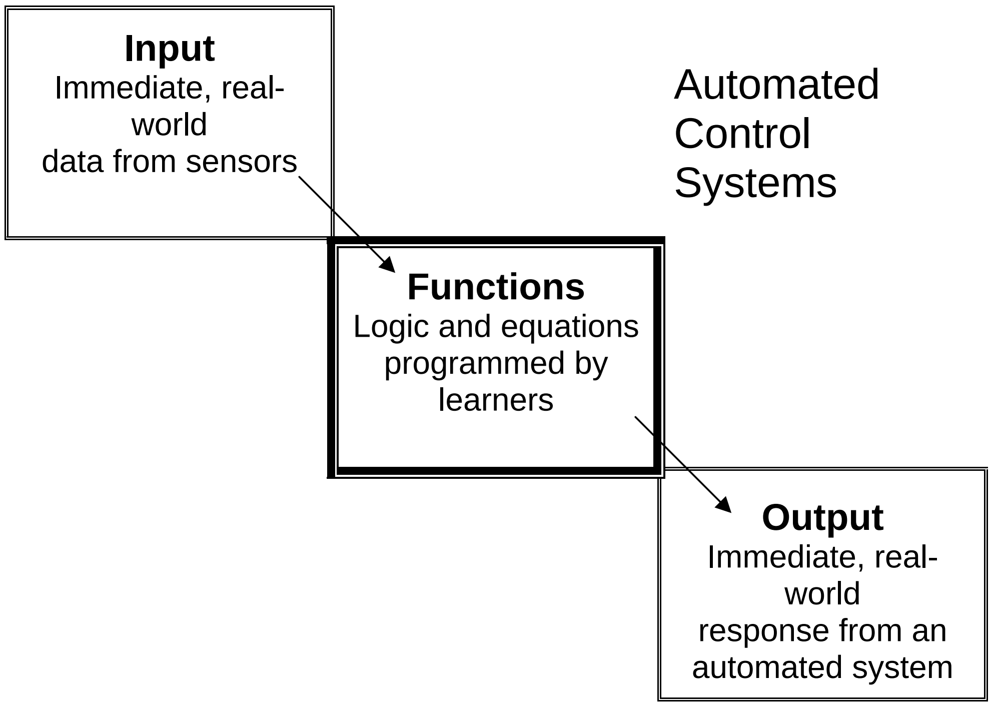
Using the Motor Controller

The *Motor Controller* is designed to be a convenient tool for teachers and students who want to use math and science to make thing happen. Mathematical equations are the heart of math, science and technology, and the *Motor Controller* can help students to see and understand those equations. The equations are not just “formulas” to find the answer to a specific word problem; they are real, functioning links between actual events. Systems that automatically control motors and other equipment are a vital part of modern engineering and technology, and they are a very powerful tool for use in math and science classrooms.


Input
Immediate, real-world data from sensors
Automated Control Systems
Functions
Logic and equations programmed by learners
Output
Immediate, real-world response from an automated system

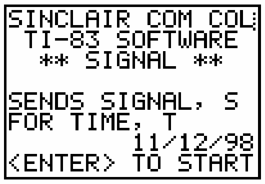
Using the Motor Controller

Table of Contents

Interactive Programs and Programming Modules

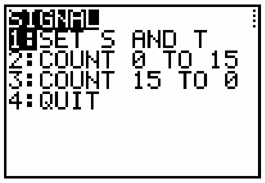
Our goal is to make the programming easy enough for students and teachers to enhance the learning of math and science, rather than obstructing it. We want students to focus on the math and science content and on the application of that content to realistic technical situations, not to focus on the details of programming. At the same time, we want all the programming to be open, so that anyone who is interested can view and perhaps improve on the existing programs.

The programs that work the Motor Controller are available both as *interactive programs* and as *programming modules*.[1] Interactive programs (such as the SIGNAL program at right) ask users to provide input through on-screen questions and menus The programming modules (such as SIGNALF below) have the same functionality but are designed for use by students or teachers as building blocks for writing their own programs. We have also established a system for the use of variables which allows the different subroutines to be used without conflicts.



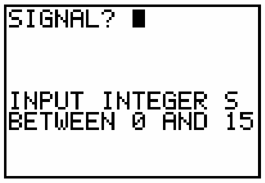
```
SINCLAIR COM COL
 TI-83 SOFTWARE
  ** SIGNAL **

SENDS SIGNAL, S
FOR TIME, T
        11/12/98
<ENTER> TO START
```



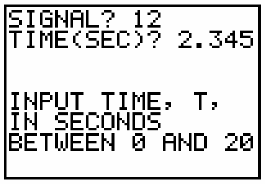
```
SIGNAL
1:SET S AND T
2:COUNT 0 TO 15
3:COUNT 15 TO 0
4:QUIT
```

The most straight-forward use of the Motor Controller is simply to turn things on and off. The interactive SIGNAL program does this by requesting that the user input an integer value for the signal, S, and then input a time, T, in second. The program activates the corresponding lines on the Motor Controller using the binary representation of S. The program at right turns on line 4 (binary 1000 = $2^3 = 8$) and line 3 (binary 0100 = $2^2 = 4$) for 2.345 seconds.



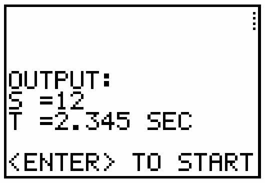
```
SIGNAL? █


INPUT INTEGER S
BETWEEN 0 AND 15
```



```
SIGNAL? 12
TIME(SEC)? 2.345

INPUT TIME, T,
IN SECONDS
BETWEEN 0 AND 20
```



```


OUTPUT:
S =12
T =2.345 SEC

<ENTER> TO START
```

The corresponding “programming module” is SIGNALF, meaning “Signal for”. SIGNALF assumes that the integer has already been stored as the variable, S, and that the time (in seconds) has already been stored as the variable, T. This subroutine activates the output using the existing values of those two variables.

```
2üS
5üT
prgmSIGNALF
0üS
3üT
prgmSIGNALF
5üS
2üT
prgmSIGNALF
```

The TI-83 / 84 program at left (easily entered from the keypad of a calculator) is all that is required to execute a series of 3 timed signals. If used with the RC Controller, this program would make the car move forward for 5 seconds, stop for 3 seconds, then back while turning left for 2 seconds.

[1] This principle applies to both the currently available programs for TI graphing calculators and to LabVIEW computer programs that will be available in the Fall of 2005 at www.mathmachines.net.

What the Motor Controller Really Can Control—and How

We call it a “Motor Controller” because it does a good job controlling several types of motors. On the other hand we may need to change the name because the board can also control almost any other type of low-current DC device, including lights, buzzers, solenoids, fans, “muscle wire” and more.

It is extremely important, however, that any power supply and any devices you attach to the Motor Controller have the correct current and voltage specifications.

- ***The devices you attach must be DC devices, the total current for all devices must not exceed 1 A (1000 mA), the current for any one device must not exceed 0.35 A (350 mA), and the voltage must not exceed 24 V.***
- ***The power supply’s voltage and current capacity must match the needs of the device.*** *(Servo motors are an exception, since the board’s voltage steps down the applied voltage to give the servo motors the 5 volts they need.)*

Lights, Buzzers and Other Basic DC Devices

We’ll start with the miscellaneous things like buzzers and lights. You need to provide a power source that matches the current and voltage requirements of the devices. In general, we prefer to work with 12-V devices and to use a 1-A, 12-V filtered power supply.

In general, these devices are attached using the 6-position, screw terminal block, Two of the terminals on that block are marked as “+,” and these are both connected directly to the positive of your input source. If you connect the positive wire of a buzzer, for example, to one of these “+” terminals and then touch the buzzer’s negative wire to the board ground (say, the heat sink on the voltage regulator or the “GND” pin on one of the servo connectors) the buzzer will sound. The trick is to make that connection to ground under software control, and that’s where the numbered terminals on the terminal block come in. Normally, terminals 1, 2, 3 and 4 are non-conducting—they keep anything connected to them isolated from ground. The same software signal that lights the each LED also completes the circuit through the corresponding numbered terminal and the ULN2003A to ground.

For example, if you put the positive lead for a buzzer in one of the “+” terminals and the buzzer’s negative lead into terminal “3,” the buzzer will sound if and only if you send a software signal that lights the green LED. Under the binary system, that signal could be binary 0100 = 2^2 = 4. The signal could also be any other value that lights the green LED, such as binary 0101 = $2^2 + 2^0$ = 5.

The motor controller provides independent control for up to 4 different devices. With the buzzer’s negative still attached to terminal 3, for example, you could also attach a light from the other “+” to the “1” terminal. (Lights typically conduct in either direction, so it doesn’t matter which lead goes in which terminal.) Which this set of connections, the signal 4 (binary 0100) would sound the buzzer, the signal 1 (binary 0001) will light the light, and the signal 5 (binary 0101) will activate both simultaneously.

Some DC devices (for example fans designed for use in computers) may have connectors that fit onto the headers. The 6 header pins match up with the terminal blocks as alternate connection points. Putting a fan connector onto pin 4 and the "+" header next to it will make it possible to activate the fan by sending the signal 8 (binary 1000). If the device is polarized (usually with a red and a black lead), be sure the red (positive) lead goes onto the "+" pin.

The various versions of the SIGNAL and CONTROL programs can all be used to control any of these DC devices.

Servo Motors

Important note: You can mix and match servo motors with the basic DC devices described above *but not using the same outputs*. *It is possible, for example, to operate a fan on line 4 (binary 1000) and a servo motor on line 2 (binary 0010), but you should never attach a servo motor to the same line you are using to operate a fan, light, etc. If the constant signal that operates the DC devices is also sent to a servo motor, the servo motor will draw a significant amount of current without actually running correctly.* *This can damage the servo motor*.

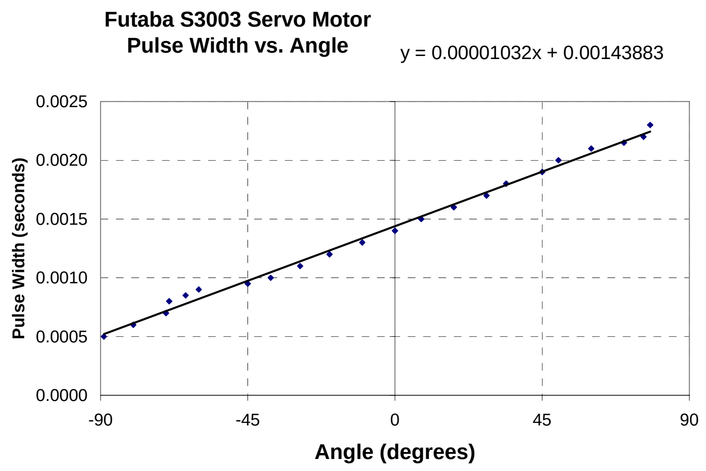
Bottom line—Each LED needs to be doing just one job.

Servo motors are widely used in advanced RC hobby vehicles such as model cars, boats and airplanes. These motors do not power the vehicles, but are used instead to turn the steering, adjust sails or rudders, etc. They include internal electronics to interpret instructions and also to measure their own position. If you tell a servo motor to move to position 35°, for example, it should move to that position. If it is already at 35° when it receives the signal, it may bounce back and forth a little but it should not move more than a little. Servo motors do not have the high precision and speed control of stepper motors, but they are a very convenient, low-cost method for achieving many goals. Four servos (or eight if you use both LabPro outputs) are enough to operate a model robotic arm. A simpler example of using a servo motor would be to raise a flag or drop a ball.

(One limitation of this Motor Controller is that it cannot simultaneously send different signals to different servos. For example, if you move servo 1 to 45° and then move servo 3 to -50°, only the friction in the motor keeps servo 1 at 45°. If a flag on servo 1 is too heavy, it might drop back down when the signal stops. In situations such as this, you need to look for physical solutions, such as adding a counter weight to better balance the flag.)

Hobby servo motors almost all come with 3-pin connects that fit onto the Motor Controller's servo connectors. **It is very important to orient the servo connector correctly, with its negative wire (almost always black) connected to the GND ground pin.**

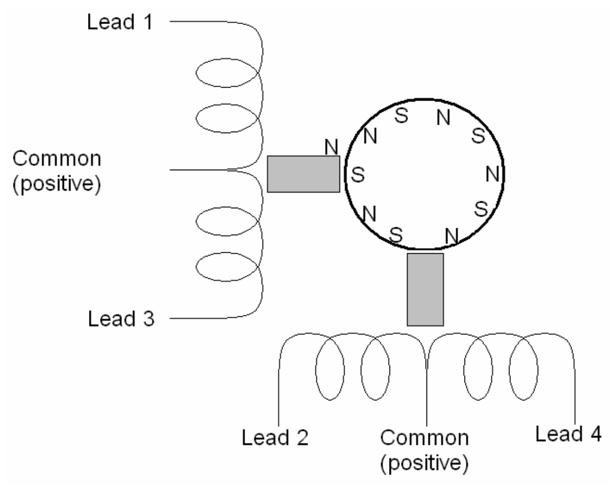
Instructions to the servo motor are in the form of pulses. The signal line (opposite the ground line) goes briefly from 0 V to 5 V about once every 20 ms. The desired position of the servo motor is signaled by the width of this positive pulse—the time is remains high. The desired pulse widths are usually between about 0.5 ms and 2.5 ms, and the resulting positions are usually between about -80° and +80°. A sample calibration curve is shown at right, but the



values can be very different for different brands and models of motors, sometimes even reversing positive and negative movements. The calibration can also be different even among servo motors that are the same brand and model. They really do need to be calibrated individually to achieve the maximum effectiveness.

The SERVO and MCPUT programs control servo motors.

Stepper Motors



Stepper Motors are rather like traditional DC motors, except that they use external circuitry to reverse the magnets and keep the motion going. In the diagram at right, a current in coil 1 makes the left magnet a north pole. When current flows instead in coil 2, the bottom magnet becomes a north pole and the central magnet rotates counter-clockwise. A current in coil 3 next makes the left magnet a south pole a produces another step in the clockwise direction. The cycle is finished by providing a current in coil 4 to make the lower magnet a south pole. Reversing the sequence (4, 3, 2, 1) yields clockwise rotation. Five-wire stepper motors connect the 2 common leads internally.

The leads that provide current to each coil are always color coded, but the codes are different for different manufacturers and even for different models made by the same manufacturer. Color codes for some of the motors we use are shown below, numbered to correspond with our programs for the Motor Controller.

	Coil 1	*Coil 2*	*Coil 3*	*Coil 4*	*Common*	*Common*
Hurst SAS series	Red	White	Blue	Black	Bicolor	Bicolor
Applied Motion	Green	Red	Green / White	Red / White	White	Black
*GBM 30BYJ011**	Orange	Yellow	Violet	Blue	Red	
GBM 35BY48B09	Green	Red	Yellow	Brown	Black	Black

* The 30BYJ011 fits onto our 6 pin header (avoiding the top pin) with a sequence opposite the one shown. That means the motor goes in the reverse direction from our standard.

Note that since stepper motors use all four lines they cannot be operated simultaneously with either servo motors or other DC devices. When you use the Motor Controller to run a stepper motor, nothing else should be connected to the board. A servo motor connected to the board while you run a stepper motor program can be damaged.

Wave Mode

Clockwise ↑ / Counterclockwise ↓

Coil 1	Coil 2	Coil 3	Coil 4
1	**0**	**0**	**0**
0	**1**	**0**	**0**
0	**0**	**1**	**0**
0	**0**	**0**	**1**
1	**0**	**0**	**0**

Full-Step Mode

Clockwise ↑ / Counterclockwise ↓

Coil 1	Coil 2	Coil 3	Coil 4
1	**1**	**0**	**0**
0	**1**	**1**	**0**
0	**0**	**1**	**1**
1	**0**	**0**	**1**
1	**1**	**0**	**0**

The simplest (but least common) mode of operation for a stepper motor is just as described above—activating one coil at a time to rotate the motor. This "wave mode" is summarized in the table above. The main disadvantage of the wave mode is that it uses only one set of magnets at a time, so torque is rather low. The biggest advantage is that this mode is simple and uses relatively little current.

The most common mode always activates two coils, so both magnets are providing torque. This "Full Step Mode" is shown at top right.

It is also common to alternate between activating one magnet and both magnets. This causes the motor to move with twice the precision. A motor with a 15° step, for example, can be made to move in 7.5° increments. Average torque is lower than in full-step mode.

The STEPPER and MCSTEP programs are designed to operate stepper motors through the Motor Controller board.

Half-Step Mode

Clockwise ↑ / Counterclockwise ↓

Coil 1	Coil 2	Coil 3	Coil 4
1	**0**	**0**	**0**
1	**1**	**0**	**0**
0	**1**	**0**	**0**
0	**1**	**1**	**0**
0	**0**	**1**	**0**
0	**0**	**1**	**1**
0	**0**	**0**	**1**
1	**0**	**0**	**1**
1	**0**	**0**	**0**

Using the Motor Controller with TI Graphing Calculators

Program Descriptions

The calculator programs below are available for download at http://mathmachines.net

SIGNAL – An interactive program that allows the calculator to activate the digital output lines. Applications include turning on lights, buzzers, fans, etc. for a time specified from the calculator.	SIGNAL.8XP
CONTROL – An interactive program that reads the value on the sensor in Analog Channel 1 and activates the output lines in accordance with the logic stored as a function Y_9 in the calculator.	CONTROL.8XP
DCUINIT -- A utility program developed by *Vernier Software* (and distributed with their permission) which verifies correct communications between the calculator and the CBL and which keeps the LabPro from shutting down prematurely.	DCUINIT.8XP
SIGNALF (Signal For) -- This modified version of **Signal** is designed to be used as program module within other programs. Rather that asking the user to input values of S (the signal) and T (time) while the program is running, the values are set in advance by a higher level program.	SIGNALF.8XP
SIGNALW (Signal While) -- This program combines features of **Signal** and **Control** in a program module for use in higher-level programs. It continuously monitors probes in analog channels 1 and 2, assigning the results to variables "E" and "F" respectively. It also records the elapsed time in seconds as "T." The program continues to monitor the inputs until the value of calculator function Y_9 becomes zero. (The program "times out" if the condition in Y_9 does not become zero within 20 minutes.)	SIGNALW.8XP
SERVO – An interactive program that allows the calculator to control hobby servo motors, available in many hobby stores. The user specifies a signal to specify one or more motors and inputs the desired angular position, and then the motor or motors move to that position.	SERVO.8XP

MCPUT – The program module version of ***SERVO***, this program uses the value of S to select one or more servos then moves them to angular position, A.	MCPUT.8XP
STEPPER – An interactive program that controls a unipolar stepper motor, allowing the user to specify the step size, angular speed and angular displacement.	STEPPER.8XP
MCSTEP -- The program module version of ***STEPPER***, this program moves a stepper motor through an angular displacement, A, at angular speed U. The motor's step size must also be stored as variable Q. A must be in degrees and can be either positive (for counterclockwise rotations) or negative (for clockwise). The angular speed, U, must be in rpm (revolutions / minute) and must be positive. The motor's step size, Q, must be specified in degrees.	MCSTEP.8XP
READ1 – A program modules that activates Channel 1, autoidentifies the sensor and take a series of 5 readings at 0.05 s intervals. The average reading is stored as variable E.	READ1.8XP
READ2 -- A program modules that activates Channel 2, autoidentifies the sensor and take a series of 5 readings at 0.05 s intervals. The average reading is stored as variable F.	READ2.8XP
READ3 -- A program modules that activates Channel 3, autoidentifies the sensor and take a series of 5 readings at 0.05 s intervals. The average reading is stored as variable G.	READ3.8XP
READ4 -- A program modules that activates Channel 4, autoidentifies the sensor and take a series of 5 readings at 0.05 s intervals. The average reading is stored as variable H.	READ4.8XP
PROG01 -- A sample "higher level" program to illustrate how ***SIGNALF*** and ***SIGNALW*** can be used by students or teachers to construct their own programs	PROG01.8XP

Variable Use in the Program Modules

Variable	**Use**	**Used but not changed**	**Changed or deleted**	**Never used**
A	Angle (degrees)	X		
B	Short-term internal use		X	
C	Short-term internal use		X	
D	Distance (centimeters)	X		
E	Ch1 Sensor		X	
F	Ch2 Sensor		X	
G	Ch3 Sensor		X	
H	Ch4 Sensor (LabPro only)		X	
I	Counter		X	
J	Counter		X	
K	Key		X	
L	*Reserved for higher level programs*			X
M	*Reserved for higher level programs*			X
N	*Reserved for higher level programs*			X
O	*Reserved for higher level programs*			X
P	*Reserved for higher level programs*			X
Q	Hardware specific data		X	
R	Pulse width		X	
S	Signal or Step counter		X	
T	Time (s)		X	
U	Angular Velocity (degrees/second)		X	
V	Velocity (cm/s)		X	
W	Flag for program control		X	
X	*Reserved for higher level programs*			X
Y	*Reserved for higher level programs*			X
Z	*Reserved for higher level programs*			X
θ	*Reserved for higher level programs*			X