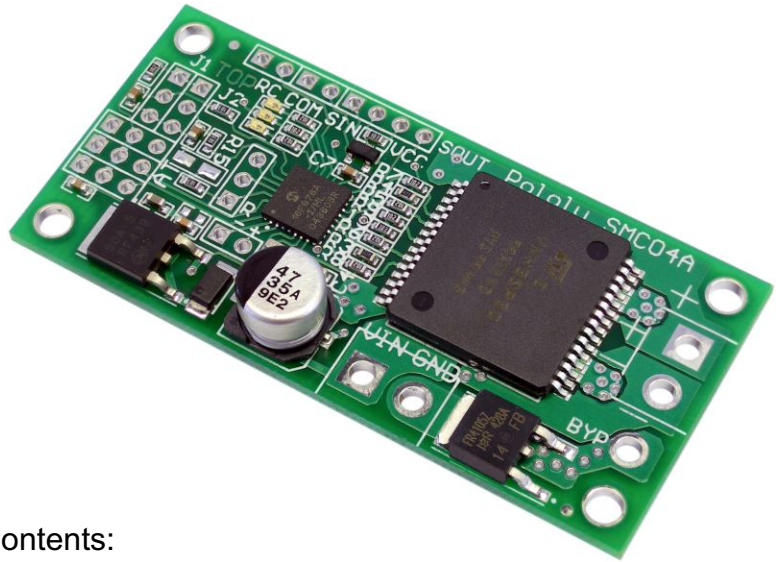
High-Power Motor Controller with Feedback

User's Guide (preliminary)



Contents:

- Safety Warning
- Contacting Pololu
- Motor Controller Layout and Pinout
- Connecting the Motor Controller
- PID Control and the Motor Feedback Modes
- Basics of the Serial Interface
- Using the Motor Controller
- Example BASIC Stamp II Program
- Troubleshooting Tips
- Tips for Best Results
- High-Resolution Serial Interface
- Diagnostic LEDs and Serial Output
- Configuring the Motor Controller
- Description and Specifications

© 2006
http://www.pololu.com/

SMC04B

Important Safety Warning

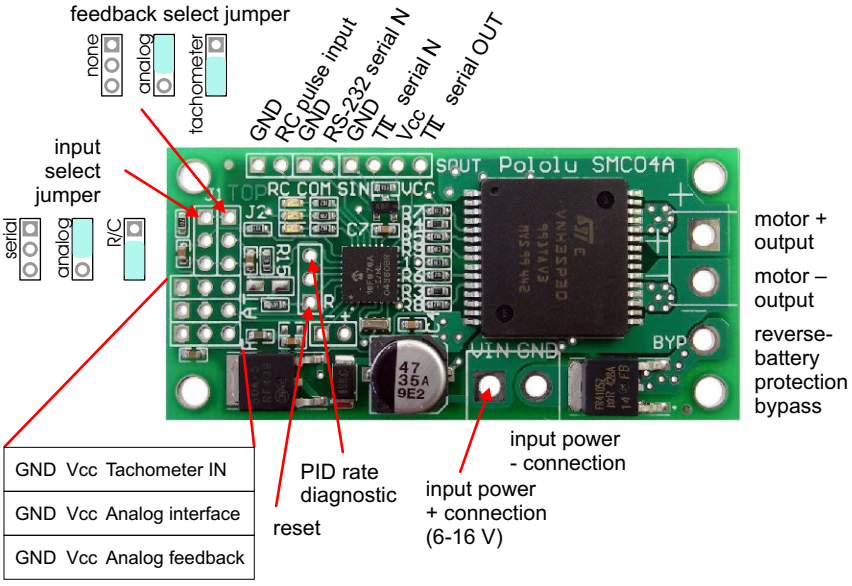
The motor controller module is not intended for young children! Younger users should use this module only under adult supervision. **By using this product, you agree not to hold Pololu liable for any injury or damage related to the use or to the performance of this product. This product is not designed for, and should not be used in, applications where the malfunction of the product could cause injury or damage.**

Contacting Pololu

You can check the Pololu web site at **http://www.pololu.com/** for the latest information about the motor controller, including color pictures, application examples, and troubleshooting tips.

We would be delighted to hear from you about your project and about your experience with our motor controller. You can contact us through our online feedback form or by email at support@pololu.com. Tell us what we did well, what we could improve, what you would like to see in the future, or anything else you would like to say!

Motor Controller Layout and Pinout



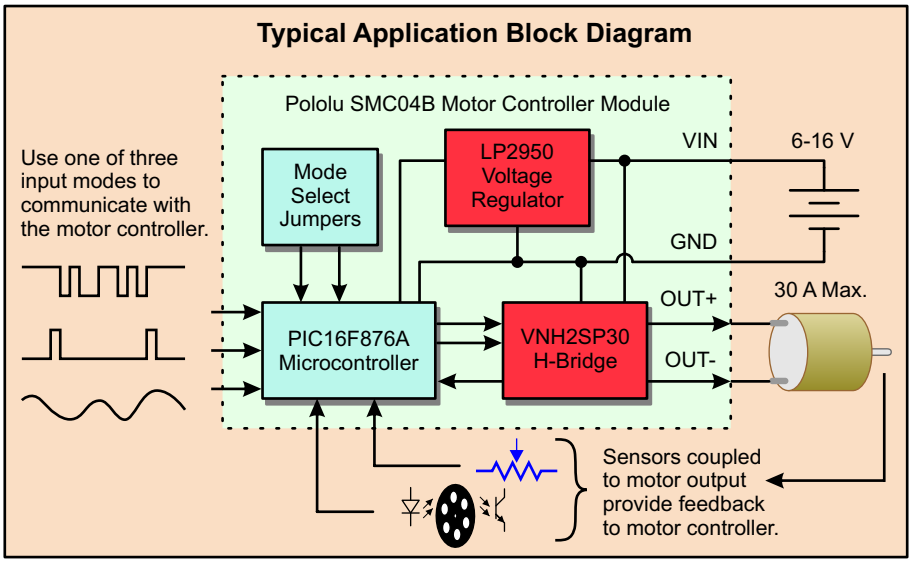
Connecting the Motor Controller

The connections to the motor controller are shown on the facing page, and the block diagram below shows how the motor controller would typically be integrated into a motion control system. There are three main connections to make: the motor itself, the motor power, and motor control inputs. For the control inputs, you can use one of three motor control interfaces (serial, hobby radio control, or analog voltage) and one of two optional feedback options (tachometer for speed feedback or analog voltage for position feedback).

Connecting Power. The main power connections for the motor controller are on the lower end of the board. Your power source should be between 6 V and 16 V, and it must be able to supply the current that the motor will draw.

A reversed-battery protection MOSFET is incorporated into the motor controller. This device has a low voltage drop but still results in a few tenths of a volt being lost between the battery voltage and the motor output (in addition to the losses of the motor driver H-Bridge). The MOSFET can be bypassed by connecting the negative power terminal to the bypass terminal. The non-bypass negative terminal must also be connected.

Connecting the Motor. The motor output ports are on the right side of the board. The polarity of the motor connection does not matter too much, but the "motor positive output" will be positive when the motor is set to go "forward" with the motor direction reverse option turned off. Of course, "forward" is a relative term, and if the motor goes in the opposite direction from what you desire, you can switch the two motor lead connections or enable the motor direction reverse setting.



Connecting the Motor Controller (continued)

Control Inputs. The control inputs are the connection points for the main controlling unit (such as a robot controller or radio control receiver) and any feedback signals. The control inputs are along the top and left sides of the board. The "Vcc" connection provides regulated 5 V that can be used as a power source for the feedback sensors or for the main control unit. The current drain from this pin should not exceed 75 mA.

Reset. The reset input is internally kept high (at 5 V) through a 10 kOhm resistor. Bringing this line low for at least 2 microseconds will reset the motor controller, bringing it to the same state as if it had just been turned on. This input can generally be left disconnected.

Motor Control Interfaces

The motor controller supports either an analog voltage interface, a hobby RC (radio control) interface, or a serial interface. Each interface allows you to control either the direction and speed or the position of the motor, depending on the feedback mode you choose: no feedback, analog voltage (motor position) feedback, or tachometer (motor speed) feedback.

Analog Voltage Interface
The analog voltage interface allows you to control the motor using a device that outputs an analog voltage, such as a potentiometer. To select the analog voltage interface, place a shorting block across the upper two pins in the "input select jumper" (J1) as illustrated on page 2.

The neutral, or stopped value for the analog input is 2.5 V. Lowering the voltage will increase the speed in reverse; raising the voltage will increase the speed forward.

Hobby RC (Radio Control) Interface
The Hobby RC interface allows you to control the motor with a standard hobby RC transmitter and receiver pair; you can also use any device that outputs the standard servo control signal (50 Hz train of 1-2 ms pulses), such as a serial servo controller. To select the hobby RC interface, place a shorting block across the lower two pins in the "input select jumper" (J1) as illustrated in page 2. Connect the RC signal to the pin labeled "RC" on the upper-left corner of the board. A ground pin is available next to it.

A 1.5-ms pulse is neutral or stopped, with longer pulses raising the speed forward and shorter pulses increasing the speed in reverse.

Serial Interface
The serial interface allows motor control with a series of commands over an asynchronous serial link. This interface would typically be used with a computer or robot controller. The controller automatically detects a baud rate of approximately 2,000-60,000.

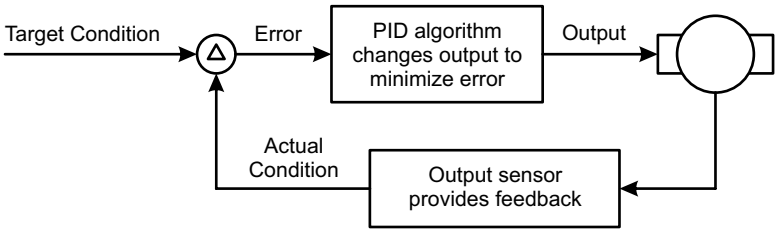
Serial Interface (continued)
There are two serial pins that can be used. The TTL-level serial input is for non-inverted data at logic levels. For direct connections to a PC serial port or other inverted sources, the RS-232 serial input pin can be used. The serial output pin transmits motor controller diagnostic data, which can be used to tune the PID parameters and to detect fault conditions. The serial output can be ignored in many cases.

When building circuits that connect to a PC, be especially careful because you could potentially destroy the PC's serial port. Before attempting to connect your own electronics to a computer, make sure you know what you are doing!

The details of the serial protocol are covered in "Basics of the Serial Interface".

PID Control and the Motor Feedback Modes

The primary feature of this motor controller is the ability to use feedback from sensors to continually update the motor operation. If one of the two feedback modes is enabled, the motor controller uses a proportional-integral-derivative (PID) algorithm to continually minimize the feedback error. This error is the difference between the desired condition, set by the user through one of the input modes, and the actual condition, reported back by a sensor that somehow monitors the motor output. Two separate types of sensors are supported: an analog voltage sensor and a frequency sensor. The general concept of feedback-based control is illustrated below:



Analog Voltage Motor Feedback (Position)

In position feedback mode, an analog voltage representing a measurement of the output is connected to the analog feedback pin. The voltage range must be 0-5 V, and this voltage is converted into a 10-bit representation. A potentiometer is the most typical sensor for this mode, but any other device that provides an analog voltage output, such as the Sharp GP2Y0A21YK optical distance sensors, can be used.

Frequency Motor Feedback (Speed)

In frequency feedback mode, an oscillating signal is connected to the tachometer feedback pin. The frequency of the oscillation is measured with 10 bits of resolution. A typical application for this mode of operation is a tachometer that uses an infrared emitter and detector pair and a slotted wheel to measure rotational speed.

The PID Calculation

The motor controller minimizes the error between the desired state and the measured state using three terms: one that is proportional to the error, one that is proportional to the integral of the error, and one that is proportional to the derivative of the error. The full equation for the calculation is

$$\text{output} = c_E E + c_I \Sigma E + c_D \frac{\Delta E}{T_{PID}}$$

where E is the error, c_E, c_I, and c_D are the user-specified coefficients, and T_{PID} is the PID loop period, which is also user configurable.

Each of the three coefficients are specified through two parts: a multiplicative coefficient and an exponent that divides the result. For example, setting the parameters ERRORMULT to 29 and ERRORDIV to 3 will result in the error term being multiplied by $29/2^3 = 29/8 = 3.625$.

The PID loop period is a multiple of the PWM (pulse width modulation) period (set by PWMFREQ), based on the PIDRATE parameter, of which bits 6-3 specify a coefficient and bits 2-0 specify a base-2 exponent:

$$T_{PID} = T_{PWM} \times [(\text{PIDRATE6:3}) + 1] \times 2^{\text{PIDRATE2:0}}$$

where

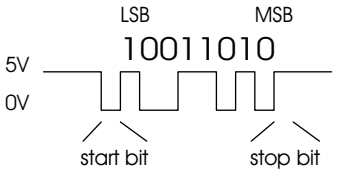
$$T_{PWM} = \begin{cases} 51 \text{ us if PWMFREQ} = 0 \\ 204 \text{ us if PWMFREQ} = 1 \\ 816 \text{ us if PWMFREQ} = 2 \end{cases}$$

All of the parameters can be set through the serial interface, and the optimal coefficients must be determined experimentally. In general, begin by setting the proportional term and leaving the integral term and derivative term 0. Depending on the application, a proportional term alone could cause oscillations or inability to correct for small errors. Increasing the integral term improves correction for small errors but can contribute to oscillation if the integral term is too large. The derivative term functions as a brake when the error is rapidly being reduced, which limits oscillation.

Basics of the Serial Interface

The motor controller uses a serial interface to communicate with a main controller, which could be a small microprocessor or a desktop computer. To use the motor controller, you must program your main controller to send data with the correct format to the motor controller's asynchronous serial input.

The motor controller expects eight bits at a time (with no parity bit) at a constant baud rate ranging from 2000 to 60000 (the motor controller will automatically detect the baud rate). The serial bits must be at logic levels and *non-inverted*, meaning that a zero is sent as a low voltage, and a one is sent as a high voltage, as shown in the diagram to the right. (The RS-232 (inverting) input must be used with a PC serial port since it outputs inverted serial data.) *Commands sent to the serial input **must** conform to the above format or else the motor controller and other devices connected to the serial line may behave unexpectedly.*



Once you can send individual bytes correctly, you must send the correct sequence of bytes to get the motor controller to run your motors. This motor controller *interface protocol* is compatible with other Pololu serial devices such as our servo controller, so you can control multiple Pololu serial devices on a single line. The protocol requires one start byte, a one-byte device identifier, and then any number of bytes, as required by the device specified in the second byte:

start byte = 0x80	device type	data byte 1	data byte 2

The start byte is identified by its most significant bit being set; all subsequent bytes must have bit 7 clear, giving them possible values of 0 to 0x7F (0 to 127 decimal). Whenever a byte is transmitted on the serial line, all devices on that line check to see if the byte is the start byte; if it is, then all devices check the next byte to see if the data is meant for them. All subsequent bytes, the data bytes in the diagram above, are only interpreted by the appropriate devices, while all other devices wait for a new start byte.

If you did not understand all of the details above and you just want to use your motor controller, don't worry. You just need to use the right serial settings and send the correct sequences of bytes, as described on the following pages.

Summary: Use non-inverted, logic-level serial transmission at baud rates between 2000 and 60000, 8 bits at a time with no parity and one stop bit.

Using the Motor Controller

To set the speed and direction of a motor, send a four-byte command with the following structure to the motor controller:

start byte = 0x80	device type = 0x00	motor # and direction	motor speed

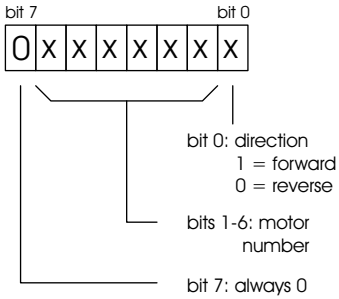
The Four-Byte Motor Controller Command

Byte 1: Start Byte. This byte should *always* be 0x80 (128 in decimal) to signify the beginning of a command. The start byte is the only byte with the highest bit (bit 7) set, and it alerts all devices on the serial line that a new command is being issued. All succeeding bytes sent down the serial line must have their highest bit cleared to zero.

Byte 2: Device Type. This byte identifies the device type for which the command is intended, and it should be 0x00 for commands sent to this motor controller. All devices that are not motor controllers ignore all subsequent bytes until another start byte is sent.

Byte 3: Motor Number and Direction. This byte has three parts, as shown in the diagram to the right:

- Bit 0 specifies the direction of the motor. Set this bit to 1 to make the motor go forward; clear the bit to make it go backward.
- Bits 1-6 specify the motor number. All motor controllers respond to motor number 0
- Bit 7 must be cleared since this is not a start byte.



To obtain the complete byte 3 value from a motor number and a direction, multiply the motor number by 2 and add 1 if the direction is forward. For example, to make motor 5 go forward, byte three should be 5 x 2 + 1 = 11. To make motor 1 go backward, byte 3 should be 1 x 2 = 2. (Two efficient ways to multiply by 2 in a microcontroller program are shifting left by one digit or adding the motor number to itself.)

Byte 4: Motor Speed. The most significant bit must be zero since this is not a start byte. The remaining seven bits specify the motor speed. The possible range of values for byte 4 is thus 0x00 to 0x7F (0 to 127 decimal). 0x00 turns the motor off, and 0x7F turns the motor fully on; intermediate values correspond to intermediate speeds. In analog feedback mode, setting a speed of 0 in reverse will cause the motor controller to hold position 0; setting a speed of 0 forward will cause the PID loop to be turned off.

Examples: (Using PBASIC "SEROUT" command with serial line on pin 5)

```
' "84" parameter sets up 9600 baud serial communication
SEROUT 5, 84,[$80,0,5,127]  'motor 2 full on, forward
SEROUT 5, 84,[$80,0,5,0]    'motor 2 off, forward (coasting)
SEROUT 5, 84,[$80,0,4,0]    'motor 2 holding position 0
```


© 2006

Controlling Multiple Motor Controllers with One Serial Line
To control a particular motor, you must specify its motor number in command byte 3. Regardless of configuration, every motor controller responds to commands for motor number 0. To control more than one motor with a single serial line, you need to use motor numbers 1 through 63. Configure each motor controller to respond to different motor numbers, then connect them to the same serial line; each motor controller will respond only to the motor number to which it is configured. After you configure a motor controller, you can write its motor number on a label to keep track of your motor numbers.

Example BASIC Stamp II Program

The program on the next page, which can run on a BASIC Stamp II controller, makes motor 0 gradually speed up, then slow down, then speed up in the other direction, and then slow down again. For the code to work, pin 15 must be connected to the reset pin, and pin 14 must be connected to the logic-level serial input. The interface code should look similar in other programming languages; the description below should help you in understanding the code and, if necessary, in translating it to other languages.

On line 1, the 8-bit variable `speed` is declared for later use. The serial line is then taken high, to its idle state, before the motor controller is reset by a low-going pulse on pin 15 (lines 3 and 4). A 100-ms pause on line 5 ensures that the motor controller is up and running before any serial data is sent to it.

The first *for loop* on lines 6-9 causes motor 0 to gradually speed up. The serial output is created by the `serout` statement on line 7. The first parameter, 14, specifies the pin number through which to send the serial signal. The next parameter, `84`, sets up the serial characteristics to be 8 bits with no parity, non-inverted, at a baud rate of 9600. The four numbers in square brackets are the data to be sent, and they correspond to the four control bytes for the motor controller. The first two bytes should always be `$80` and `0`. The second `0` makes motor 0 go backward. The speed variable, which increases every time through the loop, is the only part of the command that changes, and that is what makes the motor gradually speed up. The `pause` statement on line 8 causes the program to wait for 20 ms (0.02 seconds) before sending the next command.

When the first loop ends, the motor is set to its full speed of 127. The second loop on lines 10-13 slows the motor back down by sending speeds from 127 down to 0. The next two loops on lines14-21 then repeat the process, except for the parameter value of `1` in byte three, which causes motor 0 to spin forward.

```
1       speed    var   byte
2       high 14        'take serial line high
3       low 15         'reset motor controller
4       high 15
5       pause 100      'motor controller startup time
6       for speed = 0 to 127
7          serout 14,84,[$80, 0, 0,speed]
8          pause 20
9       next
10      for speed = 127 to 0
11         serout 14,84,[$80, 0, 0,speed]
12         pause 20
13      next
14      for speed = 0 to 127
15         serout 14,84,[$80, 0, 1,speed]
16         pause 20
17      next
18      for speed = 127 to 0
19         serout 14,84,[$80, 0, 1,speed]
20         pause 20
21      next
```

Troubleshooting Tips

All motor controllers are fully tested prior to shipment; if your motor controller does not work at first, it can be difficult to determine the cause. Nevertheless, patience and meticulous attention to detail, along with these few tips, should usually help you through.

- Double check all of your connections. Are your logic and motor supply grounds connected?
- Double check your code. Are your baud rate settings correct? If you cannot get your design working with the top baud rate of 60,000, try lowering it to 9600, where slight timing mismatches are less likely to frustrate your efforts.
- Are you using the correct motor number? If nothing seems to be working, start by using motor number 0, which should work regardless of the configuration.
- Setting the PID coefficients to certain values can cause unexpected results. If you are in a feedback mode, check what you are setting the PID loop to do.
- Keep wiring as short as possible. Keep the electrically noisy motor and power lines away from other signals. See next page for additional information about dealing with noise.
- Are you using a good power supply? Make sure that your supply can deliver the necessary current without big fluctuations in the voltage. A capacitor across the power input pins can help.

Tips for Best Results

The user-configurable parameters mentioned below are listed at the end of this manual.

Limit noise.

In analog feedback mode, good measurements of position are critical to achieving good results. Try to keep the analog signals away from the noisy motor wires, and keep all wires as short as possible. The motor controller can average up to 64 samples each time to get the best possible reading; however, taking more samples can also limit the PID frequency.

The derivative term can be especially sensitive to noise. To reduce the effect of noise, bit 6 of MISCPARAM0 can be set to make the derivative calculation based on 5 sample times instead of 1. This effectively increases the derivative term by a factor of 5 for legitimate error changes without changing the effect of noise. Typically, turning on this feature and reducing the derivative term coefficient by a factor of 5 improves performance.

Power Considerations.

The motor driver on the SMC04B motor controller can deliver up to 30 A to the load. However, the practical performance is limited by thermal issues, so the 30 A can appear for only a short time before the motor controller overheats. Various tradeoffs of run-time and current can be achieved, and mounting a heat sink to the motor driver can also improve run time. Without a heat sink, the motor controller will typically begin to overheat after a few minutes of running at 10 A. At 15 A, the motor controller will run for about 30 seconds.

While the run time and current tradeoffs may be constrained by a particular installation, limiting the worst-case current surges can make the difference between a successful application and an overheated controller. The current that a motor draws can vary a great deal depending on its load, and switching directions is the most demanding operation for the motor and controller. To help manage the associated current spikes, the ACCELLIMIT parameter allows the motor to gradually ramp up to a target speed. Bit 6 of MISCPARAM1 allows the maximum PWM duty cycle to be capped at 50%. While this can limit the peak performance of a motor, it can also bring the load within the limits of the motor controller.

Since switching directions is very demanding on a system, careful tuning of the PID parameters can make a substantial difference. Eliminate oscillations in the output, and you might very well eliminate any overheating problems.

High-Resolution Serial Interface

The four-byte serial protocol detailed on page 8 is designed to be compatible with the Pololu dual serial motor controllers and the SMC03A motor controller with feedback. The SMC04 module has expanded resolution for finer control of a motor, and to access that resolution, a 5-byte protocol is available.

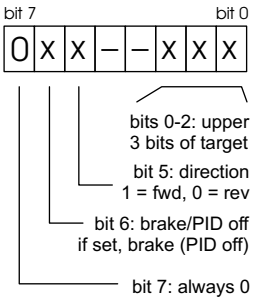
start byte = 0x80	device type = 3	motor number	data byte 1	data byte 2

The Five-Byte Motor Controller Command

Bytes 1 and 2: Start Byte and Device Type. The first two bytes must be 0x80 and 0x03 for high-resolution commands.

Byte 3: Motor Number. This is the motor number for which this command is issued. All devices respond to motor number 0; unique numbers can be set to control multiple controllers off of one serial line.

Byte 4: Data Byte 1. The first data byte has three parts, as shown in the diagram to the right:



- Bit 6 can be set to turn off PID and to instead brake at the target value. Braking is accomplished by tying both motor outputs to ground.
- Bit 5 applies to speed-feedback and no-feedback modes. A 1 sets a forward speed target; a 0 sets a reverse speed target. This bit is ignored in analog-feedback mode.
- Bits 0-2 are the upper three bits of the 10-bit target value.

Byte 5: Data Byte 2. This byte has the lower 7 bits of the target value (bit 7 must be 0).

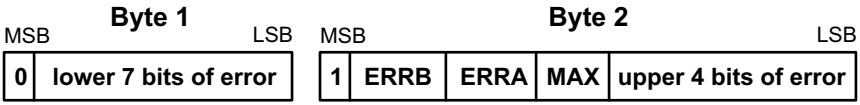
Diagnostic LEDs and Serial Output

Diagnostic LEDs. The motor controller includes three status LEDs. After initial setup, the LEDs have consistent functions across all modes:

- Green LED: error small. This LED lights if the PID error is small. When the PID function is turned off, this LED will always be on.
- Yellow LED: motor error. This LED lights if the motor driver overheats or otherwise signals a fault condition.
- Red LED: maximum speed. This LED lights if the motor PWM duty cycle is close to 100%.

Diagnostic LEDs and Serial Output (continued)

Serial Output. The motor controller sends two data bytes to the serial output pin for every iteration through the PID calculation loop. The baud rate is the same as the input baud rate in serial mode; for analog and RC inputs, the output rate is 38400 bits per second.



The serial data corresponds roughly to the LED values, but more detail is available. All 11 bits of the PID error calculation are sent out as a 2's complement signed number. The MAX bit is set if the PWM duty cycle is close to 100% (when the red LED is on). The ERRA bit is set if the motor driver reports a problem on the A (positive output) side, and the ERRB bit is set if the motor driver reports a problem on the B (negative output) side. (The yellow LED thus corresponds to the logical ORing of the two bits.)

Using the Diagnostic Feedback

The LEDs or serial output can be used to determine the operation of the PID controller. Typically, the error should be small, and the red LED should turn on only when there is a big change in the target condition. If the error does not get smaller and the motor speed is not at its maximum, the proportional or integral terms might need to be increased. If the error does not get smaller and the motor output is at its maximum for a prolonged period of time, the desired target might not be attainable. Such a condition could arise if the mechanism is jammed or if a target speed faster than the capability of the mechanism is requested.

The serial output can be useful for run-time monitoring of the motor controller, or for calibration of the PID parameters. For instance, a computer program could be written to set the PID parameters, move to a new position, and record the error outputs. A graph of the error could then be presented to the user for an objective measure of the mechanism's performance. The computer program could even iterate through multiple PID parameter settings to automate the calibration of the PID parameters.

A separate pin above the reset line (see pinout on page 2) sends out a pulse every time the PID calculation is executed. This pin can be used with a frequency counter to measure the PID rate. The pulse is also synchronized with the serial output, so it can be used as a trigger when looking at the serial output on an oscilloscope.

Configuring the Motor Controller

The motor controller has many parameters that can be configured to customize and optimize the operation of the motor controller for many different installations. To change any of these settings of the motor controller, send a four-byte command with the following structure to the motor controller:

start byte = 0x80	config type = 0x02	parameter address	value

After each parameter change command, the motor controller stores the new parameter, echoes back the value over the serial output, and then reinitializes the controller, allowing the new parameter to take effect immediately. The parameters are stored in non-volatile memory that can be rewritten thousands of times but not infinitely many times. An automated setup that rewrites a parameter over and over could potentially burn out the memory in a few minutes.

The configuration command is very similar to the usual speed setting command, but byte 2 has a device type of '2' instead of '0'. The next two bytes specify the parameter that is to be set and the value for that parameter. The parameters and their functions are listed below:

address	name	description
0	MOTORID	motor number to which this motor controller responds valid range is 0-63; default value is 1
1	ERRORMULT	error (proportional) term multiplier valid range is 0-127; default value is 0x19
2	ERRORDIV	error (proportional) term divider valid range is 0-7; default value is 1
3	INTEGMULT	integral term multiplier valid range is 0-127; default value is 0x0C
4	INTEGDIV	integral term divider valid range is 0-7; default value is 4
5	DERIVMULT	derivative term multiplier valid range is 0-127; default value is 0x19
6	DERIVDIV	derivative term divider valid range is 0-7; default value is 1
7	PIDRATE	sets the PID update rate (also affected by PWMFREQ) valid range is 0-127; default value is 0x59 (~200 Hz)

Configuring the Motor Controller (continued)

address	name	description
8	MISCPARAM0	bit 6 increases the derivative sample period if set bit 5 enables power-up potentiometer test if set bits 4-3 set the frequency feedback divider frequency is divided by 2^<these two bits> bit 2 inverts the analog feedback if set bit 1 inverts the analog control input if set bit 0 inverts the motor direction if set valid range is 0-31, default value is 0
9	MISCPARAM1	bit 6 limits the PWM output to 50% if set bits 3-5 set the number of samples the analog input takes bits 0-2 set how many samples the analog feedback takes For both inputs: 3-bit value — number of samples 0 — 1 1 — 1 2 — 2 3 — 4 4 — 8 5 — 16 6 — 32 7 — 64 Default value is 0x3F (64 samples on both inputs, speed limit off)
10	INTEGLIMIT	Limits the integral to (<this parameter value> + 1) x 8 default value is 7
11	UPPERLIMIT	In analog feedback mode, do not allow target above 1023–(<this parameter value> * 4) default value is 13
12	LOWERLIMIT	In analog feedback mode, do not allow target below <this parameter value> * 4 default value is 13
13	ACCELLIMIT	Each time through the PID loop, the motor speed cannot be raised more than this value. If ACCELLIMIT is set to 0, this feature is turned off. Default value is 60.
14	PWMFREQ	Sets PWM frequency. 0 for 20 kHz, 1 for 5 kHz, 2 or 3 for 1.25 kHz . Default value is 1.

The Pololu SMC04B High-Power Motor Controller with Feedback

The Pololu high-power motor controller with feedback simplifies servo control of commonly available DC motors. The module features three independent interfaces: a serial protocol for microcontroller-based applications, a pulse-width interface for connection to hobby radio control equipment or serial servo controllers, and an analog voltage interface for simple tests and demonstrations. Two feedback alternatives allow for closed-loop control of position or speed.

The motor controller offers a complete feedback-based solution for applications requiring bi-directional, closed-loop control of motor speed or motor position. You can select either an analog voltage feedback or a digital encoder feedback (quadrature encoding is not supported). Various simple devices such as potentiometers can be used as sensors to achieve position or speed control.

In a typical application, a user first sets up the motor controller's parameters to suit the mechanical system that is being driven. A feedback potentiometer is coupled to the mechanism output, and the user can then send position commands to the motor controller, which automatically drives the motor to reach the position. Alternatively, the mechanism and sensor can be arranged to provide speed feedback, in which case the motor controller maintains given speeds despite fluctuations in friction or supply voltage.

The motor controller takes up less than 3 square inches and has an operating voltage of 6-16 volts, making the device well-suited for medium-sized robots and other projects using DC motors and rechargeable 7.2 V to 12 V battery packs. The motor controller serial protocol is compatible with other Pololu motion control devices, allowing multiple units to be connected to a single serial line to control a mixture of motors sizes and hobby RC servos.

Specifications

PCB size	2.475" x 1.175"
Motor ports	1
Motor speeds	1023 speeds forward, backward, brake
Motor Positions	1024 (analog modes)
Maximum current	30 A
Motor supply voltage	6-16 V
PWM frequency	1.25 kHz, 5 kHz, or 20 kHz
Serial baud rate	2000-60000 (automatically detected)