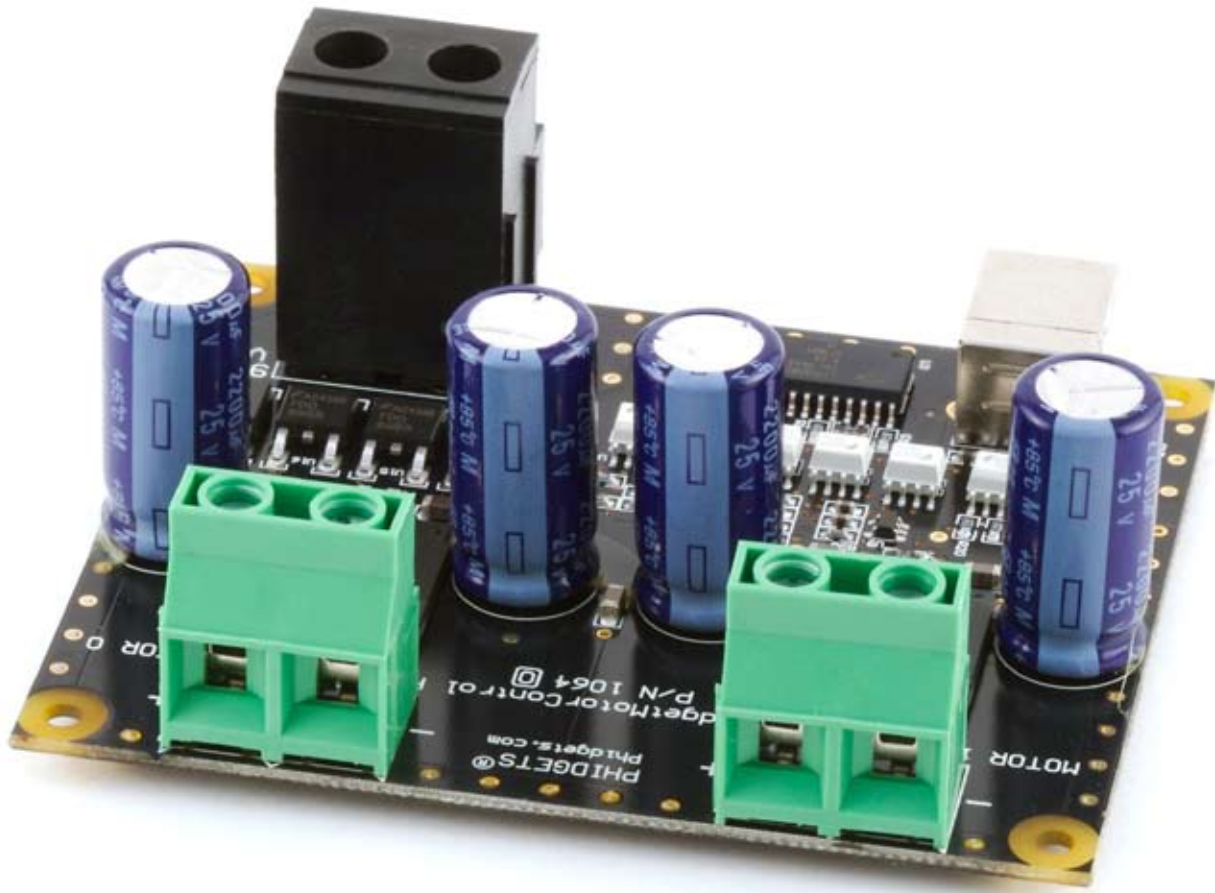


# PhidgetMotorControl HC



## Operating Systems:

Windows 2000/XP/Vista, Windows CE, Linux, and Mac OS X

## Application Programming Interfaces (APIs):

Visual Basic, VB.NET, C, C++, C#, Flash 9, Flex, Java, LabVIEW, and Matlab

## Examples:

You will find program examples in the download section of [www.phidgets.com](http://www.phidgets.com)

## What Can the PhidgetMotorControl HC Do?

The PhidgetMotorControl HC allows you to control the angular velocity and acceleration of up to two high-current DC motors. It provides a generic, convenient way to interface your PC with DC motors and other Phidgets devices.

### DC Motors

The PhidgetMotorControl HC will work with a variety of low-current and high-current brushed DC motors. A few motors are listed below.

Manufacturer	Part Number	Description
GWS	RC-04	12V 1550RPM 0.15oz-in Brushed DC Motor
Banebots	M4-R0062-12	12V 5280RPM 345oz-in Brushed DC Motor
Hennkwell Ind. Co.	PK22G2150	12V 64RPM 341oz-in 1:231 PGH Brushed DC Motor

These and many other brushed DC motors are available directly from the manufacturer or at local distributors.

## Getting Started

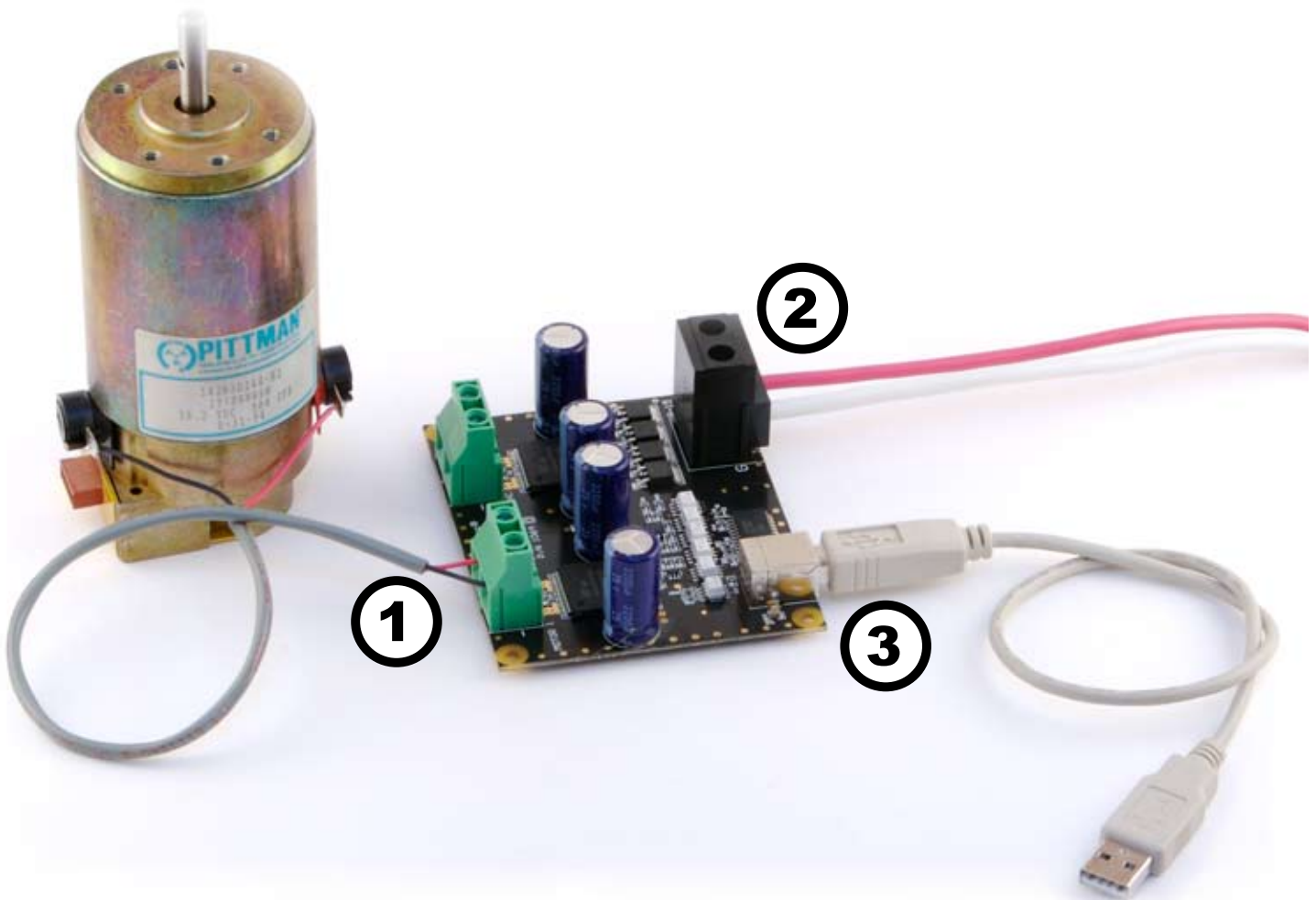
### Installing the hardware

The kit contains:

- A PhidgetMotorControl HC board
- A USB Cable

You will also need:

- A DC Motor
- A 9 to 15V DC Power Supply



1. Connect the motor to the PhidgetMotorControl board.
2. Connect the power supply to the terminal block on the PhidgetMotorControl board.
3. Connect the PhidgetMotorControl board to your PC using the USB cable.

## Download and Install the software

Go to [www.phidgets.com](http://www.phidgets.com) >> [downloads](#)

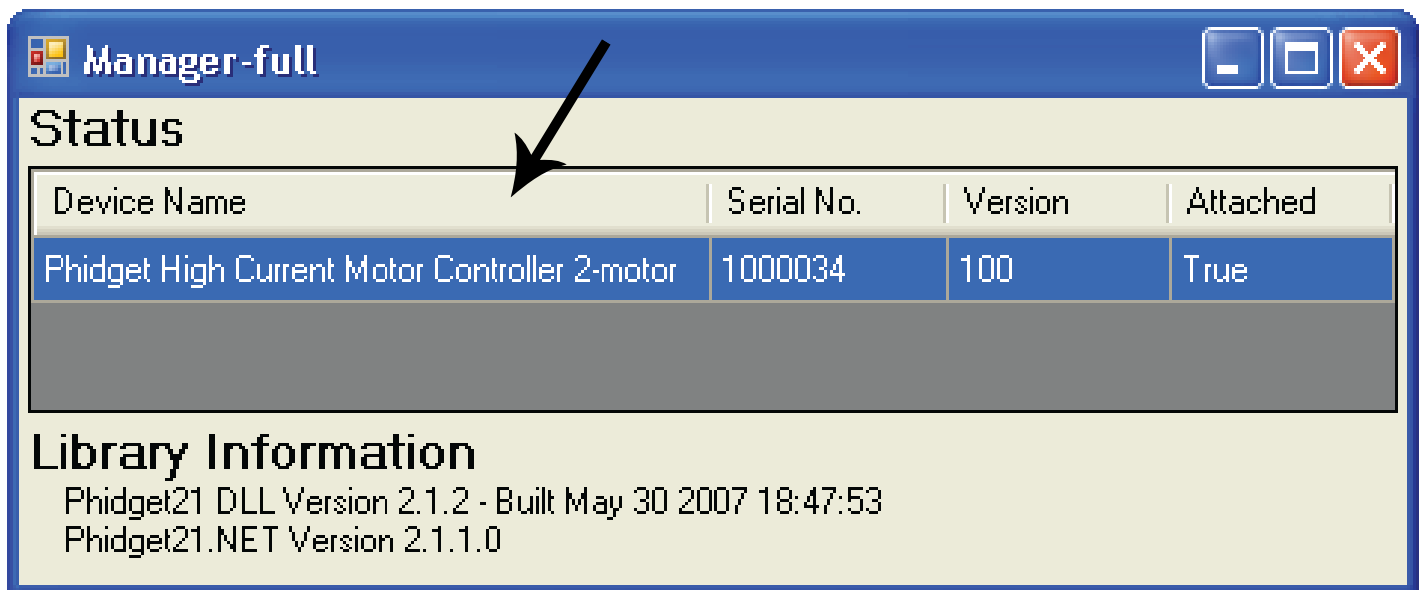
Select your operating system (Windows, Linux, MAC OS)

Select the language you want to use and download the appropriate examples and Libraries.

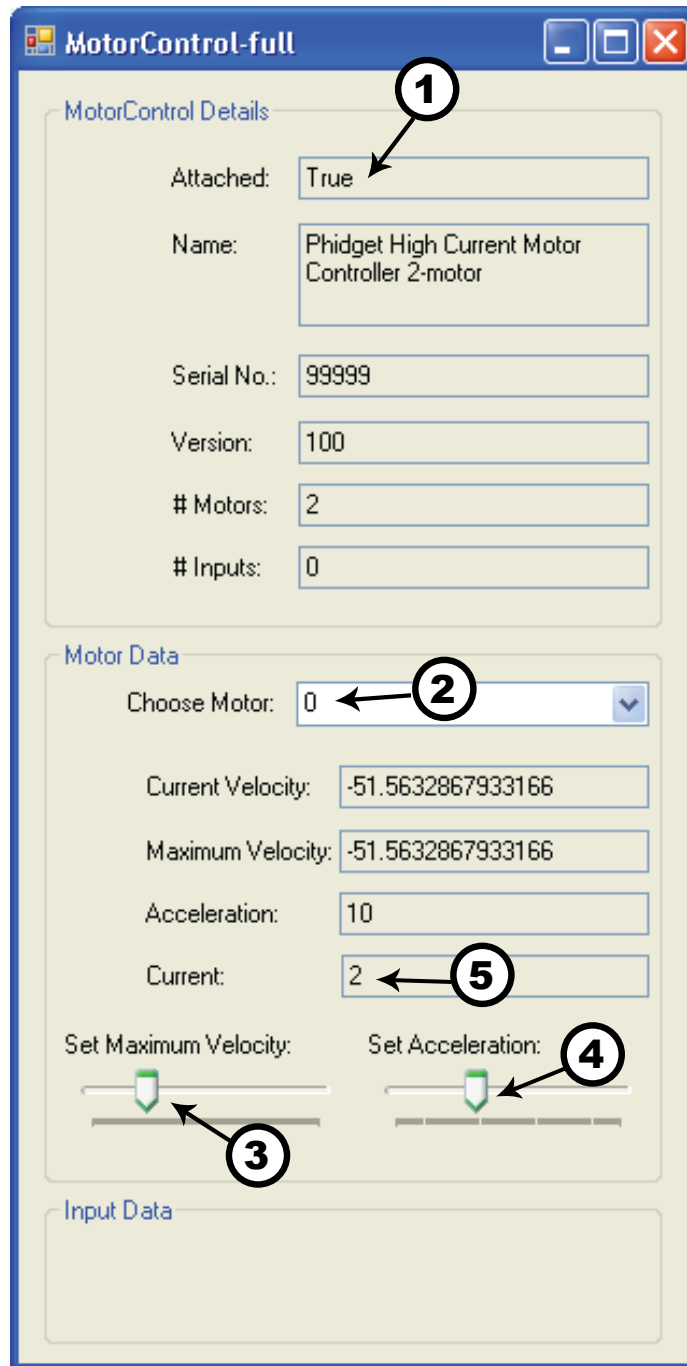
Install the Libraries and decompress the Example file.

## Testing the PhidgetMotorControl HC using Windows

- Note that some examples are not available for Linux, Mac OSX or Windows CE.
- Make sure that you have installed the libraries and decompressed your example file.



Run the program **Manager-full** to make sure that the **PhidgetMotorControl** is properly connected to your PC.



1. Run the program **MotorControl-full** and check that the box labelled Attached contains the word True.
2. Select the connected motor. If you have connected your motor at the same place as the one in the picture on page 3, it should be at position 0.
3. Move the position slider to set the maximum velocity. The maximum velocity is shown in the Maximum Velocity box, and the current motor velocity in the box above.
4. Change the acceleration by moving the slider. The value is displayed in the Acceleration box.
5. The electrical current flowing through the motor is displayed in the Current box.

# Programming a Phidget

## Where to get information

- Go to [www.phidgets.com](http://www.phidgets.com) >> [downloads](#)
- Select the Operating System and the language you want to use.
- Download the appropriate API manual and read under the **PhidgetMotorControl** heading.
- Have a look at the source code of the **PhidgetMotorControl** program.
- Have a look at the C# example below.
- Modify an existing program or write your own program from scratch.

## Simple example written in C#

```
/* - MotorControl simple -
 * This example simply creates a MotorControl object, hooks the event handlers, and opens it for connections.
 * Once a MotorControl Phidget has been attached, it will accelerate the motor to full speed in one direction,
 * then speed down and accelerate in the other direction before stopping the motor.
 * For a more detailed example, please see the MotorControl-full example.
 *
 * Please note that this example was designed to work with only one Phidget MotorControl connected.
 * For an example using multiple Phidget MotorControl, please see a "multiple" example in the
 * MotorControl Examples folder.
 *
 * Copyright 2007 Phidgets Inc.
 * This work is licensed under the Creative Commons Attribution 2.5 Canada License.
 * To view a copy of this license, visit http://creativecommons.org/licenses/by/2.5/ca/
 */

using System;
using System.Collections.Generic;
using System.Text;
using Phidgets;
using Phidgets.Events; //Needed for the MotorControl class, Phidget base classes, and the PhidgetException class
using System.Threading; //Needed for the Phidget event handling classes

namespace MotorControl_simple
{
    class Program
    {
        //Declare a MotorControl object
        static MotorControl motoControl;

        static void Main(string[] args)
        {
            try
            {
                //Initiallize the MotorControl object
                motoControl = new MotorControl();

                //Hook the basic event handlers
                motoControl.Attach += new AttachEventHandler(motoControl_Attach);
                motoControl.Detach += new DetachEventHandler(motoControl_Detach);
                motoControl.Error += new ErrorEventHandler(motoControl_Error);

                //Hook the phidget specific event handlers
                motoControl.CurrentChange += new CurrentChangeEventHandler(motoControl_CurrentChange);
                motoControl.InputChange += new InputChangeEventHandler(motoControl_InputChange);
                motoControl.VelocityChange += new VelocityChangeEventHandler(motoControl_VelocityChange);

                //open the object for MotorControl device connections
                motoControl.open();

                //Wait for a MotorControl device to be attached
                Console.WriteLine("Waiting for MotorControl to be attached...");
                motoControl.waitForAttachment();

                //Set the acceleration to 100 for this example and intialize the velocity to 0.00 (stopped)
                motoControl.motors[0].Acceleration = 100.00;
                motoControl.motors[0].Velocity = 0.00;

                //Prompt for user input to continue
                Console.WriteLine("Press any button to continue...");
                Console.Read();
            }
        }
    }
}
```

```

        //Slowly speed the motor to full speed going one direction
        double i;
        for (i = 0.00; i < 100; i++)
        {
            Thread.Sleep(100);
            motoControl.motors[0].Velocity = i;
        }

        //Slowly speed down and start accelerating in the other direction
        for (i = 100; i > -100; i--)
        {
            Thread.Sleep(100);
            motoControl.motors[0].Velocity = i;
        }

        //Stop the motor
        motoControl.motors[0].Velocity = 0.00;

        //Prompt the user for input to terminate
        Console.WriteLine("Press any key to end...");
        Console.Read();

        //User input was read so we can terminate, close the MotorControl object
        motoControl.close();

        //Set the MotorControl object to null to get it out of memory
        motoControl = null;

        //If no exceptions were thrown by this point it is ok to terminate
        Console.WriteLine("ok");
    }
    catch (PhidgetException ex)
    {
        Console.WriteLine(ex.Description);
    }
}

//Attach event hanlder...Display the serial number fo the attached MotorControl phidget
static void motoControl_Attach(object sender, AttachEventArgs e)
{
    Console.WriteLine("MotorControl {0} attached!", e.Device.SerialNumber.ToString());
}

//Detach event handler...Display the serial number of the detached MotorControl phidget
static void motoControl_Detach(object sender, DetachEventArgs e)
{
    Console.WriteLine("MotorControl {0} detached!", e.Device.SerialNumber.ToString());
}

//Error event handler...Display the error description to the console
static void motoControl_Error(object sender, ErrorEventArgs e)
{
    Console.WriteLine(e.Description);
}

//Current change event handler...Display the index and the new current value to the console
static void motoControl_CurrentChange(object sender, CurrentChangeEventArgs e)
{
    Console.WriteLine("Current Index {0} Current {1}", e.Index, e.Current);
}

//Input change event handler...Display the index and new input value to the console
static void motoControl_InputChange(object sender, InputChangeEventArgs e)
{
    Console.WriteLine("Input Index {0} Value {1}", e.Index, e.Value);
}

//Velocity change event handler...Display the motor index and the current velocity value to the
//console.
static void motoControl_VelocityChange(object sender, VelocityChangeEventArgs e)
{
    Console.WriteLine("Index {0} Velocity {1}", e.Index, e.Velocity);
}
}
}

```

## Learning more ...

- check out the forums
- check out the Phidgets projects

## Technical Section



### Brushed DC Motors

A brushed DC motor is a device that converts electricity into mechanical rotation through electromagnetism. Many variations of brushed DC motors exist: permanent magnet motors, electromagnet motors, coreless motors, linear motors... the PhidgetMotorController can be used with any of these, as well as other devices like solenoids, incandescent light bulbs, heaters, and hydraulic or pneumatic devices like pumps and valves.

### Pulse Width Modulation (PWM)

In pulse width modulation, the HIGH and LOW time of a select time period (~50 $\mu$ s in this case) is varied to produce a 0 to 100% adjustment in velocity. For example, at 20% the PWM signal is at a HIGH voltage level for 10 $\mu$ s and at a LOW voltage level (0 volts) for 40 $\mu$ s.

When a PWM signal is passed through a low-pass filter, a constant voltage results. The DC motor, in this case, acts like a low-pass filter and smoothes the PWM signal into an analog voltage. This voltage dictates the speed of the motor: a 10V control signal set to 50% PWM will be filtered at the motor to 5V, and will allow the motor half the total power it would normally see at 100% PWM.

### Using the PhidgetMotorControl with a DC Motor

The PhidgetMotorControl has been designed to be used with a variety of DC motors independent of the motor-specific velocity and torque limits. Select a motor that suits your application and falls within the MotorControl device specifications (see page 10).

To use a DC motor, first select (in software) which attached motor the PhidgetMotorControl should affect. Velocity (as a percentage out of 100) and acceleration can be controlled for each motor in both directions of rotation. The software can also display a readout of the electrical current flowing through each motor, and will fire events if a motor attempts to draw too much current (overcurrent), or if a motor running at high-current for a period of time causes hardware devices on the PhidgetMotorControl to overheat and shutdown (overtemperature).

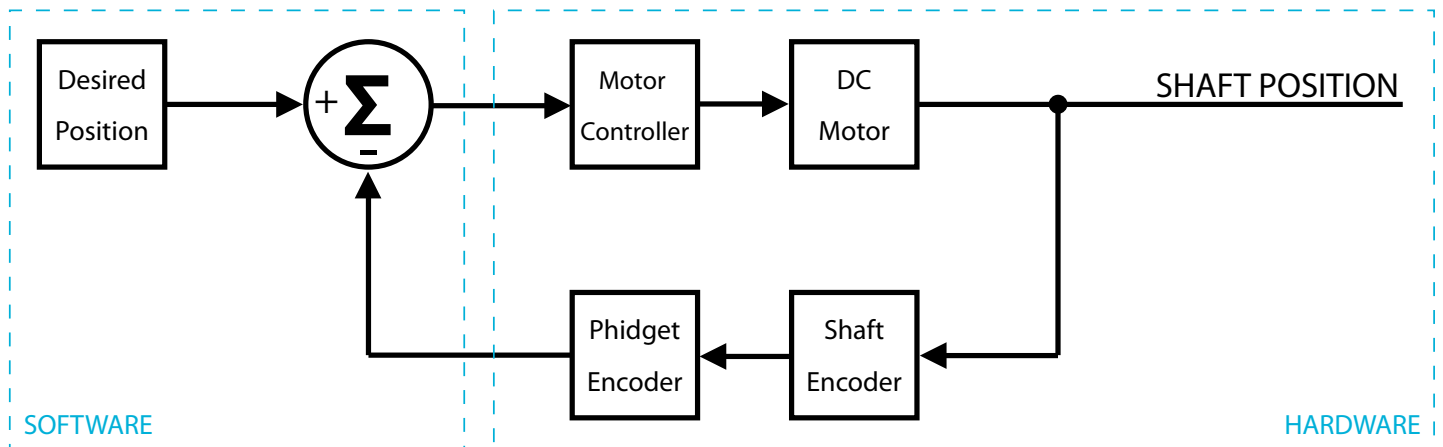
### Using the PhidgetMotorControl with Brushless DC Motors

The PhidgetMotorControl HC cannot be used with Brushless DC Motors.

## Using the PhidgetMotorControl to create a simple Position Control System

Unlike a servo motor, a DC motor does not contain the internal electronics necessary to control its shaft position. However, a control system built from simple hardware and software components can perform this task.

One of the key components that allows an output variable (such as position, velocity, acceleration) to be controlled is a device used to sense its current value. In a position control system, we require an encoder mechanically mounted to the motor shaft to tell us the current shaft position. Once this value has been quantized and is available in software, the rest of the work is in the programming (try using a PhidgetEncoder High Speed for a simple USB interface).



The block diagram above depicts the abstracted operation of a DC motor position control system. The shaft position is *fed back* to the PC through the shaft encoder and PhidgetEncoder. There it is compared with the Desired Position of the shaft (set in software) to produce an error signal which represents the difference between the desired position and the actual shaft position in the real world. The error signal is then applied as the Velocity of the PhidgetMotorController which causes the DC motor to rotate towards the target position. As the shaft nears the desired angle, the error signal moves closer to zero and less power is applied to the motor, causing the shaft to slow and stop at the desired position.

## Proportional, Integral and Derivative (PID) Control

Depending upon the motor and the load it is driving, the shaft may slow too much before reaching the target position and subsequently not arrive there at all. In this case, some *amplification* or *proportional control* of the error signal may be necessary. This is accomplished by simply multiplying the error signal by a static value before applying it to the PhidgetMotorController. This will serve to speed up the motor response, but too much amplification may cause the shaft to *overshoot* the target position or *oscillate* around it.

If the motor shaft approaches the desired angle at an acceptable speed but there still remains an error above or below the wanted position, what is required is known as *integral control*. The integral value in software is simply the running sum of all the error signals calculated over time. This value should also be multiplied by a static number to scale it properly, then added to the proportional control before being applied to the PhidgetMotorController.

If the shaft reaches the desired position but overshoots and oscillates around it before returning to the correct angle, use some *derivative control*. The derivative is the difference in value between the current and last calculated error signals (the rate of change), and is used to predict when the shaft position nears its target. Like integral, this value should be scaled then added to the proportional control value before being applied to the PhidgetMotorController.

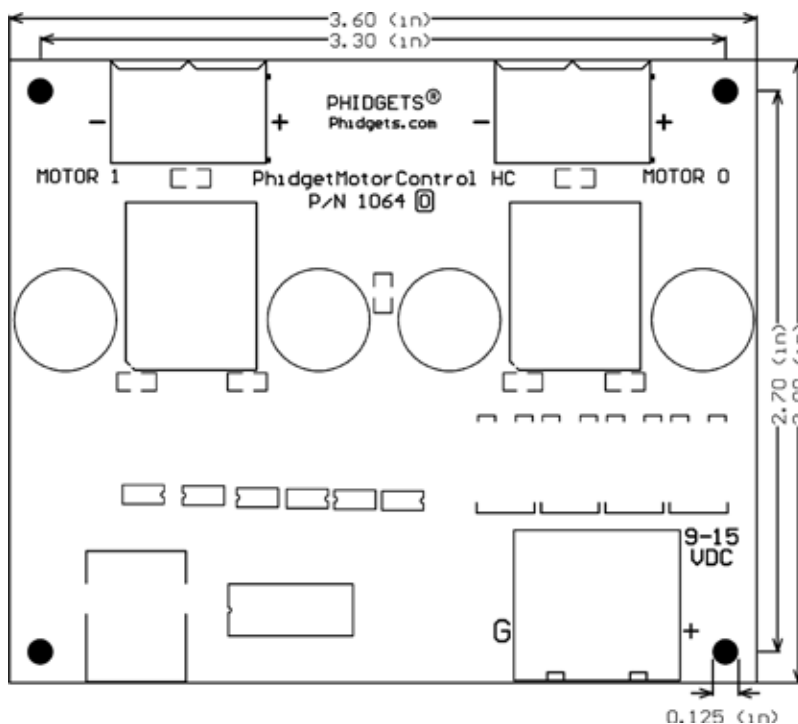
## Device Specifications

Output Controller Update Rate	50 updates / second
Velocity Resolution	1.5%
Acceleration Resolution	1.5% Velocity / Second <sup>2</sup>
Acceleration Limit (-100% to +100% velocity)	120mS
PWM Frequency	20kHz
Minimum Power Supply Voltage	9V
Maximum Power Supply Voltage	15V
Continuous Motor Current	14A
Peak Motor Current - 60 seconds	19A
Peak Motor Current - 10 seconds	25A
Peak Motor Current - 2 seconds	32A
Motor Overcurrent Trigger on Load	< 100mΩ (50A typical)
USB-Power Current Specification	500mA max
Device Quiescent Current Consumption	20mA
Device Active Current Consumption	20mA max

Note: current from USB supply is not available for motors

## Mechanical Drawing

1:1 scale



## Product History

Date	Product Revision	Comment
October 2007	1064_0	Product Release