una versión actualizada deberías instalar el Service Pack 3. Si no lo tienes inténtalo con el Hotfix KB918365, pero algunos usuarios han detectado problemas con el hotfix que solo se han resuelto instalando el Service Pack 3.

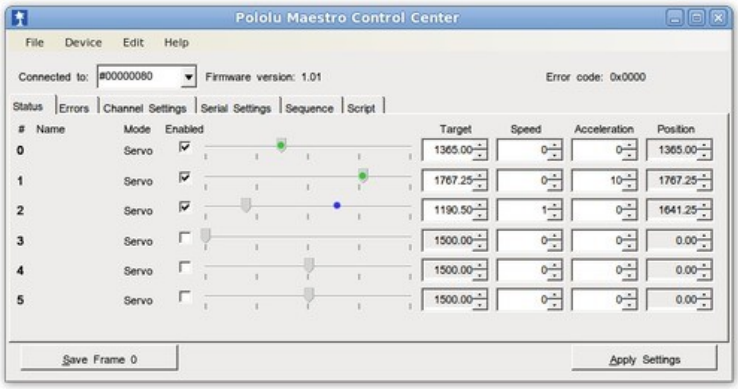


El software de configuración y control funcionará incluso si los controladores de puerto de serie no están instalados correctamente
Después de instalar los drivers, vete a Control de dispositivos y expande la lista de “Puertos (COM & LPT)” deberías encontrar dos puertos COM: el Comando Port y el TTL Port. En paréntesis después de los nombres veras el nombre del puerto (Ej. “COM5” o “COM6”). Si expandes “Pololu USB Devices” verás las entradas de Maestro. Hay software que no permite el acceso a puertos COM con numeración alta: Puedes reasignar el número de puerto desde Control de Dispositivos en el cuadro Propiedades y en botón Avanzado. Desde allí podrás realizar el cambio de numeración del puerto.



3.b. Instalación de drivers y software en Linux

Debes descargar el fichero Maestro Servo Controller Linux Software (112k gz)
Desempaqueta el archivo tar/gz haciendo “`tar -xzvf`” seguido del nombre del fichero.
Después sigue las instrucciones que hay en `README.txt` y ya podrás ejecutar los programas `MaestroControlCenter` y `UscCmd`.



También puedes descargarte el código fuente en C# para `UscCmd` que forma parte del Pololu USB Software Development Kit. Lee `README.txt` en el SDK para más información.
Los dos puertos serie de Maestro pueden usarse en Linux sin necesidad de instalar drivers. Los puertos son manejados por el modulo kernel *cdc-acm*, cuyo código fuente puede encontrarse en el kernel’s source code `drivers/usb/class/cdc-acm.c`. Cuando conectas Maestro

al PC los dos puertos virtuales aparecen y los dispositivos se renombran como `/dev/ttyACM0` y `/dev/ttyACM1` (el número dependerá de cuantos dispositivos ACM estén conectados). El puerto con número menor es el Command Port, el de mayor numeración será el TTL serie. Puedes usar un Terminal (como kermit) para enviar y recibir bytes por estos puertos.

3.c. Uso del Maestro sin USB

Puedes utilizar Maestro como controlador de servos serie sin necesidad de instalar ningún driver USB, mediante un PC. Si no usas USB no podrás cambiar las configuraciones en Maestro, pero puedes utilizar la configuración por defecto que es adecuada para muchas aplicaciones. La configuración por defecto se describe a continuación..

Ajustes por defecto

Modo serie "`UART, detect baud rate`"; después de enviar `0xAA` como byte de comprobación de baudios, Maestro ya puede aceptar comandos a través de la línea serie RX en TTL.

El número de dispositivo con el protocolo Pololu es 12, en el Mini SSC offset es 0 y el timeout y el CRC están deshabilitados.

Todos los canales están configurados para trabajar con servos con un pulso mínimo de 992 µs y máximo de 2000 µs.

El punto neutral en 8 bits es 1500 µs y el rango es de 476.25 µs.

Al inicio o en estado de error, los servos se paran (no reciben pulsos).

Al arrancar, no hay limitación de velocidad y aceleración pero se puede ajustar usando comandos.

El periodo del servo esta en 20 ms (cada servo recibe pulsos cada 20 ms).

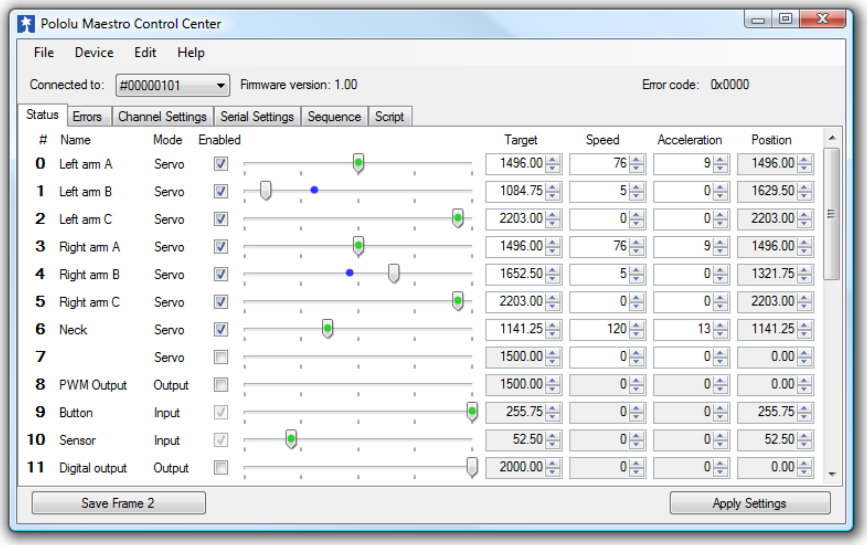
El script de usuario está vacío.

4. Uso del Maestro Control Center

La aplicación para Maestro's USB da acceso a todas las opciones de configuración soportando control en tiempo real, reacción y depuración. El Maestro Control Center es una herramienta gráfica que facilita el trabajo a través de USB; para muchos proyectos puedes usar la aplicación para ajuste y verificación del Maestro. Esta sección explica muchas de las características de Maestro y del Maestro Control Center.

4.a. Estado y control en tiempo real

La pestaña **`Status`** se utiliza para controlar las salidas de Maestro y el seguimiento de su estado en tiempo real. Hay una fila separada con los controles para cada uno de los canales de Maestro. En la imagen vemos los 12 canales que aparecen en Maestro Control Center y que está conectado a un Mini Maestro 12. Cuando se conectan otros modelos de Maestro el número de canales disponibles se mostrarán en pantalla.



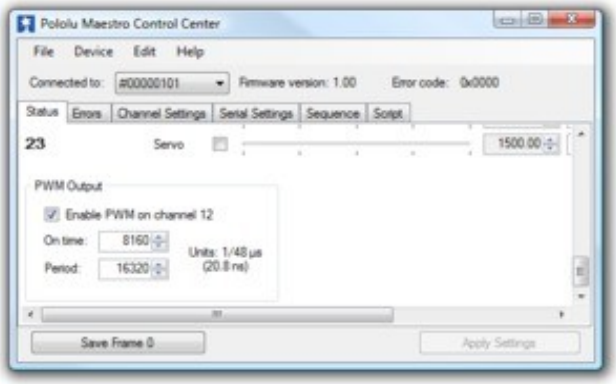
Según esté la casilla del canal configurada como servo o como salida, la barra de control deslizante te servirá para ajustar el objetivo deseado y la bola verde indica la posición actual del canal. Por ejemplo, si el canal está ajustado a una velocidad relativamente baja, cuando muevas el control a una nueva posición, la bola verde se moverá lentamente hasta que haya alcanzado al control deslizante, lo que nos hace intuir como se alcanzará el objetivo. Para un control más preciso, también se puede

introducir un valor objetivo directamente en la casilla `"Target (destino)"`. El control deslizante automáticamente se escala para que coincida con los valores mínimo y máximo que figuran en la pestaña de configuración.

Para un canal configurado como entrada, el control deslizante de bolas verdes, "destino" y "posición" muestra el valor actual de la entrada. No hay control para las entradas. Las entradas en los canales 0-11 son analógicas: sus valores van desde 0 hasta 255.75, representando las tensiones de 0 hasta 5 V. Las entradas en los canales 12-23 son digitales: sus valores son exactamente el 0 o el valor 255,75.

La entradas "velocidad" y "aceleración" permiten ajustar los canales de servo de forma individual y en tiempo real. Los valores por defecto se especifican en la ficha configuración, pero puede ser útil ajustarlos aquí para afinar. Todos los controles en esta pestaña siempre muestran los valores actuales según Maestro, por lo que son útiles para realizar el seguimiento de acciones causadas por otros programa o dispositivos. Por ejemplo, si un microcontrolador utiliza la interfaz serie TTL para cambiar la velocidad del canal para servos del 2 al 10, este valor se mostrará inmediatamente en la entradas correspondientes, aunque allí no se haya introducido nada antes.

`PWM Output (Mini Maestro 12, 18, y 24 sólo)`



En los Mini Maestro 12, 18 y 24 un canal puede usarse como salida PWM de propósito general con frecuencias de entre 2.93 KHz hasta 12 MHz y con 10 bits de resolución. Esta ventaja puede utilizarse por ejemplo como entrada de control de motores o como control de brillo para leds. La salida PWM está en el canal 8 del Mini Maestro 12 y en el canal 12 para los Mini Maestro 18 y 24. Este canal debe configurarse como salida PWM si la necesitamos en el proyecto.

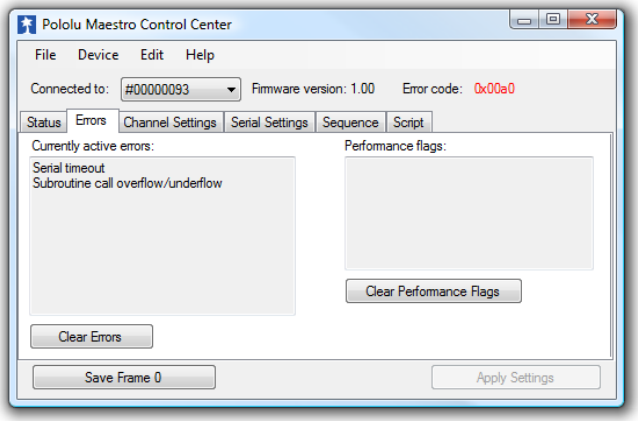
Puedes usar el control de salida PWM en la pestaña de estado para testear el PWM chequeando la casilla y entrando los valores específicos de tiempo y periodo en unidades de 1/48 µs. Un periodo de 4800, por ejemplo, generará una frecuencia de 10 KHz. La resolución de estos valores depende del periodo mostrado en la tabla siguiente:

Periodo rango	Periodo resolución	On-time resolución
1–1024	4	1
1025–4096	16	4
4097–16384	64	16

Los periodos extremos 1024, 4096 y 16384 no se recomiendan ya que el 100% duty cycle no está disponible para estos valores. Si el on time se ajusta igual al periodo en unos de esos valores el duty cycle queda en 0% (baja salida). Los periodos 1020 (47.1 KHz), 4080 (11.7 KHz) y 16320 (2.9 KHz) producen la mejor resolución posible al 100% y 0% de duty cycle, así que usa estos periodos a ser posible.

Es posible que tengas que usar comandos serie o scripts para hacer el PWM en tu proyecto. Mira la sección 6.b sección 5.e para más información

4.b. Errores



La pestaña **`Errors`** sirve para mostrar problemas que Maestro detecta durante su funcionamiento y que pueden estar provocados por fallos de comunicación o por la mala definición del script.

Cada error se corresponde con un bit del registro de dos bytes de errores. El led rojo se encenderá siempre y cuando alguno de los bits del registro de error se ponga a 1 (también puede activarse con el comando `led_on`). El valor `Error code` se muestra en la esquina superior derecha de la ventana principal.

Cuando ocurre un error, el bit correspondiente en el registro de error se pone a 1 y el Maestro envía los servos y las salidas digitales a sus posiciones iniciales, tal como se haya especificado en la ficha configuración (sección 4.e). Cualquier servo o salida configurada con la opción "`Ignorar`" no cambiará el modo. El registro de error se elimina con el comando "`GetError`".

Los errores y sus correspondientes bits son los siguientes:

Serial Signal Error (bit 0)

Error de hardware que ocurre cuando el stop bit no se ha detectado en el momento oportuno. Eso ocurre cuando hay diferencia entre los baudios de transmisión y los configurados.

Serial Overrun Error (bit 1)

Error de hardware que ocurre cuando el buffer interno de la UART está lleno. Esto no debe ocurrir en operaciones normales.

Serial RX buffer lleno (bit 2)

Error de firmware que se produce cuando el buffer de bytes recibidos por la línea RX esta lleno y como consecuencia se ha perdido un byte.

Este error no debe ocurrir durante la operación normal.

Serial CRC error (bit 3)

Este error se produce cuando el Maestro trabaja con el CRC activado y al comprobar la redundancia cíclica al final del paquete de comandos no coincide con lo que el Maestro ha calculado para ese paquete (sección II). En tal caso, el Maestro ignora el paquete de comandos y genera un error CRC

Serial protocolo error (bit 4)

Este error se produce cuando el Maestro recibe en un formato incorrecto o hay paquetes absurdos. Por ejemplo, si el byte no coincide con un comando conocido o un paquete de comandos es recortado o interrumpido por otro paquete, se produce este error.

Serial timeout error (bit 5)

Cuando está habilitado timeout se producirá error siempre y cuando haya transcurrido el período de tiempo de espera sin que Maestro haya recibido los comandos válidos.

Este error de tiempo de espera puede usarse para hacer que los servos vuelvan a sus posiciones iniciales en el momento en que se interrumpan las comunicaciones.

Script stack error (bit 6)

Error que se produce cuando la secuencia de comandos causa un desbordamiento de pila. Cualquier secuencia de comando que modifique la pila tiene potencial para causar este error.

La profundidad de pila es de 32 niveles en el Maestro Micro y de 126 en los Maestros Mini.

Script call stack error (bit 7)

Error que se produce cuando la secuencia ha causado desbordamiento en la pila de llamadas. El desbordamiento puede ocurrir por haber demasiados niveles de subrutinas anidados o por una subrutina llamadas a sí misma demasiadas veces.

La profundidad de la pila de llamadas es de 10 en el Maestro Micro y de 126 de los Maestros Mini.

También de produce un desbordamiento cuando hay un retorno sin llamada en la subrutina correspondiente. Si ejecutas una subrutina mediante el comando "`Restart Script at Subroutine`" y termina con un `RETURN` en lugar de un `QUIT` o bucle infinito, dará este error.

Script program counter error (bit 8)

Este error se produce cuando la secuencia de comandos provoca que el contador de programa (la dirección de la siguiente instrucción a ejecutar) esta fuera de límites. Esto sucede si tu programa no termina en quit, return o bucle infinito.

Performance Flags

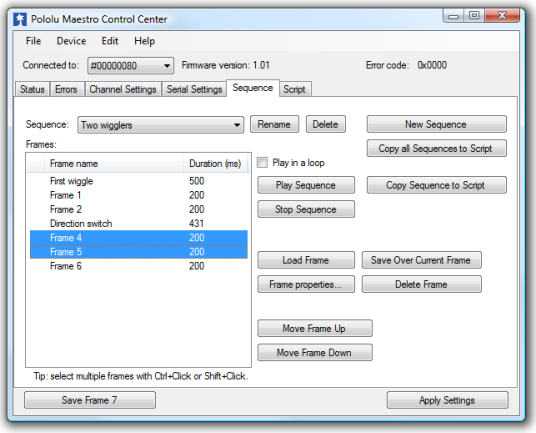
La pestaña de errores muestra también los perfomance flags establecidos. Esta característica sólo se aplica a Mini Maestro 12, 18 y 24.

Los perfomance flags indican que el procesador ha perdido el tiempo límite para realizar una tarea de servo control y como resultado el control del Maestro consiguió ralentizarlo de alguna manera.

Los perfomance flags no deberían ocurrir durante las operación normales, ya que la configuración está dentro de los limites descritos en la sección 9.

4.c. Secuencias

La pestaña **Sequence** permite realizar secuencias de movimiento simple para ser reproducidas por Maestro. Una secuencia es simplemente una lista de "fotogramas" que especifican las posiciones de los servos y la duración (en milisegundos) para cada uno de ellos. Las secuencias se almacenan en el ordenador desde donde se creó la secuencia. Las secuencias pueden copiarse como secuencias de comandos y entonces sí se guardarán en memoria del Maestro. Las secuencias también pueden exportarse a otro equipo creando y guardando un archivo de configuración.



Para empezar a crear una secuencia, clic en `New Sequence` y dale un nombre a la misma. Con los controles de la pestaña estado, establece en cada uno de los servos la posición para el primer cuadro, luego clic en `Save Frame` en la parte inferior izquierda de la ventana. Repite este procedimiento para crear varios cuadros y a continuación volver a la pestaña de secuencias. Puedes ahora reproducirlos mediante el botón `"Play Sequence"`, o puedes establecer las posiciones para los servos del cuadro específico haciendo clic en el `Load Frame`.

El marco `Frame properties` te permite establecer el nombre y la duración de un cuadro. El botón `Save Over Current Frame` sobrescribe los cuadros seleccionados con los valores actualizados desde Maestro.

Si la casilla de verificación `Play in a loop` está activada la reproducción de la secuencia se repite en bucle cuando llega al final. El cuadro de lista desplegable `Secuence` junto con `Rename`, `Delete` y `New Sequence` permiten crear y administrar varias secuencias.

Una secuencia puede utilizarse también para crear un script que se almacena en el Maestro.

Hay dos botones para copiar secuencias a scripts:

`Copy Sequence to Script` copia la secuencia de comandos a un script. En la mayoría de casos, desearás marcar también la opción `"Run script on startup"` para que el script se ejecute automáticamente al arrancar el Maestro y sin necesidad de conexión con el PC.

`Copy all Sequences to Script` es una opción avanzada que añade un juego de subrutinas al final del script actual. Cada subrutina representa una de las secuencias; llamar a la subrutina desde el script hará que la secuencia se reproduzca una sola vez. Estas subrutinas pueden utilizarse junto con el código particular del script; por ejemplo, podrías hacer una presentación por la que un botón conectado a las entradas de los canales reprodujera la secuencia

Edición de secuencias. Trucos

Puedes seleccionar varias secuencias pulsando Ctrl o Mayús al hacer clic en ellas. Esto agiliza el mover, eliminar, establecer la duración de, cortar, copiar o guardar varios fotogramas a la vez.

Puedes arrastrar cualquiera de las fichas de la ventana principal a sus propias ventanas. Si arrastra la pestaña de secuencia fuera de ventana veras la pestaña de estado y la de secuencia al mismo tiempo.

Puede cortar, copiar y pegar secuencias seleccionándolas mediante Editar o con los atajos de teclado estándar. Las secuencias serán almacenados en el Portapapeles como texto separadas por tabuladores, por lo que se pueden cortar, pegar en una hoja de cálculo, editar y copiar en otra lista siempre que quieras. (Esta característica no funciona en Linux).

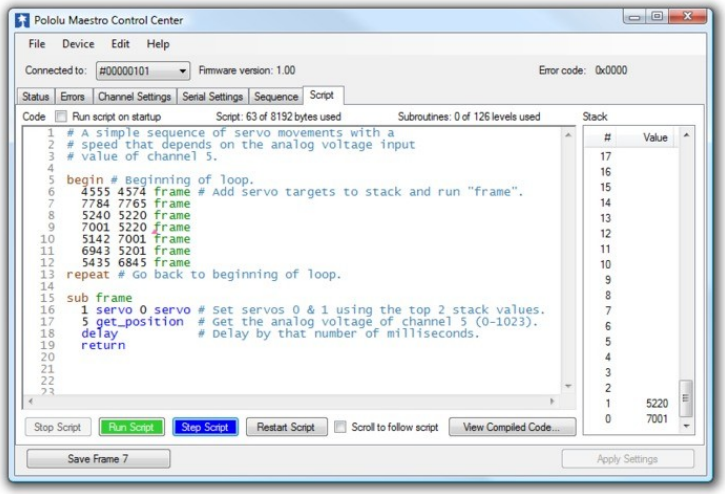
Atajos de teclado

Los siguientes atajos de teclado se pueden usar con las secuencias:

Ctrl+A: Seleccionar todas las secuencias.
Ctrl+C: Copiar.
Ctrl+V o **Shift+Insert**: Pegar/insertar.
Ctrl+X: cortar.
Ctrl+Up: Mover hacia arriba.
Ctrl+Down: Mover hacia abajo.
Del: Borrar.

4.d. Crear un script

La pestaña **`Script`** es el lugar donde escribir un script para ser cargado por Maestro. Para más información sobre el lenguaje de scripts de Maestro, ver la sección 6. Una vez haya introducido un script, clic en "`Apply Settings`" para cargar la secuencia en el dispositivo. Hay disponibles varias opciones para probar y depurar la secuencia de comandos en la pestaña de script.



Moverse a través de un script

Para empezar a ejecutar un script, haga clic en el botón verde "`Run Script`". Este será procesado por Maestro, una instrucción cada vez, hasta que se llegue a la instrucción QUIT o se produzca un error.

En muchos casos será útil utilizar un bucle de algún tipo para que el script no pare de ejecutarse. Mientras se ejecuta, esta disponible el botón rojo "Stop script" y vemos un pequeño triángulo rosa que se coloca junto al código fuente y muestra la instrucción que actualmente se está ejecutando. Si el script coloca datos en la pila, será visible en la parte derecha de la pestaña y las llamadas a rutinas anidadas se contarán en la barra de la parte superior de la ficha.

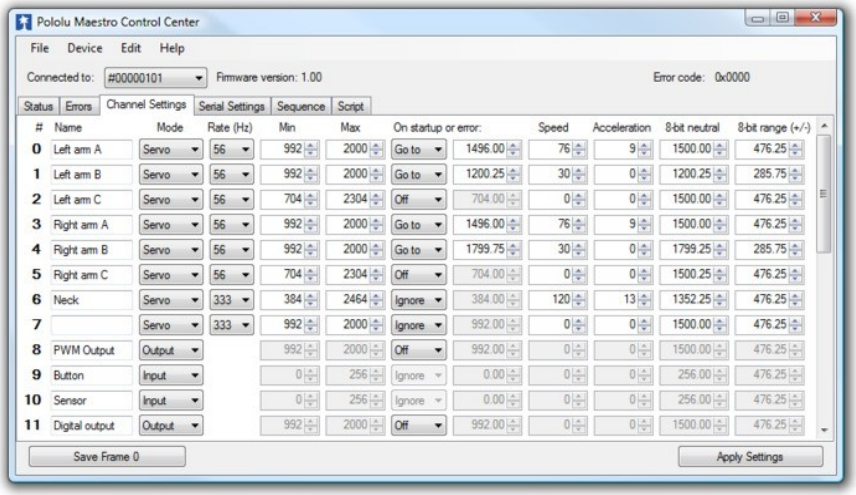
Para examinar el funcionamiento del script, clic en el botón azul "`Step Script`". Este botón hace que la secuencia de comandos se ejecute paso a paso. Esto nos servirá para recorrer el script y poder comprobar que cada parte programada hace exactamente lo esperamos que lo haga.

Ajustar un script para que corra desde el inicio de arranque

Por defecto el script solo corre si haces clic en el botón "`Run Script`". Sin embargo para algunas aplicaciones podría ser necesario que el script funcionara automáticamente sin necesidad de tener Maestro conectado de forma permanente a USB. Clic en la opción "`Run script on startup`" produce que Maestro corra el script desde el momento que recibe la alimentación. Aún puedes utilizar controles en la pestaña script para su depuración o para detener la ejecución del mismo.

Examinando el código compilado

Clic en el botón "`View Compiled Code`" para ver los bytes creados en cada línea del script. Esta herramienta es necesaria para los usuarios avanzados; si tienes interés en detalles de los códigos usados por Maestro (por ejemplo si quieres escribir tu compilador) escríbenos. Al final del código compilado se encuentra la lista de las subrutinas, incluyendo sus números de línea en decimal y hexadecimal. Estos números pueden usarse por comandos para introducir las subrutinas desde un control externo.



4.e. Ajuste de canales

La pestaña **`Channel Settings`** contiene controles para los canales utilizados cuyos parámetros son almacenados y afectan a las operaciones de forma inmediata.

Hay una serie de columnas que se muestran para cada uno de los canales:

Name. Cada canal puede tener un nombre según convenga. El nombre de los canales no se almacena en el dispositivo pero forma parte del registro de sistema en el ordenador. Si vas a utilizar Maestro en otro ordenador debes guardarlos en un fichero para su posterior utilización en el otro PC.

Mode. El modo de trabajo es el ajuste básico que determina su funcionamiento. Hay tres opciones:

- **Servo** (por defecto) indicado para salidas PWM para servos R/C.
- **Input** especifica que este canal se usara como entrada analógica o digital. Las entradas de los canales 0–11 son analógicas: están midiendo constantemente el valor de entre 0 y 1023 (VCC), en el rango máximo de 20 KHz. Las entradas para los canales 12–23 son digitales: sus valores son exactamente de 0 o 1023. Fíjate que los valores mostrados en Target y Position de la pestaña Estado son una cuarta parte del valor mostrado actual si hay 1023 se mostrara 255.75.
- **Output** especifica que canales se han designado como salidas digitales. En lugar de indicar el ancho de pulso, el valor de la posición del canal se utiliza para controlar si la salida es baja (0 V) o alta (VCC). Específicamente, la salida es baja a menos que el valor de posición sea mayor o igual a 1500.00 µs

Rate especifica el pulso para cada canal servo. En Micro Maestro 6, todos los servos deben tener guardado el pulso, mientras que en Mini Maestro 12, 18 y 24, pueden tener dos diferentes y escoger el que quieras para cada canal. Los pulsos disponibles son controlados por los ajustes en el **Period** y **Period multiplier** descritos más abajo.

Min y **Max** se refiere a los valores del canal en cuanto a *position*. Cada canal configurado como servo, los ajustes Min y Max corresponden respectivamente a los valores de ancho de pulso para cada servo en unidades de microsegundos.

On startup or error. Esta opción sirve para marcar el valor *destino* del canal que el dispositivo pondrá en marcha al arrancar (inicio o reset) o cuando se produzca un error. Fíjate que también hay velocidad y aceleración para cada canal, las salidas cambiarán suavemente a la posición especificada en caso de error, pero no al arrancar, ya que no tienen ninguna información acerca de la posición anterior del servo, en este caso.

- **Off** especifica que el servo esta inicialmente parado (valor 0) y que debe ser apagado en caso de producirse error.
- **Ignore** inicialmente apagado pero si hay error se queda en la posición que está..
- **Go to** especifica la posición por defecto a la que ir el servo. Será la posición inicial y a la que volverá en caso de error.

Speed. Sirve para marcar la velocidad del servo en unidades de 0.25µs/10ms. Así para una velocidad de 4, la posición cambiará a razón de 1µs por cada 10 ms, o 100.00 µs/s. **Mini Maestro 12, 18 y 24 solo:** Si estás usando un periodo por defecto de 20 ms, las unidades de velocidad son diferentes. Sigue la información más abajo.

Acceleration. Es la aceleración en unidades de 0.25µs/10 ms/80ms. Una valor de 4 hará que el servo vaya cambiando como máximo 1250 µs/s por segundo. **Mini Maestro 12, 18, y 24 solo:** si estas usando un periodo por defecto de 20 ms, las unidades de aceleración son diferentes. Sigue la información más abajo.

8-bit neutral. Se usa para especificar los valores destino en microsegundos, que se corresponde con el valor de 127 (neutral) en los comandos de 8 bits.

8-bit rango. Esta opción muestra el rango de valores de destino accesibles con comandos de 8 bits. Una valor de 8 bits igual a 0 corresponde a *neutral – range*, cuando el valor es de 254 resultara un valor destino de *neutral + range*.

Controles avanzados de pulso:

Period es una opción avanzada para el control de los pulsos de los servos en milisegundos. Es la cantidad de tiempo entre pulsos sucesivos para un canal determinado. Si no estás seguro acerca de esta opción, déjalo en el valor predeterminado que por defecto es de 20 ms. Mini Maestro 12, 18, y 24: las unidades de aceleración y de velocidad dependen de la frecuencia del pulso. Las unidades

sólo dependen en el período, no hay período multiplicador. Consulte la tabla siguiente para ver la relación entre las unidades de velocidad / aceleración:

Periodo (T)	Ratio	Unidades de velocidad	Unidades de aceleración
T = 20 ms	50 Hz	(0.25 µs)/(10 ms)	(0.25 µs)/(10 ms)/(80 ms)
T = 3–19 ms	> 50 Hz	(0.25 µs)/T	(0.25 µs)/T/(8T)
T > 20 ms	< 50 Hz	(0.25 µs)/(T/2)	(0.25 µs)/(T/2)/(4T)

Servos disponibles es una opción avanzada para Micro Maestro que especifica el número de canales que pueden utilizarse para controlar los servos. Por ejemplo, si este valor se establece en 4, sólo los canales 0 al 3 estarán disponibles, todos los demás canales deben establecerse como entrada o salida. Las razones para tener menos servos disponibles sólo son para permitir alargar la longitud de pulso máximo o recortar el período.

Period multiplier es una función avanzada para Mini Maestro 12, 18, y 24 que permite establecer un período largo (ratio corto) en algunos canales. Por ejemplo si seleccionas un periodo de 3 y un multiplicador de 6 tienes que algunos servos funcionan a 3 ms (333 Hz) y otros 18 ms (55 Hz). Cuando el multiplicador es mayor de 1, las opciones de pulso se muestran para cada canal en la columna **Rate.**

Para el Mini Maestro 24 existe una opción extra disponible:

Enable pull-ups en los canales 18-20 conecta resistencias de pull-up para los canales numerados si están configurados como entradas. Esto garantiza que el valor de entrada será alto cuando nada este conectado a la línea. Si habilitas esta característica te permite poder conectar pulsadores o interruptores en los canales 18, 19 o 20 sin necesidad de resistencias externas: simplemente conecta los actuadores entre la línea y GND.

4.f. Actualización del Firmware

Nota: *No han habido actualizaciones de firmware para por los Mini-Maestros todavía, por lo que esta sección sólo se aplica al Micro-Maestro 6 USB Servo*.

El Maestro tiene un firmware que puede actualizarse fácilmente para fijar errores o mejorarse con nuevas características.

Puedes determinar la versión de firmware de tu Maestro ejecutando Centro de Control Maestro, conectado al mismo y mirando el número de versión que se muestra en la esquina superior izquierda junto a la lista desplegable `"Connected to"`. La versión 1.01 del firmware para el Mini-Maestro de 6 canales tiene una corrección que hace "Ignorar" el modo para servos que se comportan correctamente al inicio. La actualización se recomienda para dispositivos con número de versión anterior, incluyendo todos los dispositivos de antes de 19 de noviembre de 2009.

Importante: No aplicar esta actualización a un Mini Maestro

Para actualizar firmware sigue estos pasos:

Guarda los ajustes almacenados en el Maestro usando `“Save settings file…”` en File menú. Todos los ajustes se ponen con los valores por defecto al realizar la actualización.

Descarga la última versión:

Firmware versión 1.01 para **Micro Maestro 6-Channel** (35k pgm) — desde 19-11-2009

Conecta el Maestro a Windows o Linux usando el cable USB.

Arranca la aplicación Pololu Maestro Control Center con Maestro conectado.

En el `Device menú`, selecciona `“Upgrade firmware…”`. Veras un mensaje preguntando si estás seguro de seguir: clic OK. El Maestro se desconecta del ordenador y aparece un nuevo dispositivo llamado “Pololu usc02a Bootloader”.

Windows 7 y Vista: el driver para el bootloader quedará automáticamente instalado.

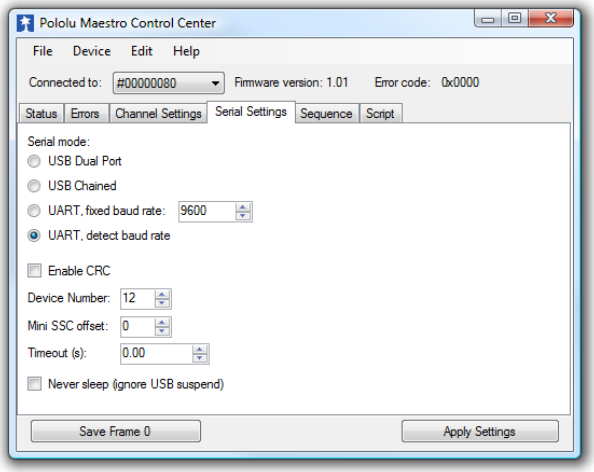
Windows XP: sigue los podas 6–8 de la sección 3.a para coger el driver que funcione.

Una vez los drivers de bootloader’s estén instalados correctamente el LED verde parpadeará en golpes de dos y deberás ver una entrada para el mismo en “Puertos (COM & LPT)” en el Administrador de dispositivos de tu ordenador.

Vete a la ventana `“Firmware Upgrade”` del Maestro Control Center y Clic en el botón `“Browse…”` y selecciona el fichero de firmware que quieres descargar.

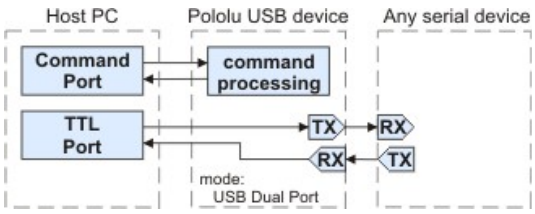
Selecciona el puerto COM port correspondiente al bootloader. Si no conoces el puerto mira en Administración de dispositivos en el apartado “Puertos (COM & LPT)”.
Clic en el botón `“Program”`. Verás un mensaje alertando que el firmware del dispositivo va a ser borrado y preguntando si quieres hacerlo: clic Yes.
Tardará unos segundos para borrar el firmware existente del Maestro’s y leer el nuevo.
No desconectes el Maestro durante una actualización de firmware.
Una vez se haya completado la ventana de actualización se cierra, el Maestro debe desconectarse del ordenador para más tarde reaparecer. Si solo hay un Maestro conectado el Maestro Control Center volverá a conectar con él. Comprueba la versión y asegúrate de que es la nueva.
Si tienes algún problema no dudes en contactar con nosotros.

5. Comunicación serie



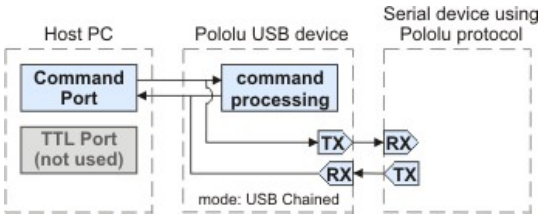
5.a. Ajustes

El Maestro dispone de tres modos de comunicación serie. Primero tiene las líneas **TX** y **RX** que permiten enviar y recibir bytes non-inverted TTL (0 – 5 V) (ver Sección 5.b). Segundo Maestro dispone de dos puertos virtuales serie por donde el ordenador se conecta a través del USB. Uno de estos puertos se llama **Command Port** y el otro **TTL port**. En Windows, tu puedes determinar el numero de puerto COM asignado mirando en Administración de dispositivos. En Linux el Command Port es generalmente /dev/ttyACM0 y el TTL Port /dev/ttyACM1. Estos números pueden ser diferentes según que cantidad de dispositivos serie estén activos al conectar. Esta sección explica las configuraciones disponibles en la pestaña `Serial Settings` en el Maestro Control Center.
Se puede configurar de tres maneras diferentes:
USB Dual Port: en este modo el Command Port se usa para enviar Comandos al Maestro y recibir respuestas de él. La velocidad en baudios es irrelevante. El TTL Port se usa para enviar bytes por la línea TX y recibirlos por la RX. La velocidad de comunicación que establezcas en tu programa terminal al abrir el puerto TTL determina la velocidad en baudios utilizada para recibir y enviar bytes RX y TX. Esto permite al control por Maestro y el poder utilizar simultáneamente las líneas de TX y RX como puerto serie de propósito general para que pueda comunicarse con otros tipos de dispositivos serie TTL.



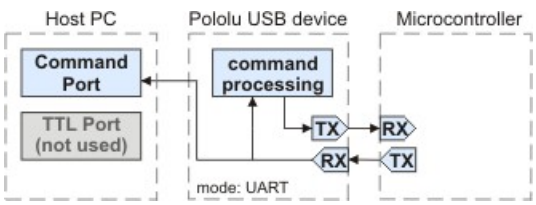
Modo USB Dual Port.

USB encadenado: En este modo el Comando Port se usa en ambos casos para transmitir bytes por la línea TX y enviar comandos al Maestro. El Maestro responde a los comandos enviados por el Command Port no por la línea TX. Los bytes recibidos por la línea RX son reenviados al Command Port pero no se interpretan como bytes de comando por Maestro. La velocidad de ajuste en tu programa terminal al abrir el Command Port determina la velocidad de recepción y envío de bytes por RX y TX. El puerto TTL no se usa. Este modo permite que un único puerto COM del ordenador pueda controlar múltiples Maestros y otros dispositivos compatibles con el protocolo utilizado.



Modo USB encadenado.

UART: En este modo, las líneas TX y RX se usan para enviar comandos al Maestro y recibir respuestas del mismo. Cada byte recibido por RX se envía al Command Port, pero los bytes enviados desde el Command Port son ignorados. El puerto TTL no se usa. La velocidad en baudios se detecta automáticamente por el Maestro cuando se recibe un byte `0xAA` por RX, o si lo fijamos a un valor concreto en bits/segs (bps). Este modo permite el control de Maestro (y el envío de bytes al programa en el ordenador) usando un microcontrolador u otro dispositivo TTL serie.



Modo UART.

Otros ajustes serie:

Enable CRC: Si esta activado, la transmisión requiere un byte de comprobación de redundancia cíclica (CRC) al final de cada comando excepto en los del Mini SSC (ver sección 5.d).

Device Number: Este es el número de dispositivo (0–127) que se usa para direccionar el dispositivo con los comandos del Pololu Protocol. Este ajuste es útil cuando usas el Maestro con otros dispositivos configurados en cadena (ver sección 5.g).

Mini SSC offset: Este parámetro determina que número de dispositivos servo responden al Mini SSC protocol (ver sección 5.e).

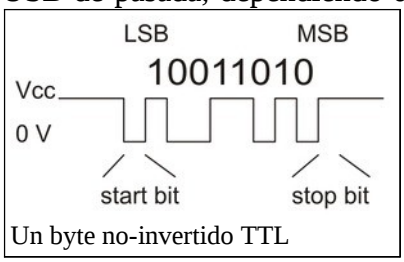
Timeout: Parámetro que determina la duración del *Serial timeout* para que se produzca error. Este error se usa como para asegurar que los servos y las salidas digitales retornan a su estado por defecto cuando el envío de comandos al Maestro se ha parado. El error de timeout también ocurre cuando se reciben comandos no válidos (o calificados como nativos USB) durante el periodo de tiempo marcado. El timeout a 0.00 lo deshabilita. La resolución es de 0.01s y el valor máximo es de 655.35 s. Los comandos nativos USB se corresponden a los métodos de la Usc class: `setTarget`, `setSpeed`, `setAcceleration`, `setPwm`, `disablePWM`, y `clearErrors`. Ejecutando el Maestro Control Center no evitará que el error de tiempo de espera se producen, sino más bien el ajuste de destinos en la pestaña Status o la reproducción de una secuencia

Never sleep (ignorar suspensión de USB): Por defecto, el procesador de Maestro se pone en reposo y detiene todas sus operaciones cuando detecta que USB ha entrado en suspensión (no hay suministro VIN). Sin embargo, se puede desactivar marcando la casilla de `Never sleep`.

5.b. TTL Serie

La línea RX puede recibir bytes cuando está conectada a una lógica no invertida (de 0 a 4.0–5V o "TTL"). Los bytes enviados al Maestro en RX pueden ser comandos o una secuencia arbitraria de datos que el Maestro pasa al ordenador mediante el puerto USB, dependiendo del modo serie configurado (sección 5.a). El voltaje en RX no debe ir por debajo de 0V ni debe exceder de 5V.

Maestro proporciona los niveles lógicos (0 a 5V) a la salida de su línea de transmisión TX. Los bytes enviados por Maestro en TX pueden ser respuestas a comandos que soliciten información o una secuencia arbitraria de datos que Maestro está recibiendo desde un ordenador desde el puerto USB de pasada, dependiendo del modo ajustado. Si no está interesado en recibir bytes TTL desde Maestro, puede dejar la línea TX desconectada. La transmisión es asíncrona, lo que significa que la recepción y el envío llevan bits a velocidad independiente. TTL asíncrona también está disponible en los módulos hardware llamados "UARTs" de muchos microcontroladores. La salida asíncrona puede ser también "bit-banged" por una línea estándar de salida digital bajo control del software.



Un byte no-invertido TTL

El formato de datos es de 8 bits, un stop bit y sin paridades decir 8-N-1. la imagen de abajo muestra el asincronismo, non-inverted de un byte serie TTL:

Una línea serie TTL invertida tiene como estado predeterminado (no activo) en alto. Un byte transmitido comienza en bajo "start bit" seguido por los bits del byte, empezando por el bit menos significativo (LSB). Los valores lógicos se transmiten como unos o altos (VCC) y ceros o bajos (0V), por lo que este formato es "no invertido". El byte se termina por un "bit de parada" que deja la línea en alto durante poco tiempo. Debido a que cada byte requiere un bit de inicio, 8 bits de datos y

un bit de parada, cada byte tiene 10 veces más bits para transmitir, luego la velocidad más rápida posible en bytes por segundo es la velocidad dividida por diez. La máxima velocidad de transmisión de Maestro es de 250.000 bits por segundo, luego la máxima transferencia de datos empezara inmediatamente después del bit de parada del byte precedente y será de 25.000 bytes por segundo.
Cada vez que conectes dispositivos, *no olvides* conectar la líneas de masa juntas, y asegurarte de que cada dispositivo se alimentada adecuadamente. *Los d*ispositivos sin alimentación con puerto serie TTL pueden generar en parte, *cargas* eléctricas en la línea, lo que significa que *hay que tener precaución* al desconectarlos o resetearlos.

5.c. Protocolos de comandos

Para el control de Maestro se usan comandos serie.
Si Maestro esta en "`UART, detect baud rate`", primero debes enviar el indicador de baudios con el byte `0xAA` a la línea RX y después los comandos y datos necesarios.
El protocolo de comandos para Maestro es similar a la de otros productos Pololu. La comunicación se logra mediante el envío de paquetes de comandos que constan de un byte de comando seguido de los bytes de datos que se requieren. Los bytes de comando tienen siempre los bits más significativos altos (`128–255, 0x80–0xFF`) mientras los menos significativos desactivados (`0–127, 0x00–0x7F`). Esto significa que cada byte de datos sólo puede llevar siete bits de información. La única excepción a esto son los comandos Mini SSC, donde los bytes de datos pueden tener cualquier valor comprendido entre 0 y 254.
El Maestro responde a tres protocolos:

Compact

Es el más simple y compacto de los dos protocolos y es el que deberías usar si tu dispositivo es el único conectado a la línea serie. El paquete con este protocolo es simple:
`Comando byte (con MSB alto), los bytes de datos necesarios`
Por ejemplo, para ajustar el destino del servo de 0 a 1500 µs, deberás enviar la secuencia de bytes:

en hex: `0x84, 0x00, 0x70, 0x2E`
en decimal: `132, 0, 112, 46`

El byte `0x84` corresponde al comando `Set Target`, el primer byte de datos `0x00` es el número de servo y los dos últimos bytes de datos contienen el destino en cuartos de microsegundos.

Pololu

Este protocolo es compatible con los usados por otros dispositivos controladores de servos y motores. Como tal, puedes encadenar un Maestro a nuestros controladores serie (incluidos los Maestros adicionales) en una sola línea y con este Protocolo enviar comandos específicamente al Maestro deseado sin confundir a los otros dispositivos de la línea. Para utilizar el protocolo Pololu, debes transmitir `0xAA` (`170` dec) como primer byte (comando) seguido de un byte de datos con el número de dispositivo. El byte `0xAA` siempre será el primer byte a utilizar con protocolo Pololu.
El número de dispositivo predeterminado para Maestro es **12**, pero se trata de un parámetro configurable que se puede cambiar. Cualquier maestro en la línea cuyo número de dispositivo coincida con el especificado acepta el comando que sigue, todos los demás dispositivos Pololu lo ignoraran. Los bytes restantes en el paquete de comandos son los mismos que los en un paquete de protocolo compact y que se enviarán, con una diferencia clave: el byte de protocolo de comando compact es ahora un byte de datos para el comando 0xAA y por lo tanto debe tener su bit más significativo aclarado. Por lo tanto, el paquete de comandos es:
`0xAA, byte numdisp, byte comando con MSB desactivado, bytes precisos de datos.`
Por ejemplo, lo mismo que antes para Maestro con nº de dispositivo 12, enviarás la secuencia:

en hex: `0xAA, 0x0C, 0x04, 0x00, 0x70, 0x2E`
en decimal: `170, 12, 4, 0, 112, 46`

Fíjate el `0x04` es el Comando `0x84` con el bit más significativo a cero.

Mini SSC

Maestro también responde al protocolo usado por el controlador Mini SSC. Este protocolo te permite el control de hasta 254 servos diferentes con el encadenado de controladoras. Solo necesita tres bytes para ajustar el destino del servo, por lo que es un buen protocolo si necesitas enviar

comandos de forma rápida. El protocolo Mini SSC transmite un `0xFF` (255 en decimal) como primer byte de comando seguido del byte de numero de servo y otro de bits con el destino a realizar. El paquete sería el siguiente:

```
0xFF, byte numservo, byte destinoservo
```

Por ejemplo si estas buscando ajustar el destino del servo 0 a la posición neutral, debes enviar la secuencia siguiente:

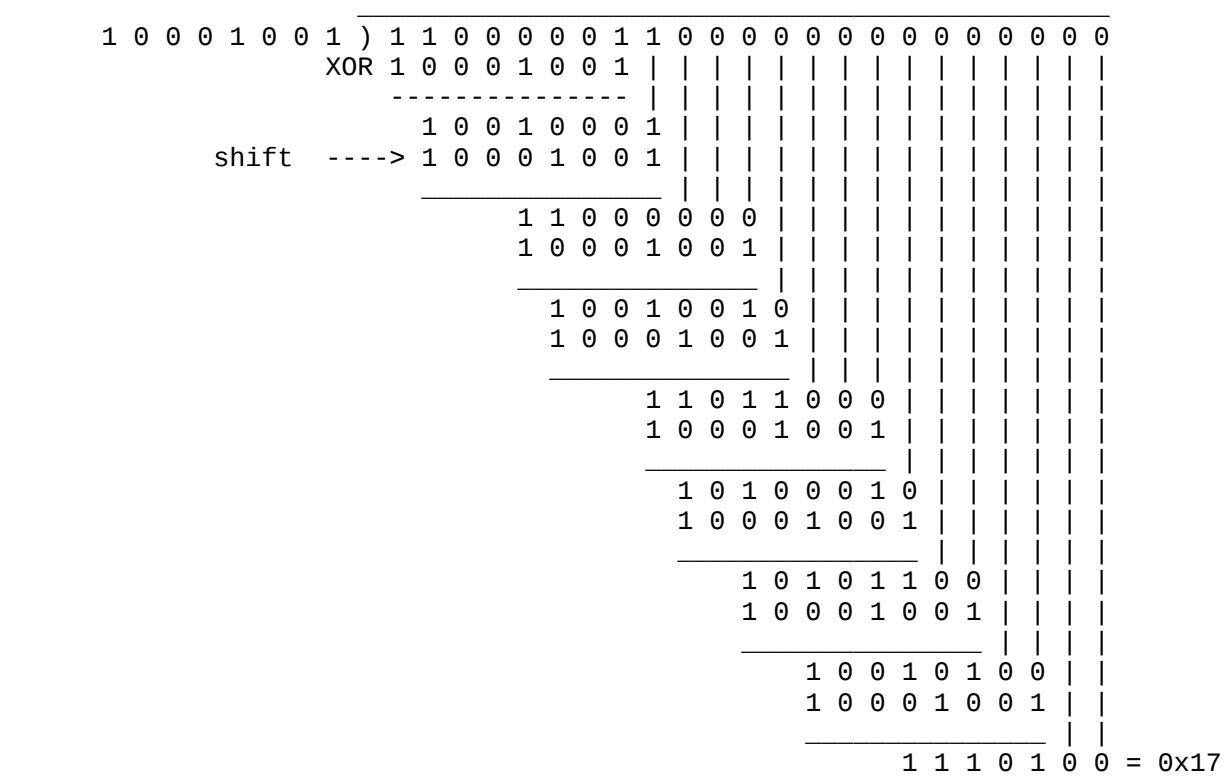
en hex: `0xFF, 0x00, 0x7F`
en decimal: `255, 0, 127`

Maestro puede configurarse para responder a bloques contiguos de servos numerados de 0 a 254. Maestro identifica los protocolos Pololu, Compact y Mini-SSC en el acto y no es necesario usar parámetros para detectar el protocolo. Incluso puedes mezclar comandos de los tres.

5.d. Control de redundancia cíclica (CRC) Detección de errores

Para algunas aplicaciones es importante la verificación de la integridad de los datos que se envían o reciben. Maestro puede realizar un chequeo de 7-bit CRC, que es algo similar a un checksum pero mas robusto y que puede detectar errores que quizás no afectarían a un checksum, como un byte 0 extra o bytes fuera de orden. La verificación se realiza habilitando la casilla `"Enable CRC"` de la pestaña `"Serial Settings"` de Maestro Control Center. En modo CRC, Maestro espera la recepción de un byte extra añadido al final del paquete de comandos (excepto en los de Mini SSC). El bit más significativo de este byte se pone a 0 y los siete bits menos significativos corresponden a los siete bit de control. Si el byte CRC es incorrecto, Maestro puede activar un bit de error *Serial CRC* al registro e ignorar el comando. Maestro no anexa un byte CRC a los datos que transmite en respuesta a los comandos. Una lista detallada de cómo hace la comprobación se puede encontrar en Wikipedia. Maestro utiliza CRC-7, lo que significa utiliza un polinomio de 8 bits y como resultado produce un resto de 7 bits. En notación binaria el número 0x91 es 10010001. Sin embargo los bits se transmiten en este orden: 1, 0, 0, 0, 1, 0, 0, 1, por lo que utilizamos 10001001 para el cálculo.

```
Pasos 1 & 2 (escribe binario, primero bit menos significativo añade 7 ceros al
final del mensaje):
    CRC-7 Polynomial = [1 0 0 0 1 0 0 1]
    mensaje = [1 1 0 0 0 0 0 1] [1 0 0 0 0 0 0 0] 0 0 0 0 0 0 0
Pasos 3, 4, & 5:
```



Luego el paquete total para el comando que se envía con el CRC habilitado es: `0x83, 0x01, 0x17`.

5.e. Comandos para servos

Maestro tiene varios comandos para ajustar el canal, recoger la posición actual y ajustar limites de velocidad y aceleración.

Set Target (Pololu/Compact protocol)

Compact protocol: `0x84,numcanal,destinoLB,destinoHB`

Pololu protocol: `0xAA,numdispos,0x04,numcanal,destinoLB,destinoHB`

Los 7 bits bajos (LB) del 3er byte representan los bits 0–6 del destino (7 bits bajos), los 7 bits bajos del cuarto byte de datos representan los bits 7–13 del destino. El destino no es entero negativo.

Si el canal está configurado para servo, el destino representa el ancho de pulso a transmitir en cuartos de microsegundo. El valor destino 0 indica a Maestro que pare el envío de pulsos al servo.

Si el canal está configurado como salida digital los valore menores de 6000 indican que debe poner la línea en bajo, en cambio si son mayores la línea estará en alto.

Por ejemplo, si el canal 2 está configurado para servo y quieres ajustar el destino a 1500 µs (1500×4 = 6000 = 01011101110000 en binario), enviarás la siguiente secuencia de bytes:

en binario: `10000100, 00000010, 01110000, 00101110`
en hex: `0x84, 0x02, 0x70, 0x2E`
en decimal: `132, 2, 112, 46`

Aquí hay algunos ejemplos en código C que generará los bytes correctos, dando un número entero al "canal" que contiene el número del canal, un entero "destino" para el destino deseado (en unidades de cuartos de microsegundos si es servo) en una matriz denominada serialBytes:

```
serialBytes[0] = 0x84;                  // Comando Set Target.
serialBytes[1] = channel;               // 1º byte datos numerocanal.
serialBytes[2] = target & 0x7F;         // 2º byte los LB 7 bits destino.
serialBytes[3] = (target >> 7) & 0x7F;  // 3º byte los HB 7-13 destino.
```

Algunas aplicaciones de servo control no necesitan una resolución en cuartos de microsegundo. Si quieres un pequeño paquete de comandos de baja resolución para trabajar usa Mini-SSC.

Set Target (Mini SSC protocol)

Mini-SSC protocol: **0xFF, direccioncanal, 8-bit destino**

Este comando ajusta el destino del canal con un valor de 8 bits, entre 0 y 254. El valor objetivo de 8 bits se convierte en otro de alta resolución de acuerdo con la configuración neutral y rango almacenados en Maestro para ese canal. Entonces 127 corresponde a neutral en ese canal, mientras que 0 o 254 corresponden a neutral menos o más el valor del rango. Esta configuración puede ser útil para calibrar la marcha sin cambiar el programa que envía los comandos.

La dirección del canal es un valor entre 0–254. Por defecto la dirección de canal es igual al número de canal y está entre 0 y 23. Para permitir múltiples Maestros controlados por la misma línea debes establecer el parámetro `Mini SSC Offset` con diferentes valores para cada Maestro. El Mini SSC Offset se añade al número de canal para dirigir el comando a la dirección correcta. Por ejemplo Micro Maestro de 6 canales con el Mini SSC Offset de 12 obedece a los comandos cuya dirección este entre 12–17.

Set Multiple Targets (Mini Maestro 12, 18, y 24 solo)

Compact protocol:
`0x9F,numdestinos,1numcanal,1destinoLB,1destinoHB,2destinoLB,2destinoHB, …`

Pololu protocol:
`0xAA,numdispos,0x1F,numdestinos,1numcanal,1destLB,1destHB,2destLB,2destHB, …`

El comando ajusta simultáneamente los destinos con bloques contiguos de canales. El primer byte especifica cuantos canales hay en el bloque; será el numero de valores de destinos que necesitas enviar. El segundo byte especifica el numero de canal bajo en el bloque. Los siguientes bytes contienen los valores de destino para cada canal en orden de número de canal con el mismo formato que el Set Target Comando. Por ejemplo, para ajustar canales 3 a 0 (off) y canal 4 a 6000 (neutral), enviaras los siguientes bytes:

`0x9F,0x02,0x03,0x00,0x00,0x70,0x2E`

El comando `Set Multiple Targets` permite la rápida actualización de Maestro, y se usa principalmente cuando se controlan un largo número de servos configurados en cadena. Por

ejemplo, usando el protocolo Pololu protocol a 115.2 kbps, enviando el comando a 24 servos tarda --4.6 ms, mientras si los envías de forma individual tardaría 12.5 ms.

Set Speed

Compact protocol: `0x87,numcanal,velocidadLB,velocidadHB`
Pololu protocol: `0xAA,numdispos,0x07,numcanal,velocidadLB,velocidadHB`

Este Comando limita la velocidad del servo al cambiar el valor del canal de salida. El límite de velocidad se da en unidades de 0.25 µs/10 ms excepto en casos especiales (ver sección 4.e). Por ejemplo, el comando `0x87,0x05,0x0C,0x01` establece la velocidad del canal 5 en un valor de 140, que corresponde a una velocidad de 3.5 µs/ms. Esto significa que si envía un comando de `Set Target` para ajustar el destino del servo de 1000 µs a 1350 µs, tardará 100 ms para realizar ese ajuste. Una velocidad de 0 es una velocidad ilimitada, así el objetivo será inmediato para llegar a la posición. Piensa que la velocidad real de los movimientos cinemáticas también está limitada por el propio diseño del servo, la tensión de alimentación y de la carga, por lo que este parámetro no servirá para ir más rápido de lo que físicamente es capaz.

Con la configuración de velocidad mínima de 1, la salida de servo tarda 40 segundos para pasar de 1 a 2 ms. La configuración de velocidad no tiene ningún efecto en los canales configurados como entradas o salidas digitales.

Set Acceleration

Compact protocol: `0x89,numcanal,aceleraciónLB,aceleraciónHB`
Pololu protocol: `0xAA,numdispos,0x09,numcanal,aceleraciónLB,aceleraciónHB`

Este comando limita la aceleración en la salida del canal servo. La aceleración puede tener un valor entre 0 y 255 en unidades de (0.25 µs)/(80 ms)/(10 ms) excepto en casos especiales (ver sección 4.e). El valor 0 se corresponde con ningún límite de aceleración. La aceleración limitada causa que la velocidad del servo ascienda lentamente hasta alcanzar la velocidad máxima para luego descender hasta llegar a la posición destino, resultando de ello, un movimiento relativamente suave de un punto a otro.

Con la aplicación de límites en velocidad y aceleración, solo harán falta unos pocos ajustes para que llegar al destino, se realice con movimientos de aspecto natural y que de lo contrario serían bastante más complicados de crear. Configurando la aceleración mínima a 1, el servo tarda unos 3 segundos en moverse desde una posición de 1ms a un destino de 2ms.

La aceleración no tiene efecto en los canales configurados como entradas o salidas digitales

Set PWM (Mini Maestro 12, 18, y 24 solo)

Compact protocol: `0x8A,pulsoLB,pulsoHB,periodoLB,periodoHB`
Pololu protocol: `0xAA,numdispos,0x0A,pulsoLB,pulsoHB,periodoLB,periodoHB`

Este comando ajusta la salida PWM en pulso y periodo con unidades de 1/48 µs. El pulso y el periodo se codifican en bytes de 7 bits de la misma manera que destino con el comando 0x84, anterior. Para más información sobre el PWM, ver sección 4.a.

La salida PWM no está disponible en el Micro Maestro.

Get Position

Compact protocol: `0x90,numcanal`
Pololu protocol: `0xAA,numdispos,0x10,numcanal`

Respuesta: un byte LB de posición y un byte HB de posición ambos en 8 bits

El comando pregunta al dispositivo que se comunica con Maestro por la posición en que se encuentra el canal. El valor se envía de inmediato en dos bytes tras haber recibido el comando.

Si el canal especificado esta como servo el valor representa el ancho de pulso que Maestro está emitiendo por el canal, reflejando los efectos de comandos previos como velocidad y aceleración o scripts funcionando en la placa.

Si el canal esta configurados como salida digital y el valor es menor de 6000, la línea está en bajo, si el valor es mayor de 6000 es que la línea esta en alto.

Si el canal esta configurado como entrada, la posición representa el voltaje medido en el canal. Los canales 0-11 son analógicos y el rango de valores será entre 0 y 1023 para voltajes de 0 a 5V. Las entradas en los canales 12-23 son digitales y sus valores serán de 0 o 1023 exactamente.

Fíjate que el formato de este comando difiere de los de destino, velocidad y aceleración

Dado que no existe ninguna restricción sobre el bit alto, la posición es formateada como un entero sin signo estándar dos bytes primero el bajo luego el alto.
Por ejemplo, a la posición 2567 le daría como respuesta de 0x07, 0x0A.
El valor devuelto de la posición es igual a cuatro veces el número mostrado en la caja `Position` en la pestaña `Status` del Maestro Control Center.

Get Moving State

Compact protocol: `0x93`
Pololu protocol: `0xAA,numdispositivo,0x13`

Respuesta: 0x00 si los servos no se mueven, 0x01 si se mueven.
El comando se usa para determinar cual de los servos ha llegado a su destino o cuales están moviéndose en función de la velocidad y la aceleración.
Este comando combinado con `Set Target` permite iniciar muchos movimientos y esperar a que unos u otros terminen su trabajo antes de seguir con las líneas de programa.

Get Errors

Compact protocol: `0xA1`
Pololu protocol: `0xAA,numdispositivo,0x21`

Respuesta: Un byte de error con los bits 0-7, otro con los bits 8-15
Este comando se usa para examinar los errores que pueda detectar Maestro. En la sección 4.b hay una lista específica de errores posibles. El registro de error envía dos bytes de respuesta después de recibir el comando y seguidamente limpia el registro. Para muchas de las aplicaciones que usan el control serie es buena idea chequear errores para realizar las acciones apropiadas si estos ocurren.

Go Home

Compact protocol: `0xA2`
Pololu protocol: `0xAA,numdispositivo,0x22`

Este comando envía todos los canales a la posición inicial, tanto si están configurados como servos o si lo están como salidas, en el caso de producirse algún error.
Para servos y salidas con el ajuste a `"Ignore"` hace que la posición no cambie.
Note: Para los servos marcados `"Off"`, si ejecutas **Set Target** inmediatamente después de **Go Home**, parecerá que estos no obedecen a la velocidad y aceleración. De hecho cuando el servo se desactiva Maestro no tiene manera de saber donde estaba por lo que pueden moverse a cualquier destino. Los siguientes comandos ya funcionaran correctamente.

5.f. Comandos serie para Script

Maestro dispone de varios comandos para el control de ejecución del script de usuario.

Stop Script

Compact protocol: `0xA4`
Pololu protocol: `0xAA,numdispositivo,0x24`

Para la ejecución del script.

Restart Script at Subroutine

Compact protocol: `0xA7,numsubrutina`
Pololu protocol: `0xAA,numdispositivo,0x27,numsubrutina`

Arranca el script en el punto indicado por el argumento de número de la subrutina. Las subrutinas están numeradas en el orden en que se haya definido el script, empezando con 0 para la primera. La primera subrutina se envía como 0x00 para este comando, 0x01 para la segunda, etc. Para encontrar el numero de subrutina particular haga clic en `"View Compiled Code…"` y mira la lista. Las subrutinas usadas de esta manera no deben terminar en RETURN (no tiene sentido), en su lugar debe poner bucles infinitos o el comando QUIT.

Restart Script at Subroutine with Parameter

Compact protocol:
`0xA8,numsubrutina,parámetroLB,parámetroHB`
Pololu protocol:
`0xAA,numdispos,0x28,numsubrutina,parámetroLB,parámetro HB`

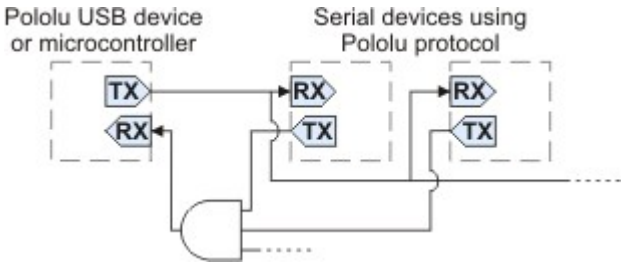
Este comando es igual que `Restart Script at Subroutine`, excepto que carga un parámetro en la pila antes de iniciar la subrutina. Como bytes de datos solo puede contener 7 bits de datos y el parámetro debe estar entre 0 y 16383.

Get Script Status

Compact protocol: 0xAE
Pololu protocol: 0xAA, numdispositivo, 0x2E
Respuesta: 0x00 si el script esta funcionando, 0x01 si está parado
El comando responde con un byte para indicar si el script esta en funcionamiento (0) o parado (1) .
Mediante este Comando junto con los anteriores, puede activar una secuencia almacenada en Maestro y esperar hasta que se complete antes de pasar a la siguiente línea de programa.

5.g. Encadenamiento

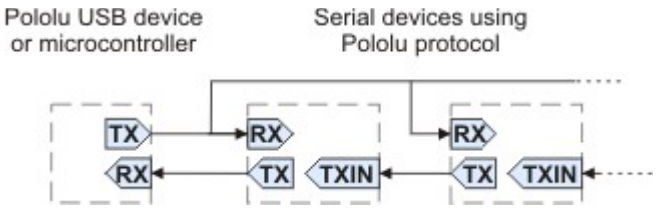
Esta sección es una guía para la integración de Maestro en un proyecto con múltiples dispositivos serie que usan un protocolo compatible. No es información nueva: toda la información se puede deducir a partir de los tres modos conexión de los protocolos usados. (Sección 5.a y 5.c).
En primer lugar, debes decidir que protocolo vas a utilizar Pololu, Mini SSC o una combinación de ambos. Debes asegurarse de que ningún comando que vas a enviar causará operaciones en los dispositivos que no estén direccionados. Si deseas encadenar varios Maestros puedes usar una mezcla de ambos protocolos. Si deseas que Maestro se encadene con otros dispositivos que usan el protocolo Pololu utiliza este protocolo. Si deseas realizar encadenamientos del Maestro con otros dispositivos que usan el protocolo Mini SSC puedes utilizar para todos este protocolo.



Daisy chaining serial devices using the Pololu protocol. An optional y gate is us*ed to join multiple TX lines.*

En segundo lugar, asignar a cada dispositivo del proyecto un número distinto o con Mini SSC (offset) compensar para que cada dispositivo pueda tratarse individualmente con sus comandos.
En Maestro esto puede hacerse desde la pestaña `Serial Settings` de Maestro Control Center. El siguiente diagrama muestra cómo conectar un maestro y varios dispositivos esclavos en cadena.
Cada dispositivo puede ser Maestro o cualquier otro, como un jrk, qik o microcontroladores.
El Mini Maestro 12, 18, y 24 tiene una entrada especial llamada **TXIN** que elimina la necesidad de una puerta AND (la puerta AND esta en Maestro.) Para realizar encadenamientos de dispositivos usaremos la entrada **TXIN** conectándola de la siguiente manera:

Uso de un PC y Maestro juntos como dispositivo maestro



Daisy chaining serial devices that have a TXIN input.

Maestro puede habilitar el dispositivo maestro al PC. La conexión entre el PC y Maestro se hará con el cable USB y configurado la conexión en modo USB Dual Port o USB Chained. En modo USB Dual Port el puerto del PC se utiliza para enviar directamente comandos al Maestro y el puerto TTL de PC se utiliza para enviar comandos a todos los dispositivos esclavos. En modo USB encadenado, sólo el Port del PC se utiliza por comunicarse con Maestro y todos los dispositivos esclavos. Seleccione el modo que sea más eficaz para su aplicación o más fácil de implementar en su programación.

Usando Maestro como dispositivo esclavo

El Maestro puede actuar como dispositivo esclavo cuando está configurado en el modo UART. En este modo, recibe comandos por la línea RX y se responde por la línea TX. La conexión USB a PC no es necesaria, aunque el puerto RX del PC se utilizará para propósitos de depuración u otros.

Conexiones

Conectar la línea TX del dispositivo maestro a las líneas RX de todos los dispositivos esclavos. Los comandos enviados por el master, a continuación, serán recibidos por todos los esclavos. La recepción de respuestas de uno de los dispositivos esclavos en el PC puede lograrse mediante la conexión de la línea TX de ese dispositivo subordinado a la línea RX del Maestro. Recibir respuestas de múltiples dispositivos esclavos es más complicado. Cada dispositivo debe transmitir sólo cuando se lo soliciten, por lo que si cada dispositivo se aborda por separado, no podrán

transmitir simultáneamente varios. Sin embargo, las salidas TX se ponen en alto al no enviar datos, por lo que simplemente no pueden estar conectados juntos. En su lugar, puede utilizar una puerta AND como se muestra en el diagrama para combinar las señales. Observa que en muchos casos el recibir respuestas no es necesario por lo que las líneas TX pueden quedar desconectadas.
Cuando haya varios dispositivos recuerda la conexión a tierra de todos ellos y asegurarte de que cada dispositivo tiene la alimentación apropiada. Los dispositivos apagados con puerto serie TTL pueden activarse en parte, dando alimentación a la línea, lo que significa que debes tener cuidado al apagarlos o al resetearlos.

Enviando Comandos

El Pololu Protocol o el Mini SSC protocol pueden usarse cuando múltiples dispositivos Pololu están recibiendo los mismos datos vía serie. Esto permite a los dispositivos tener direcciones individuales y las respuestas que se envíen no colisionen. Si los dispositivos están configurados para detectar la velocidad en baudios, entonces cuando se emita el primer comando de protocolo Pololu, los dispositivos podrán detectar automáticamente la velocidad desde el byte de inicio 0xAA. Algunos dispositivos antiguos de Pololu utilizan 0x80 como byte inicial. Si la cadena de dispositivos espera 0xAA, primero podemos transmitir 0x80 para que estos dispositivos detecten la velocidad y sólo entonces enviar el byte 0xAA para que el Maestro pueda también detectarla. Una vez que todos los dispositivos han detectado los baudios, los dispositivos Pololu que esperan que un byte 0x80 ignoraran el 0xAA y Maestro pasará hará lo mismo con el comando 0x80.

5.h. Ejemplo de código de transmisión

5.h.1. PIC18F4550

El siguiente ejemplo de código para un PIC18F4550 ha sido remitido por Ney Palma Castillo. Ha de ser compilado con el compilador de C de PIC MCU. El código muestra cómo controlar un Maestro mediante comandos serie desde los pins C6 y C7. El primer comando para el canal 0 de servo sirve para llevarlo a su mínima posición y después un segundo más tarde, para ir a la posición neutral. Ver sección 5.e para obtener información sobre la conexión del PIC al Maestro y definir el modo serie para Maestro "UART, detect baud rate."

```
#include<18f4550.H>
#fuses HSPLL, NOMCLR, PUT, BROWNOUT, BORV43, NOWDT, NOPROTECT, NOLVP
#fuses NODEBUG, USBDIV, PLL5, CPUDIV1, VREGEN, CCP2B3
#use delay(clock=48000000)
#define TTL_TX1 PIN_C6
#define TTL_RX1 PIN_C7
#use rs232(xmit=TTL_TX1, rcv=TTL_RX1, bits=8, parity=N)
void main() {
    delay_ms(2000);
    while(true) {
        // Envia Set Target usando protoclo Pololu.
        putc(0xAA);  // Byte de arranque
        putc(0x0C);  // Dispositivo ID = 12
        putc(0x04);  // Comando = Set Target
        putc(0x00);  // Canal = 0
        putc(0x20);  // Posicion destino = 1000 us (minima para servos)
        putc(0x1F);
        delay_ms(1000); // espera un segundo
        // Envia Set Target usando protoclo Pololu.
        putc(0xAA);
        putc(0x0C);
        putc(0x04);
        putc(0x00);
        putc(0x70);  // Posicion destino = 1500 us (neutral para servos)
        putc(0x2E);
        delay_ms(1000);
    }
}
```

6. Lenguaje de scripts de Maestro

El script es una secuencia de comandos que puede ejecutar Maestro. Los comandos pueden servir para ajustar el destino del servo, la velocidad y la aceleración entrando valores y realizando operaciones matemáticas. Se dispone de estructuras básicas de control – bucles y condicionales – disponibles para crear scripts complejos. El lenguaje de scripts de Maestro es simple, muy parecido a FORTH y la compilación crea un código compacto en donde comandos y subrutinas tienen un solo byte. Un editor/depurador básico está en la pestaña `Scripts` del Maestro Control Center.

6.a. Lenguaje básico para scripts

Comandos y pila

La programación de scripts consiste en crear una secuencia de comandos que actúan como una pila de valores. Los valores son enteros que van desde -32768 a +32767. En el Micro Maestro de seis canales hay espacio para 32 valores en la pila y en los Mini Maestro este pila llega hasta 126 valores. Los comandos actúan siempre con los valores altos de la pila y dejan los resultados en la parte superior de la misma. Los comandos son un simple juego de literales, valores numéricos que se van colocando directamente en la pila. Por ejemplo el programa “-10 20 35 0” mete los valores -10, 20, 35 y 0 de forma secuencial.

lugar	valor
3	0
2	35
1	20
0	-10

El comando más complicado es el comando PLUS, el cual añade arriba dos números dejando el resultado en la parte superior de la pila.

Supongamos los números 1, 2, 4 y 7 colocados secuencialmente en la pila y el comando PLUS esta en funcionamiento. La siguiente tabla muestra el resultado: Fíjate que PLUS siempre hace menguar el tamaño de la pila en uno.

lugar	antes	después
3	7	
2	4	11
1	2	2
0	1	1

Depende de ti asegurarse de que tienes suficientes valores en la pila para completar los comandos que deseas ejecutar. Veamos un ejemplo más complicado: Supongamos que queremos calcular el valor de (1-3) × 4, usando los comandos MINUS y MULTIPLY. La forma de escribir este cómputo como una secuencia de comandos es "1 3 MINUS 4 TIMES".

Comentarios, caso, espacios en blanco, saltos de línea

Todas las partes del lenguaje de secuencias de comandos de Maestro son mayúsculas y minúsculas, y pueden utilizar cualquier tipo de espacios en blanco (espacios, tabulaciones, saltos de línea) para separarlos. Los comentarios se indican mediante el signo "#": desde el # hasta el final de línea será ignorado por el compilador. Por ejemplo, el cálculo anterior puede escribirse como:

```
1 3  miNUS
4  # this is a comment!
  times
```

que no afectará a la compilación. Generalmente usamos minúsculas para comandos y dos o cuatro espacios de indentación para las estructuras de control y subrutinas pero tu puedes hacerlo como quieras.

Estructuras de control

El lenguaje de scripts de Maestro tiene varias estructuras de control, que permiten escribir complicados programas. A diferencia de las subrutinas, no hay ningún límite en el nivel de anidamiento de las estructuras de control ya todas en última instancia están basadas en comandos GOTO (ver mas adelante) y en una sencilla ramificación. La estructura de control más útil es el BEGIN…REPEAT, bucle infinito, un ejemplo de lo que se da a continuación:

```
# move servo 1 back y forth con un periodo de 1 segundo
begin
  8000 1 servo
  500 delay
  4000 1 servo
  500 delay
repeat
```

Este bucle continuará siempre. Si quieres un bucle que pueda salirse alguna vez debes utilizar WHILE que es lo mejor para ese caso. WHILE consume mayor número en la pila y salta al final del bucle en caso de ser 0. Por ejemplo, supongamos que queremos repetir el bucle 10 veces:

```
  10 # arranca con 10 en la pila
  begin
    dup     # copia el numero a la pila - the copy will be consumed by WHILE
    while   # salta al final si contador es 0
            8000 1 servo
            500 delay
            4000 1 servo
            500 delay
            1 minus # resta 1 del numero de veces que quedan
  repeat
```

Fíjate que los bucles BEGIN…WHILE…REPEAT son similares a los bucles de muchos lenguajes pero solamente en los scripts de Maestro, el WHILE salta *después* de su argumento.

Para las acciones condicionales el IF…ELSE…ENDIF también se usan. Supongamos que queremos construir un sistema con un sensor en el canal 3 y queremos ajustar el servo 5 a 6000 (1,5ms) si el valor de entrada es menor de 512 (cerca de 2,5V). En caso contrario el servo se ajusta a 7000 (1,75ms). El código siguiente realiza esta labor complicada:

```
 3 get_position   # coge el valor de la entrada 3 como numero de 0 a 1023
 512 less_than    # comprueba si es menor de 512 -> 1 cierto, 0 es falso
 if
   6000 5 servo   # esta parte funcioan si input3 < 512
 else
   7000 5 servo   # esta parte funciona cuando input3 >= 512
 endif
```

En muchos lenguajes ELSE es opcional. Tenga en cuenta una vez más que esto parece en principio ir hacia atrás respecto a otros lenguajes, ya viene *después* la condición. Las estructuras de WHILE y de IF son suficientes para crear cualquier tipo de secuencia de comandos. Sin embargo, hay momentos en que es conveniente reflexionar sobre lo que está intentando hacer un bucle o una rama. Si sólo desea saltar directamente de una parte del código a otra, puede utilizar un GOTO.

```
 goto mylabel
            # ...cualquier línea de código se salta...
 4000 1 servo     # pasa por encima de esta línea
 mylabel: # el programa continua  a partir de aqui
 4000 2 servo
```

Solo el servo 2 se pone a 4000, mientras que el servo 1 no cambia ya que la etiqueta esta después.

Subrutinas

Puede ser útil usar la misma secuencia de comandos varias veces a lo largo del programa. Las subrutinas se utilizan para hacer más fácil y eficiente esto. Por ejemplo, supongamos es necesario establecer varias veces una posición neutral de 6000 (1,5 ms) utilizando la secuencia "servo 6000 1 servo 6000 2". Se puede encapsular en una subrutina como sigue:

```
 sub neutral
   6000 1 servo
   6000 2 servo
   return
```

Cuando quieras realizar la operación para los servos solo tienes que escribir como comando “neutral”. Hay rutinas avanzadas que pueden tomar valores de la pila. Por ejemplo la subrutina:

```
 sub set_servos
   2 servo 1 servo
   return
```

ajusta el canal 2 según valor de la parte superior de la pila y el canal 1 según el valor siguiente. Luego si escribes “4000 6000 set_servos”, el script ajusta el canal 1 a 4000 y el canal 2 a 6000.

Hay subrutinas que pueden llamar a otras subrutinas, hasta un límite de 10 niveles de recursividad. Por ejemplo, puede aplicarse la subrutina "neutral" por encima de la llamada a set_servos:

```
 sub neutral
   6000 6000 set_servos
 return
```

6.b. Referencia de comandos

Esta tabla es una lista de los comandos y palabras usadas en el lenguaje de scripts de Maestro. La columna “efecto” especifica cuantos números de pila consumen y que se añaden a la pila. Por ejemplo el comando PLUS coge dos números y devuelve uno; luego el “efecto” en la pila será -2+1. Los comandos con efectos especiales se marcan con *.

palabras	efecto	descripción
BEGIN	nada	Marca el principio del bucle
ENDIF	nada	Finaliza un bloque condicional IF…ENDIF
ELSE	nada	Empieza un bloque alternativo en IF…ELSE…ENDIF
GOTO *label*	nada	Salta a la etiqueta *label* (definida con *label*:)
IF	-1	Entra en un bloque condicional y el argumento es verdadero (no-cero) en IF…ENDIF o IF…ELSE…ENDIF
REPEAT	nada	Marca el final de un bucle
SUB *name*	nada	Define una subrutina de nombre *name*
WHILE	-1	Salta al final del bucle si el argumento es falso (cero)
De control	efecto	descripción
QUIT	nada	Para el script
RETURN	nada	Final de subrutina
De tiempo	efecto	descripción
DELAY	-1	Retardo en ms
GET_MS	+1	Valor actual del timer en ms (entre 32767 a -32768)
De pila	efecto	descripción
DEPTH	+1	Recoge el cantidad de números de la pila
DROP	-1	Remueve el numero superior de la pila
DUP	+1	Duplica el numero superior
OVER	+1	Duplica el numero siguiente al superior copiando sobre el superior
PICK	-1,+1	takes a number n, then puts the nth number below the top onto the stack (0 PICK is equivalent to DUP)
SWAP	a,b → b,a	Intercambia los dos números superiores
ROT	a,b,c → b,c,a	Permuta los tres números superiores en el orden descrito
ROLL	-1,*	takes a number n, then permutes the top n+1 numbers so that the n+1th becomes the top and all of the others move down one
PEEK	-1,+1	Mini Maestro 12, 18, y 24 solo) takes a number n, then copies the nth value on the stack (medido desde abajo) to the top of the stack
POKE	-2,+1	Mini Maestro 12, 18, y 24 solo) takes a number n, then removes the next value from the stack and puts it at the nth location on the stack (medido desde abajo)

Comandos matemáticos (unitarios)

Estos comandos cogen un argumento simple de la parte superior de la pila y devuelven un valor simple como resultado. Algunos tienen equivalencia en C (y en otros lenguajes), como vemos en la columna “en C”. Usamos “falso” equivale a 0 y “verdadero” equivale a cualquier valor que no lo sea. El comando “verdadero” siempre devuelve 1.

Comando	Equivale en C	descripción
BITWISE_NOT	~	Invierte todos los bits del argumento
LOGICAL_NOT	!	Reemplaza verdadero con falso, falso con verdadero
NEGATE	–	Reemplaza x por -x
POSITIVE	nada	Verdadero solo si el argumento es mayor de 0
NEGATIVE	nada	Verdadero solo si el argumento es menor de 0
NONZERO	nada	Verdadero (1) solo si el argumento es “no-cero”

Comandos matemáticos (binarios)

Estos comandos cogen dos argumentos de la parte superior de la pila y devuelven uno como resultado. El orden de los argumentos cuando importa son los standard en matemáticas; por ejemplo 1-2 escribirás “1 2 MINUS”. Estos comandos tienen su equivalencia en C (y en otros muchos lenguajes), lee la columna “C equivalent”.

Comando	Equivale en C	descripción
BITWISE_AND	&	Aplica la función AND a los bits escogidos del argumento.
BITWISE_OR	|	Aplica la función OR a los bits escogidos del argumento
BITWISE_XOR	^	Aplica la función XOR a los escogidos bits del argumento
DIVIDE	/	división
EQUALS	=	Verdadero solo si los argumentos son iguales
GREATER_THAN	>	Verdadero solo si el primer argumento es mayor que el segundo
LESS_THAN	<	Verdadero solo si el primer argumento es menor que el segundo
LOGICAL_AND	&&	Verdadero si ambos argumentos son verdaderos
LOGICAL_OR	||	Verdadero si solo uno de los argumentos es verdadero
MAX	nada	Selecciona el mayor de los dos argumentos
MIN	nada	Selecciona el menor de los dos argumentos
MINUS	–	Resta
MOD	%	Resto de división
NOT_EQUALS	!=	Verdadero si los argumentos NO son iguales
PLUS	+	Suma
SHIFT_LEFT	<<	Desplazamiento de bits a la izquierda (sin ajuste)
SHIFT_RIGHT	>>	Desplazamiento de bits a la derecha (sin ajuste)
TIMES	*	Multiplicar
Salidas (servo, led,.)	efecto	descripción
SPEED	-2	Ajusta la velocidad del canal especificado por el elemento superior al valor del segundo elemento (ver sección 4.e)
ACCELERATION	-2	Ajusta la aceleración del canal especificado por el elemento superior al valor del segundo elemento (ver sección 4.e)
GET_POSITION	-1,+1	Coge la *posición* del canal especificado por el elemento superior.
GET_MOVING_STATE	+1	Verdadero si cualquiera de los servos se esta moviendo limitado por la velocidad y aceleración
SERVO	-2	Ajusta el destino del canal especificado por el elemento superior al valor del segundo elemento (ver sección 4.e).Unidades 0.25 µs
SERVO_8BIT	-2	Ajusta el destino del canal especificado por el elemento superior al valor del segundo elemento (ver sección 5.e) que puede tener valores de 0 a 254 (ver Mini SSC Comando en sección 5.e)
LED_ON	nada	Enciende el LED rojo
LED_OFF	nada	Apaga el LED rojo (asumiendo que no hay bits de error)
PWM	-2	Mini Maestro 12, 18, y 24 solo) habilita la salida PWM, con el periodo del elemento superior y el pulso del siguiente elemento, en unidades de 1/48 µs (ver sección 4.a para más información)
SERIAL_SEND_BYTE	-1	Mini Maestro 12, 18, y 24 solo) saca el numero superior de la pila, lo interpreta como un byte y lo envía TTL por la línea (TX)

6.c. Ejemplo de Scripts

Parpadeo de LED

El siguiente script provoca que el LED rojo de Maestro parpadee cada segundo:

```
# Parpadeo del LED rojo cada segundo.
begin
  led_on
  100 delay
  led_off
  900 delay
repeat
```

Es buena idea repasar el script antes de seguir. En particular, preste atención a cómo el comando "100" pone el número 100 en la pila y el comando DELAY consume ese número. En el lenguaje de scripts de Maestro, los argumentos siempre tienen colocarse en la pila antes que los comandos que los utilizan, por lo que parece que vaya al revés de otros lenguajes. También nos indica que puedes organizar el código de maneras diferentes. Por ejemplo, este programa es equivalente a la anterior:

```
# Blinks the red LED once per second.
begin
  900 100
  led_on delay
  led_off delay
repeat
```

Los números se meten en la pila al principio del bucle, cuando se consumen más tarde. Presta atención en el orden empleado aqui: el 900 entra en la pila primero y se usa el último.

Una simple secuencia de servo

El siguiente script muestra como direccionar el servo 0 a cinco posiciones diferentes en un bucle.

```
  # Mover servo 0 a cinco posiciones diferentes, en el bucle.
  begin
    4000 0 servo # ajusta servo 0 a 1.00 ms
    500 delay
    5000 0 servo # ajusta a 1.25 ms
    500 delay
    6000 0 servo # ajusta a 1.50 ms
    500 delay
    7000 0 servo # ajmusta a 1.75 ms
    500 delay
    8000 0 servo # ajusta a 2.00 ms
    500 delay
  repeat
```

El modo serie no debe estar configurado para detectar la velocidad de transmisión en esta secuencia de comandos. En ese modo Maestro no permite ninguna de las salidas de los servos hasta que el byte de inicio ha sido recibido.

Tenga en cuenta que la posición de los servos se especifica en unidades de 0,25 us, por lo que un valor de 4000 corresponde a 1 ms. Los comentarios son esenciales para programas complicados. Es importante tener en cuenta los comandos DELAY, sin los cuales, el script no espera a entre todos los comandos ejecutándose el bucle cientos de veces por segundo.

Comprender **la secuencia**

El programa anterior tiene 58 bytes de espacio de programa: 11 bytes para cada posición del servo y 3 para el bucle. A este ritmo, podríamos almacenar hasta 92 posiciones en la memoria de 1024 bytes de Micro Maestro, o 744 posiciones la memoria de 8192 bytes de los Mini-Maestros.

Para obtener el máximo rendimiento de la memoria que es limitada, hay una gran variedad de formas de comprimir el programa. Lo más importante es hacer uso de subrutinas. Por ejemplo, ya que se repiten las instrucciones de "0 servo 500 delay" en varias ocasiones, podemos pasar a una subrutina para ahorrar espacio. Al mismo tiempo, esto simplifica el código y hace que sea más fácil hacer las futuras modificaciones, como cambiar la velocidad de la secuencia completa.

```
  # Move servo 0 to five different positions, in a loop.
  begin
    4000
    frame
```

```
    5000
    frame
    6000
    frame
    7000
    frame
    8000
    frame
  repeat

  sub frame
    0 servo
    500 delay
    return
```

Usando la subrutina recortamos el script 31 bytes: 4 por la posición y 11 bytes por el bucle y la definición de FRAME. Podemos ir más allá: al inspeccionar el código compilado nos muestra que poner cada número en la pila requiere 3 bytes: un byte como comando y dos para el número de dos bytes. Números del 0 al 255 pueden cargarse en la pila con sólo dos bytes. Supongamos que en nuestra aplicación no necesitamos la resolución completa 0.25 µs del dispositivo ya que todos nuestros valores son múltiplos de 100. Entonces podemos usar números más pequeños para guardar en un byte:

```
  # Move servo 0 to five different positions, in a loop.
  begin
    40 frame
    50 frame
    60 frame
    70 frame
    80 frame
  repeat
  # loads a frame specified in 25 us units
  sub frame
    100 times
    0 servo
    500 delay
    return
```

Este programa tiene 29 bytes, con 3 bytes para la posición y 14 para la cabecera. Tenemos la misma eficacia que usando el comando SERVO_8BIT, que necesita un byte de argumento de 0 a 254. Podemos rebajar más poniendo todos los números juntos:

```
  # Move servo 0 to five different positions, in a loop.
  begin
    80 70 60 50 40
    frame frame frame frame frame
  repeat
  # loads a frame specified in 25 us units
  sub frame
    100 times
    0 servo
    500 delay
    return
```

Si paso a través de esta versión del programa, también se dará cuenta que los cinco números se colocan en la pila en un solo paso, esto es porque el compilador puede utilizar un comando único para poner varios números en la pila. Utilizando un único comando para varios números ahorra espacio, ahora estamos a sólo 26 bytes. Sólo 12 bytes se usan para los 5 frames, con una media de de 2,4 bytes para cada. Esto es probablemente lo suficientemente compacto – duplicando esta estructura podríamos ajustar hasta 420 posiciones diferentes en la memoria de 1024 bytes de Micro Maestro, sin embargo, el código puede ser aún más pequeño.

Fijate en esta secuencia que utiliza 31 frames para hacer un movimiento suave de ida y vuelta:

```
# Moves servo in a sine wave between 1 y 2 ms.
begin
  60 64 68 71 74 77 79 80 80 79 78 76 73 70 66 62
```

```
  58 54 50 47 44 42 41 40 40 41 43 46 49 52 56
  all_frames
repeat
sub all_frames
  begin
    depth
  while
    100 times
    0 servo
    100 delay
  repeat
  return
```

En este código hemos escrito FRAME como subrutina usando el comando DEPTH que automáticamente las lee desde la pila mientras no trabaja. Este programa usa 34 bytes para almacenar 31 frames con una media de 1,2 bytes por frame. Puedes almacenar una secuencia con 900 posiciones diferentes en memoria usando el rey del script.

Secuencias suaves con GET_MOVING_STATE

Velocidad y aceleración pueden usarse para hacer las secuencias suaves. Sin embargo, un problema común es que no sabes la cantidad de retardo entre frames para permitir que el servo pueda alcanzar su posición final. Un ejemplo de cómo utilizar la función integrada de GET_MOVING_STATE para hacer una secuencia de suave movimiento, en lugar de usar DELAY.

```
  # This example uses speed y acceleration to make a smooth
  # motion back y forth between 1 y 2 ms.
  3 0 acceleration
  30 0 speed
  begin
    4000 0 servo # set servo 0 to 1.00 ms
    moving_wait
    8000 0 servo # 2.00 ms
    moving_wait
  repeat
  sub moving_wait
    begin
      get_moving_state
    while
      # wait until it is no longer moving
    repeat
    return
```

GET_MOVING_STATE devuelve un 1, siempre y cuando haya al menos un servo en movimiento, lo que se puede utilizar cuando quieras esperar a que todos los movimiento se detengan antes de proceder al siguiente paso del script.

Entrada analógica para controlar servos

Una característica importante del maestro es que puede ser usado para leer entradas de sensores, interruptores y otros dispositivos. Como ejemplo, supongamos que queremos usar un potenciómetro para controlar la posición de un servo. Conectar un potenciómetro para formar un divisor de tensión entre 5 V y 0, con la toma central conectada al canal 1. Configurar el canal 1 como entrada, y examinar la señal en la pestaña `Status` del Centro de Control Maestro. Debes ver el indicador de posición que varía de 0 hasta 255 us cuando giras el potenciómetro de un lado a otro. En el script, este rango corresponde a los números de 0 a 1023. Podemos ampliar este número hasta el rango del servo, cuando la posición llegue a este numero, todo en el bucle:

```
 # Ajusta servo 0 a la posición basada en una entrada analógica.
 begin
   1 get_position  # get the value of the pot, 0-1023
   4 times 4000 plus # scale it to 4000-8092, approximately 1-2 ms
   0 servo      # set servo 0 based to the value
 repeat
```

Por otra parte, tal vez desees que el servo vaya otras posiciones en función del valor de entrada:

```
  # Set the servo to 4000, 6000, or 8000 depending on an analog input.
  begin
```

```
    1 get_position  # get the value of the pot, 0-1023
    dup 300 less_than
    if
      4000   # go to 4000 for values 0-299
    else
      dup 600 less_than
      if
        6000 # go to 6000 for values 300-599
      else
        8000 # go to 8000 for values 600-1023
      endif
    endif
    0 servo
    drop      # remove the original copy of the pot value
  repeat
```

En el ejemplo anterior funciona, pero cuando el potenciómetro está cerca de 300 o 600, las interferencias en la conversión AD pueden causar que el servo vibre. Una mejor manera de hacerlo es con histéresis:

```
# Ajusta el servo a 4000,6000,8000 según la entrada analógica con hysteresis.
begin
  4000 0 300 servo_range
  6000 300 600 servo_range
  8000 600 1023 servo_range
repeat
# usage: <pos> <low> <high> servo_range
# If the pot is in the range specified by low y high,
# keeps servo 0 at pos until the pot moves out of this
# range, with hysteresis.
sub servo_range
  pot 2 pick less_than logical_not     # >= low
  pot 2 pick greater_than logical_not # <= high
  logical_and
  if
    begin
      pot 2 pick 10 minus less_than logical_not    # >= low - 10
      pot 2 pick 10 plus greater_than logical_not # <= high + 10
      logical_and
    while
      2 pick 0 servo
    repeat
  endif
  drop drop drop
  return
sub pot
  1 get_position
  return
```

En este ejemplo se utiliza un rango para decidir a dónde ir cuando hace una transición y luego espera a que el servo haga un rango un poco más grande antes de hacer otra transición. Siempre y cuando la diferencia (10 en este ejemplo) sea mayor que la cantidad de ruido, esto evitará los saltos. Tenga en cuenta que este ejemplo sólo funciona si se conecta el potenciómetro a uno de los canales de entrada analógicos capaces (canales 0-11). Las entradas en los otros canales son digitales.

Uso de un pulsador o de un interruptor para controlar servos

Es posible conectar un pulsador o un interruptor al Maestro y detectar su estado en el script. El script siguiente mueve un servo a través de una secuencia predeterminada de movimientos, avanzando a la siguiente etapa cada vez que pulsamos. Utilizamos el canal 0 para el pulsador y el canal 1 para el servo. El canal del pulsador debe estar configurado como entrada y conectado correctamente. ver la sección 7.b para las instrucciones de cómo realizar la conexión con una resistencia de pull-up, con lo que la entrada estará en alto y al pulsar pasará a nivel bajo.

```
  goto main_loop    # Run the main loop when the script starts (see below).
  # Esta rutina devuelve 1 si pulsas, 0 si no.
  # Para covertir el valor de entrada (0-1023) a valor digital (0 o 1) que
```

```
# representa el estado del pulsador hacemos una comparación arbitraria (500).
# La subrutina pone el valor logico 1 o a 0 en la pila, según pulsemos.
sub button
  0 get_position 500 less_than
  return
# Esta subrutina usa la anterior para esperar la pulsación, incluyendo
# un pequeño retardo para eliminar rebotes de la entrada.
sub wait_for_button_press
  wait_for_button_open_10ms
  wait_for_button_closed_10ms
  return
# Espera la NO pulsación durante 10 ms.
sub wait_for_button_open_10ms
  get_ms # put the current time on the stack
  begin
    # reset the time on the stack if it is pressed
    button
    if
      drop get_ms
    else
      get_ms over minus 10 greater_than
      if drop return endif
    endif
  repeat
# Espera la pulsación durante 10 ms.
sub wait_for_button_closed_10ms
  get_ms
  begin
    # reset the time on the stack if it is not pressed
    button
    if
      get_ms over minus 10 greater_than
      if drop return endif
    else
      drop get_ms
    endif
  repeat
```

Como usar wait_for_button_press

```
# Usa WAIT_FOR_BUTTON_PRESS para permitir el paso a paso
# de secuencias en cada pulsación para posicionar el servo 1
main_loop:
begin
  4000 frame
  5000 frame
  6000 frame
  7000 frame
  8000 frame
repeat
sub frame
  wait_for_button_press
  1 servo
  return
```

Igual que el ejemplo anterior, el script de pasos a través de una secuencia de frames, pero en lugar de un retardo entre frames aqui espera sólo a pulsar un botón.

La subrutina WAIT_FOR_BUTTON_PRESS se puede utilizar en una variedad de secuencias con comandos diferentes, cada vez que tengas que esperar la pulsación. También se podría ampliar este ejemplo para permitir que varios pulsadores o quizás otros tipos de control con el pulsador.

Utilización de varios pulsadores o interruptores para control de servos

Este script muestra cómo conectar el Maestro a varios pulsadores. Cuando se pulsa se ejecuta la secuencia correspondiente.

```
# When the script is not doing anything else,
# this loop will listen for button presses.  When a button
```

```
# is pressed it runs the corresponding sequence.
begin
  button_a if sequence_a endif
  button_b if sequence_b endif
  button_c if sequence_c endif
repeat
# These subroutines each return 1 if the corresponding
# button is pressed, y return 0 otherwise.
# Currently button_a is assigned to channel 0,
# button_b is assigned to channel 1, and
# button_c is assigned to channel 2.
# These channels must be configured as Inputs in the
# Channel Settings tab.
sub button_a
  0 get_position 500 less_than
  return
sub button_b
  1 get_position 500 less_than
  return
sub button_c
  2 get_position 500 less_than
  return
# These subroutines each perform an arbitrary sequence
# of servo movements.  You should change these to fit
# your application.
sub sequence_a
  4000 3 servo 1000 delay
  6000 3 servo 500 delay
  return
sub sequence_b
  8000 4 servo 900 delay
  7000 4 servo 900 delay
  6000 4 servo 900 delay
  return
sub sequence_c
  10 4 speed
  7000 4 servo 3000 delay
  6000 4 servo 3000 delay
  return
```

Tenga en cuenta que este script no hace multitarea. Si una secuencia se ejecuta, el script no detecta otras pulsaciones hasta que la secuencia se termina. Es posible hacer que los pulsadores funcionen de forma independiente, pero el script tendría que ser mucho más complicado.

Dependiendo de cómo trabajes con la escritura de scripts es posible que prefieras usar varios controladores Maestro en su lugar.

Retardos largos

Los retardos largos son posibles con el comando DELAY hasta unos 32 segundos. En algunos casos puedes necesitar retardos mucho más largos. Aquí tienes un ejemplo para muchos segundos e minutos:

```
# Mover servo 0 adelante y atrás, con retardos de 10 minutes entre movimientos.
begin
  4000 0 servo
  10 delay_minutes
  8000 0 servo
  10 delay_minutes
repeat
# delay by a specified number of seconds, up to 65535 s
sub delay_seconds
  begin dup while       # check if the count has reached zero
    1 minus 1000 delay # subtract one y delay 1s
  repeat
  drop return            # remove the 0 from the stack y return
# delay by a specified number of minutes, up to 65535 min
```

```
sub delay_minutes
  begin dup while
    1 minus 60 delay_seconds # subtract one y delay 1min
  repeat
  drop return                    # remove the 0 from the stack y return
```

Es fácil escribir subrutinas para retardos de horas, días, semanas, o lo que quieras. Tenga en cuenta, sin embargo que el temporizador del Micro Maestro no es tan preciso como un cronómetro - estos retrasos fallar fácilmente en un 1%.

Salida digital

Conexión del Micro Maestro a una cadena de ShiftBars. Una sola alimentación de 12V para todos los dispositivos.

La función de salida digital del Maestro es capaz de controlar cualquier cosa, desde circuitos simples a dispositivos inteligentes, como los módulos ShiftBrite o ShiftBar. Son módulos controladores de leds que utilizan un simple protocolo serie síncrono. En este ejemplo, los pins del ShiftBrite, el clock, el latch, y datos están conectadas a los canales 0, 1, y 2, respectivamente, y estos canales estén configurados como salidas. La subrutina RGB definida aquí toma valores de 10-bits de la pila para los colores rojo, verde, azul, luego envía un paquete color de 32-bytes que cambia el latch para actualizar el ShiftBrite con el valor nuevo recibido. La subrutina puede ser modificada para el control de una cadena más grande de ShiftBrites si lo deseas.

```
begin
  1023    0    0 rgb  500 delay  # red
     0 1023    0 rgb  500 delay  # green
     0    0 1023 rgb  500 delay  # blue
repeat
# Subroutine for setting the RGB value of a ShiftBrite/ShiftBar.
# example usage: 1023 511 255 rgb
sub rgb
  0 send_bit # this bit does not matter
  0 send_bit # the "address" bit - 0 means a color Comando
  swap rot rot
  send_10_bit_value
  send_10_bit_value
  send_10_bit_value
  0 1 8000 1 servo servo # toggle the latch pin
  return
# sends a numerical value as a sequence of 10 bits
sub send_10_bit_value
  512
  begin
    dup
  while
    over over bitwise_and send_bit
    1 shift_right
  repeat
  drop drop
  return
# sends a single bit
sub send_bit
  if 8000 else 0 endif
  2 servo                # set DATA to 0 or 1
  0 0 8000 0 servo servo # toggle CLOCK
  return
```

Tenga en cuenta que se utiliza 0 para establecer la salida en baja y 8000 para establecer el alto. Estas son opciones razonables, pero cualquier valor desde 0 hasta 5999 podría ser utilizado para como bajo y desde 6000 hasta 32767 podría ser utilizado para alto, si lo desea.

Salida serie (Mini Maestro 12, 18, y 24 solamente)

En Mini-Maestro Mini 12, 18, y 24, un script puede ser usado para enviar datos serie por el puerto TTL serie (línea RX). Esto significa que Maestro puede controlar Maestros adicionales, lo que permite un gran número de canales sin necesidad de un microcontrolador aparte. Aquí tienes un sencillo programa que muestra cómo un comando se puede utilizar para controlar a otro Maestro.

Para utilizar este código, configura los Maestros en modo UART con la misma velocidad de transmisión y conectados a la línea TX del master con la RX del esclavo.

```
100 delay # initial delay to make sure that the other maestro has time to
initialize
begin
  127 0 mini_ssc # set servo 0 to position 127, using the mini-SSC Comando
  254 0 mini_ssc # set servo 0 to position 254
repeat
sub mini_ssc
  0xFF serial_send_byte serial_send_byte serial_send_byte
  return
```

6.d. Especificaciones para los scripts

El lenguaje de scripts de Mini Maestros esta disponible también para Micro Maestro. Las diferencias las vemos en la tabla siguiente.

	Micro Maestro	Mini Maestro 12, 18, y 24
Script tamaño	1KB	8 KB
Stack tamaño	32	126
Call stack tamaño	10	126
Numero de subrutinas	128	Sin limite
Comandos extras:		PEEK, POKE, PWM, SERIAL_SEND_BYTE

Script tamaño: Es el numero de bytes en memoria que se usa para almacenar código. Un script grande permite almacenar muchas frames y escribir complejos programas.

Stack tamaño: Es el máximo número de valores que pueden meterse en la pila al mismo tiempo. Una pila más grande permite tener más variables, hacer cálculos más grandes, y preocuparse menos de su desborde.

Call stack tamaño: Es el máximo de anidamientos de subrutinas (máximo de rellamadas).

Número de subrutinas: Es el número de subrutinas que puedes tener. Las primeras 128 subrutinas Del script son especiales; estas pueden iniciarse usando comandos USB nativos y cada llamada desde dentro de la secuencia de comandos solo requiere un byte de espacio. Micro Maestro solo soporta 128 subrutinas. Mini Maestro 12,18 y 24 soportan un número ilimitado pero las definidas después de la 128 no pueden iniciarse desde USB y cada llamada requiere 3 bytes en el script.

7. Ejemplos de circuitos

Esta sección contiene ejemplos de circuitos para Maestro para mostrar diferentes formas en que se puede conectar a tu proyecto.

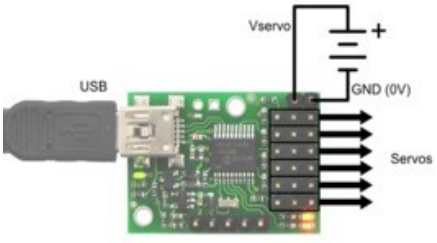
Aunque muchas de las imágenes sólo muestran el Micro Maestro, la información se puede aplicar a todos los modelos (a menos que se indique lo contrario).

7.a. Alimentación de Maestro

Hay varias maneras de alimentar a Maestro y controlar los servos.

Alimentación desde USB

Si se conecta una fuente de alimentación para el servo en los terminales de conexión y el Maestro a USB como se muestra en la imagen de la derecha, entonces el procesador del Maestro será alimentado por USB, mientras que los servos por la fuente de alimentación.

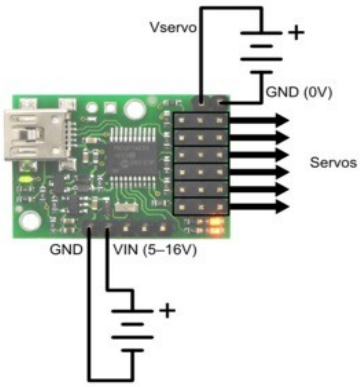


El procesador de Micro Maestro puede alimentarse desde USB mientras los servos por separado..

La alimentación debe tener una tensión dentro del rango de los servos capaz de suministrar toda la corriente que los servos necesiten.
En esta configuración, si el ordenador (u otro host USB) al que el Maestro está conectado se pone en reposo, de forma predeterminada el Maestro también y deja de enviar pulsos a los servos. Si necesitas mover los servos, mientras el ordenador está apagado, puedes utilizar “ignorar suspensión USB” en la ficha `Serial Settings` del Centro de Control Maestro. Tenga en cuenta que esto sólo funcionará si el equipo suministra energía al puerto USB mientras está en reposo a no ser que Maestro no sea compatible con USB.

Dos alimentaciones

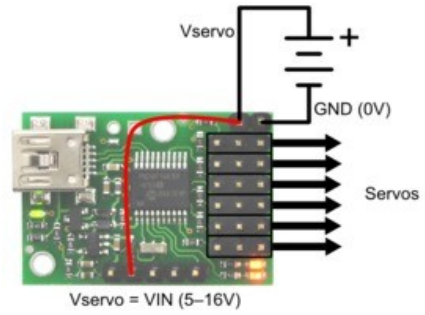
Si conectas la alimentación al terminal de servos y conectas otra alimentación a GND/VIN para el procesador de Maestro, debes asegurarte de que VIN tiene de 5–16V y es capaz de suministrar al menos 30 mA al Micro Maestro o 50 mA a Mini Maestro. La alimentación de los servos deberá estar en los valores correctos y con la potencia suficiente para que puedan moverse.



El procesador del Micro Maestro y los servos alimentados con dos tensiones.

Una sola alimentación

Si conectas todo con una sola alimentación en VIN debes unirla al terminal de alimentación de los servos Debe estar entre 5-16V y tener la fuerza suficiente para que los respectivos servos puedan trabajar en sus rangos y con la potencia necesaria para ellos.



El procesador del Micro Maestro y los servos pueden alimentarse con 5–16V conectando el positivo del servo a VIN.

En Micro Maestro de 6 canales, una manera de hacerlo es circuitando según el dibujo ambas tomas de alimentación. En los Mini Maestro 12, 18 y 24 recomendamos hacer la conexión a través del puente dedicado que hay en el lateral de la placa usando el jumper azul para conectar los pins etiquetados como “VSRV=VIN”.

7.b. Conexión de servos y Periféricos

En Maestro, cualquiera de los canales puede ser utilizado como salida de pulso para un servo RC, como entrada analógica/digital, o como salida digital. Esto permite a Maestro el control de servos, lectura de pulsadores, de posiciones de un potenciómetro, control de leds y más. Los canales pueden ser controlados desde el script de usuario en el maestro o externamente en TTL de serie o USB.

Servo

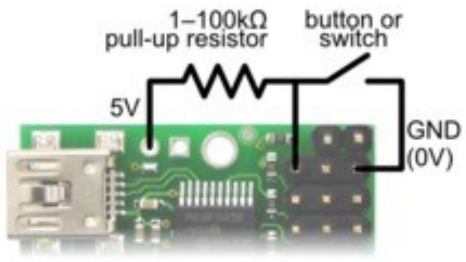
Para conectar un servo al Maestro, primero debes decidir que canal vas a utilizar. Si no está configurado para modo servo (por defecto) en el Centro de Control Maestro, en la pestaña `Chanel settings`, debes cambiarlo a modo *servo* y clic en "Aplicar". Conecta los cables del servo al canal, con cuidado de no conectarlos al revés o podría destruirlo. El cable de señal (blanco, naranja o amarillo) debe ir hacia el interior de la placa y el de tierra (negro o marrón) debe estar más cerca del borde. Deberías conectar una fuente de alimentación de CC para dar corriente a los servos (mira en sección 7.a para las formas de alimentación).
Ahora puedes probar el servo estableciendo el destino del canal en la pestaña `Status` del Centro de Control Maestro. Si está activado para servo con la casilla de verificación habilitada debe ser capaz de mover el servo al arrastrar la barra deslizante a izquierda y derecha. Ahora ya puedes controlar los servos con un script mediante el comando SERVO, o usando el comando "Set Target". Estos comandos usan argumentos en unidades de cuarto de microsegundo, por lo que se deben multiplicar los valores objetivo que se ven en la ficha Status por cuatro para calcular los argumentos correctos.
Hay comandos más avanzadas que también están disponibles para el control de servos.

Pulsador o Interruptor

Para realizar la conexión de estos dispositivos, primero debes decidir el canal a usar y configurarlo en la pestaña `Channel Settings`, con ponerlo en modo *input* y clic en “Aplicar”. después intercala

una resistencia pull-up (1–100k) entre la línea de señal y 5V para que la entrada esté en alto (5V) cuando no se utiliza. Pon el dispositivo entre la línea de señal y GND (0V) para que al activar la entrada se ponga en 0V. El dibujo muestra como hacerlo en un Micro Maestro de seis canales.

Nota: Las resistencias externas de pull-up resistor no son necesarias si usas los canales 18, 19 o 20 en Mini Maestro 24 ya que están pueden ser habilitadas internamente en la pestaña `Channel Settings` del Maestro Control Center.

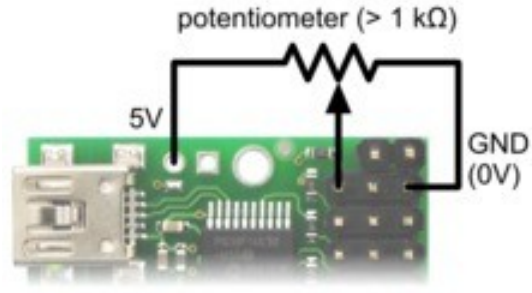


Conexión de un pulsador o un interruptor al Micro Maestro .

Puedes testear la entrada activando el pulsador o el interruptor y verificando la variable *position* que se muestra en la pestaña Status del Maestro Control Center que refleja el estado del dispositivo; estará cerca de 255,75 sin pulsar y 0 si se actúa sobre el. Ahora ya podrás leer el estado del dispositivo en tu script con el comando `GET_POSITION` o con la "`Get Position`". Estos comandos devuelven valores que están entre 1023 si no se actúa y de 0 cuando se activa.

Potenciómetro

Para conectar un potenciómetro primero decidir el canal a utilizar. Si tienes el Mini Maestro 18 o 24 canales asegúrate de escoger entre uno de los canales de entrada analógicos (canales 0-11). En el Centro de Control Maestro, en la pestaña `Channel Settings`, cambia el modo de entrada del canal y clic en "Aplicar". Luego, conecta el potenciómetro al Maestro para que los extremos vayan a GND y 5 V, y el centro a la línea de señal. La imagen muestra cómo hacer para conectar un potenciómetro en el canal 0 del Micro Maestro Micro de 6 canales. El potenciómetro debe tener una resistencia de al menos 1k para que no absorba mucha corriente de la línea de 5V.

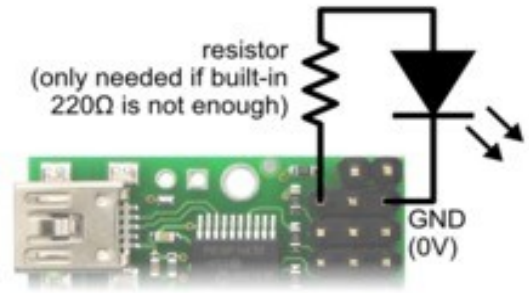


Conexión de un potenciómetro a un Micro Maestro.

Puedes probar girando el potenciómetro y ver que la *Position* varia como se muestra en la ficha `Status` del Centro de Control Maestro y refleja el valor del potenciómetro: aproximadamente debe variar entre 255,75 y 0. Ahora puedes leer la posición del potenciómetro en un script con el comando `GET_POSITION` o con el comando serie "`Get Position`". Estos comandos devolverán valores aproximados entre 1023 y 0.

Led

Para la conexión de un LED al Maestro después de decidir que canal se va a utilizar debes cambiar el modo del canal a *Output* en la pestaña `Channel Settings` del Maestro Control Center y clic “Aplicar ajustes”. después conecta el CÁTODO del led a GND (cualquier toma de tierra del la placa nos vale) y el ÁNODO a la línea de señal con una resistencia de 220 ohmios para su protección, lo que significa que puedes conectar la mayoría de leds directamente a la línea, sin embargo, lee la ficha técnica del LED para asegurarte, y añadir su propia resistencia, si es necesario.



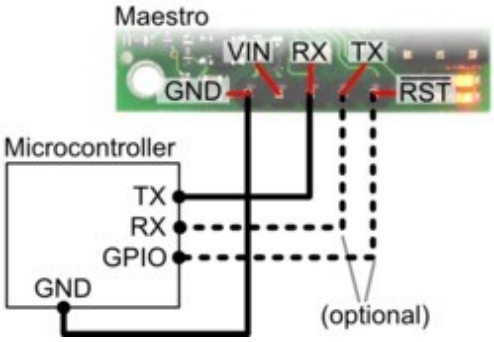
Conexión de un LED al Micro Maestro

Prueba el LED mediante el ajuste del “destino” en la pestaña Status del Maestro Control Center. El LED se encenderá si se establece el objetivo para ser mayor o igual a 1500 ms y apagado en otro caso. Puedes controlar el LED en el script con el comando SERVO o en comunicación serie con el comando "Set Target". Estos comandos usan argumentos en unidades de un cuarto de microsegundos, por lo que el LED debe encenderse si se envía un número mayor o igual a 6000 (1500x4=6000) y apagarse en caso contrario. La potencia en las líneas de señal es limitada. Ver Sección 1.a Sección 1.b para más información).

7.c. Conectar un microcontrolador

Maestro acepta comando serie TTL de un microcontrolador. Para conectarlo debes primero conectar la línea GND del micro con la correspondiente del Maestro. Cuando conectes la línea TX (transmisión serie) del microcontrolador a la línea RX del Maestro el micro ya podrá enviar comandos. Si necesitas recibir respuestas del Maestro entonces hacemos la conexión entre la línea RX (recepción serie) del microcontrolador y la línea TX del Maestro.

Para más información sobre comunicaciones consulta la sección 5.



Conexión Micro Maestro y microcontrolador.

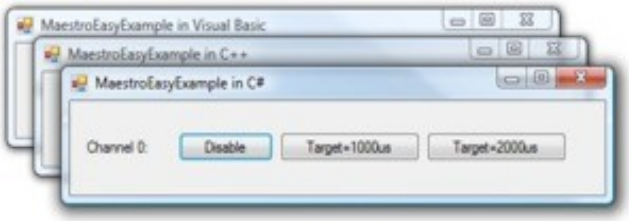
Para que tu controlador pueda resetear el Maestro debes de conectar la línea RST de mismo a una línea cualquiera de entrada/salida del micro. Debería ser una línea I/O tres estados o 5V cuando quieras que el Maestro funcione y que pase a bajo (0V) temporalmente al provocar un reset. Si quieres que el microcontrolador tenga capacidad para detectar los errores del Maestro por una entrada digital en lugar de comandos serie o si deseas recibir información directa del script conecta la línea ERR de Maestro a cualquier línea I/O del microcontrolador. La línea ERR está disponible solo en el Maestro Mini 12 , 18, 24 canales. Ver 1.b sección para más información de la línea ERR.

8. Escribir software para el control de Maestro

Hay dos maneras de escribir software para el control de Maestro: la interfaz nativa USB y el puerto serie virtual. La interfaz USB dispone de muchas características y permite configurar parámetros y seleccionar el maestro según su numero de serie. También con USB se puede recuperar más eficazmente frente a desconexiones temporales. El puerto serie virtual es fácil de usar si no estás familiarizado con la programación y funciona con software que use este puerto como LabView.

Native USB Interface

El Pololu USB Software Development Kit soporta Windows y Linux e incluye código fuente para:



La Pololu USB SDK contiene ejemplo de código para Maestro en C#, Visual C++, y Visual Basic .NET.

- *MaestroEasyExample:* Una aplicación simple que usa USB con tres pulsadores de control para el Maestro. La versión de este ejemplo esta en C#, Visual Basic .NET, y Visual C++.
- *MaestroAdvancedExample:* es un ejemplo de aplicación para USB que envía comandos y recibe desde Maestro y se recupera automáticamente en caso de desconexión (escrito en C#).
- *UscCmd:* Utilidad para comandos en línea para configurara y controlar Maestro (escrito en C#).
- *C# .NET* librerías que habilitan la comunicación USB con Maestro (escrito en C#).

Puedes modificar las aplicaciones en el SDK según tus necesidades o puede utilizar las librerías para integrar al Maestro en tus propias aplicaciones.

Virtual Serial Ports

Casi cualquier lenguaje de programación puede acceder a los puertos COM creado por Maestro. Se recomienda el Microsoft. NET Framework, que es libre de usar y contiene una clase SerialPort que hace que sea fácil leer y escribir bytes por un puerto serie. Puedes descargar Visual Studio Express (para C #, C + +, o Visual Basic) y escribir programas que utilizan la clase SerialPort para comunicar con Maestro. Configura modo serie de Maestro para "USB Dual Port" o "USB Chained"

9. Maestro limitaciones de ajustes

Limitaciones en las velocidades en baudios

En Mini Maestro 12, 18, y 24, las velocidades siguientes no deben excederse o el procesador puede

Modo serie	10–100 Hz	111–250 Hz	333 Hz
UART/USB encadenado*	200 kbps	115.2 kbps	115.2 kbps
Dual-port	115.2 kbps	57.6 kbps	57.6 kbps

sobresaturarse y perder efectividad.

- Asumiendo que los bytes no se envían y reciben simultáneamente a requerimiento del protocolo Pololu.

Mini Maestro. Limitaciones en la longitud del pulso (high pulse rates solo).

	6 servos	5 servos	4 servos	1–3 servos
67 Hz	2448	2944	3000+	3000+
71 Hz	2272	2736	3000+	3000+
77 Hz	2112	2544	3000+	3000+
83 Hz	1936	2336	2944	3000+
91 Hz	1776	2144	2688	3000+
100 Hz	1616	1936	2448	3000+

Muchos servos se han designado para operaciones a 50 Hz. Si no vas a usar pulsos altos puedes ignorar esta sección.

Micro Maestro tiene restringida la longitud del pulso a un máximo de 67 Hz. La restricción depende del ajuste en *Servos available* y se aplica de forma automática por el Maestro Control Center.

Las longitudes de pulso máximas en microsegundos son:

Mini Maestro. Limitaciones en la longitud del pulso (high pulse rates solo).

En Mini Maestro 12, 18 y 2 las tasas de pulso de 200–333 Hz ponen restricciones en el ancho de pulso del servo. Estas restricciones se aplican a todos los canales, incluso si algunos tienen una tasa diferente por la función de *Period multiplier*. Las tablas muestran longitudes de pulso máximo y mínimo permitido en ms, para una variedad de combinaciones de números de pulso tasas y servo. Los servos habilitados siempre deben satisfacer las restricciones sobre alguna fila. Configura los máximos y mínimos para canales específicos de acuerdo con estas. No es necesario especificar los intervalos de los servo de antemano: puedes apagar algunos canales o ajustar sus posiciones para tener acceso a una gama más amplia de otros canales. Si la configuración puede violar estas restricciones, el período puede aumentar y los límites de velocidad y aceleración cambian en consecuencia, pero no afecta al funcionamiento del Maestro...

Así con Mini Maestro 24 a 250 Hz (4 ms periodo), puedes usar 12 servos con un rango de 576–2880 µs y 6 servos con un rango de 64–2880 µs. Los seis canales restantes pueden ponerse en off o como **Input** u **Output** (no como **Servo**).

	Pulsos a 333 Hz		Pulsos a 250 Hz		Pulsos a 200 Hz	
Servos	Min	Max	Min	Max	Min	Max
6	64	2328	64	3000+	64	3000+
6	192	2648				
12	64	1752	64	2752	64	3000+
12	384	2456	384	3000+		
6/6	384/64	2072	384/64	3000+	384/64	3000+
18	64	1176	64	2176	64	3000+
18	576	2264	576	3000+		
12/6			576/64	2880	576/64	3000+
24	768	2072	64	1600	64	2600
24			768	3000+	768	3000+
18/6			768/64	2688	768/64	3000+