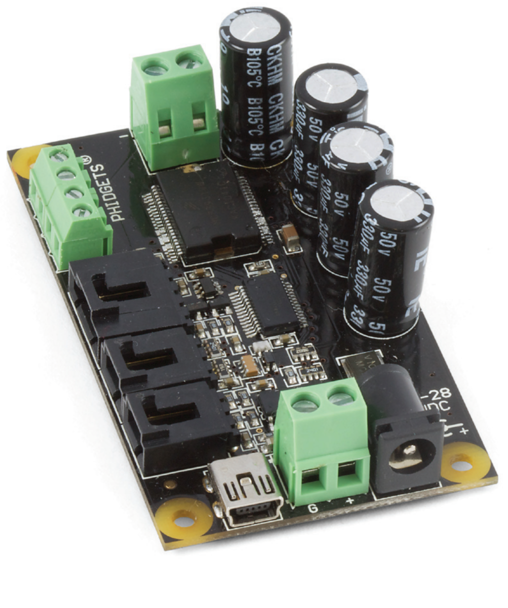
# Product Manual

## 1065 - PhidgetMotorControl 1-Motor



**Phidgets 1065 - Product Manual**

**For Board Revision 0**

# Contents

## Product Features

- Controls the direction, velocity and acceleration of one DC motor
- Motors are powered from an external power supply (9 to 28VDC)
- Built in sensing to support software-based control loops (2 digital inputs, 2 analog inputs, 1 encoder input, Back-EMF measurement, Current sensing)
- Provides over-current, over-voltage and over-temperature protection
- Connects directly to a computer’s USB port

## Programming Environment

**Operating Systems:** Windows 2000/XP/Vista/7, Windows CE, Linux, and Mac OS X

**Programming Languages (APIs):** VB6, VB.NET, C#.NET, C++, Flash 9, Flex, Java, LabVIEW, Python, Max/MSP, and Cocoa.

**Examples:** Many example applications for all the operating systems and development environments above are available for download at www.phidgets.com >> Programming.

## Connection

The board connects directly to a computer’s USB port.

# Getting Started

## Checking the Contents

### You should have received:

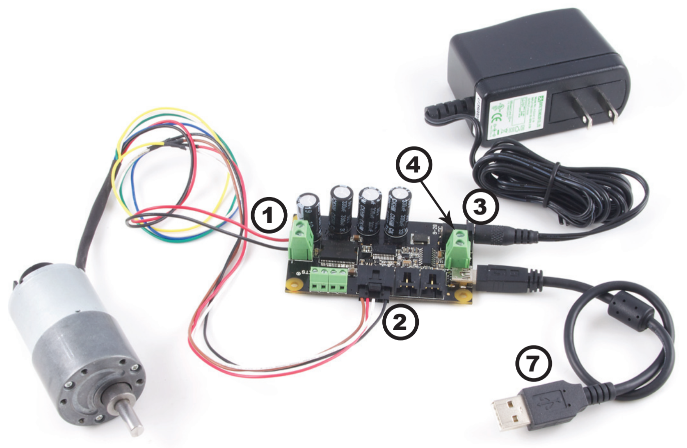
- A PhidgetMotorControl 1-Motor
- A Mini-USB cable

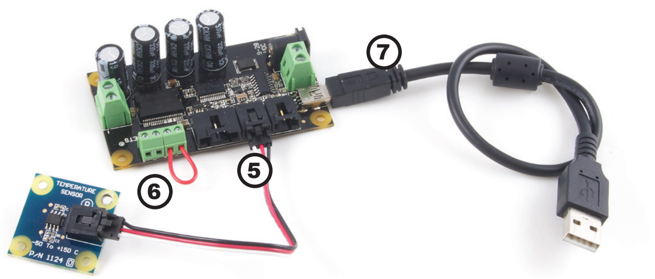
### In order to fully test your new Phidget you will also need:

- A 9 to 28VDC power supply
- A DC motor with integrated encoder
- An Analog Sensor (we are using an a temperature sensor)
- A piece of wire to test the digital inputs

## Connecting all the pieces

1. Connect the Motor to the PhidgetMotorControl
2. Connect the Encoder
3. Plug in a power supply using the barrel connector.
4. You can also connect a power supply to the Terminal Block. Be sure to observe correct polarity.
5. Connect your sensor to an analog input. We are using a Phidget Slider-60.
6. Connect one end of the wire to the Ground connector and the other end to connector 0.
7. Connect the MotorControl to your computer using the USB cable.





## Testing Using Windows 2000/XP/Vista/7

### Downloading the Phidgets drivers

Make sure that you have the current version of the Phidget library installed on your PC. If you don't, do the following:

Go to www.phidgets.com >> Drivers

Download and run Phidget21 Installer (32-bit, or 64-bit, depending on your PC)

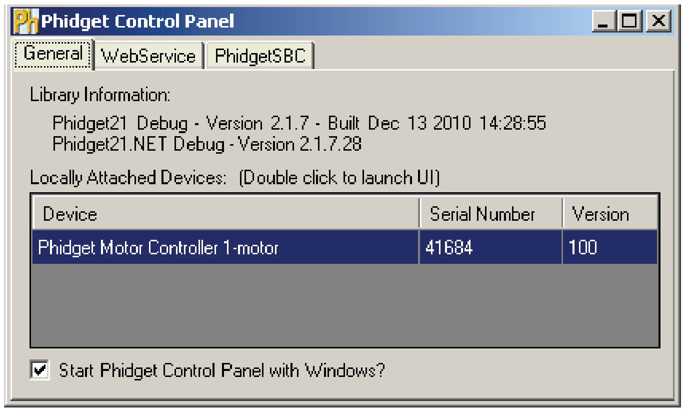
You should see the Ph icon on the right hand corner of the Task Bar.

## Running Phidgets Sample Program

Double clicking on the Ph icon loads the Phidget Control Panel; we will use this program to make sure that your new Phidget works properly.

The source code for the motorcontrol-full sample program can be found under C# by clicking on www.phidgets.com >> Programming.

Double Click on the Ph icon to activate the Phidget Control Panel and make sure that the **Phidget Motor Controller 1-Motor** is properly attached to your PC.

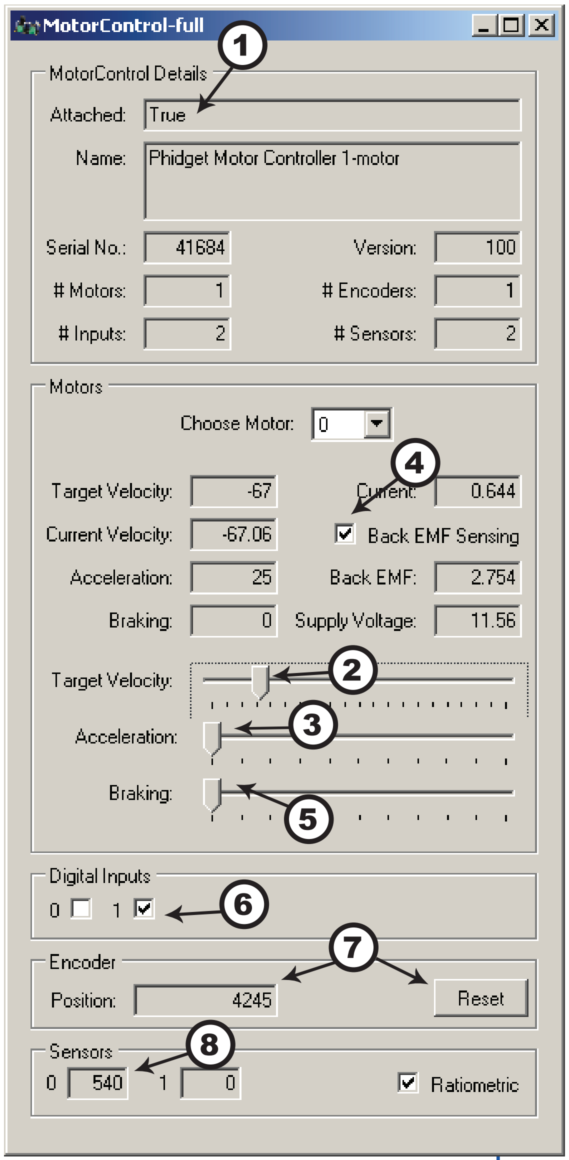


Library Information:
Phidget21 Debug - Version 2.1.7 - Built Dec 13 2010 14:28:55
Phidget21.NET Debug - Version 2.1.7.28

Locally Attached Devices: (Double click to launch UI)

| Device | Serial Number | Version |
|---|---|---|
| Phidget Motor Controller 1-motor | 41684 | 100 |

☑ Start Phidget Control Panel with Windows?



MotorControl Details
Attached: True
Name: Phidget Motor Controller 1-motor
Serial No.: 41684 Version: 100
# Motors: 1 # Encoders: 1
# Inputs: 2 # Sensors: 2

Motors
Choose Motor: 0
Target Velocity: -67 Current: 0.644
Current Velocity: -67.06 ☑ Back EMF Sensing
Acceleration: 25 Back EMF: 2.754
Braking: 0 Supply Voltage: 11.56
Target Velocity:
Acceleration:
Braking:

Digital Inputs
0 ☐ 1 ☑

Encoder
Position: 4245

Sensors
0 540 1 0 ☑ Ratiometric

1. Double Click on **Phidget Motor Controller 1-Motor** in the Phidget Control Panel to bring up MotorControl-full and check that the box labelled Attached contains the word True.
2. Move the target velocity slider to a target velocity setting. The Current Velocity is displayed in the Current Velocity Box. The motor will accelerate until the current velocity is equal to the target velocity.
3. You can change the acceleration by using the acceleration slider.
4. Turn On Back EMF Sensing.
5. When the motor is stopped, you can create some shaft resistance by using the braking slider.
6. Test the digital input by disconnecting the wire end connected to the digital input connector. The tick mark in the box will go away.
7. You can read the Encoder Position and reset it to 0 by clicking on the Reset button.
8. The sensors box displays the SensorValue of the Analog Inputs. Click on the Ratiometric box if your sensor is ratiometric.

## Testing Using Mac OS X

- Click on System Preferences >> Phidgets (under Other) to activate the Preference Pane
- Make sure that the PhidgetMotorControl 1-Motor is properly attached.
- Double Click on Phidget MorotControl 1-Motor in the Phidget Preference Pane to bring up the MotorControl-full Sample program. This program will function in a similar way as the Windows version.

## If you are using Linux

There are no sample programs written for Linux.

Go to www.phidgets.com >> Drivers

Download Linux Source

- Have a look at the readme file
- Build Phidget21

The most popular programming languages in Linux are C/C++ and Java.

Notes:

Many Linux systems are now built with unsupported third party drivers. It may be necessary to uninstall these drivers for our libraries to work properly.

Phidget21 for Linux is a user-space library. Applications typically have to be run as root, or udev/hotplug must be configured to give permissions when the Phidget is plugged in.

## If you are using Windows Mobile/CE 5.0 or 6.0

Go to www.phidgets.com >> Drivers

Download x86, ARMV4I or MIPSII, depending on the platform you are using. Mini-itx and ICOP systems will be x86, and most mobile devices, including XScale based systems will run the ARMV4I.

The CE libraries are distributed in .CAB format. Windows Mobile/CE is able to directly install .CAB files.

The most popular languages are C/C++, .NET Compact Framework (VB.NET and C#). A desktop version of Visual Studio can usually be configured to target your Windows Mobile Platform, whether you are compiling to machine code or the .NET Compact Framework.

# Programming a Phidget

Phidgets' philosophy is that you do not have to be an electrical engineer in order to do projects that use devices like sensors, motors, motor controllers, and interface boards. All you need to know is how to program. We have developed a complete set of Application Programming Interfaces (API) that are supported for Windows, Mac OS X, and Linux. When it comes to languages, we support VB6, VB.NET, C#.NET, C, C++, Flash 9, Flex, Java, LabVIEW, Python, Max/MSP, and Cocoa.

## Architecture

We have designed our libraries to give you the maximum amount of freedom. We do not impose our own programming model on you.

To achieve  this goal we have implemented the libraries as a series of layers with the C API at the core surrounded by other language wrappers.

## Libraries

The lowest level library is the C API.  The C API can be programmed against on Windows, CE, OS X and Linux.  With the C API, C/C++, you can write cross-platform code.  For systems with minimal resources (small computers), the C API may be the only choice.

The Java API is built into the C API Library.  Java, by default is cross-platform - but your particular platform may not support it (CE).

The .NET API also relies on the C API.  Our default .NET API is for .NET 2.0 Framework, but we also have .NET libraries for .NET 1.1 and .NET Compact Framework (CE).

The COM API relies on the C API.  The COM API is programmed against when coding in VB6, VBScript, Excel (VBA), Delphi and Labview.

The ActionScript 3.0 Library relies on a communication link with a PhidgetWebService (see below).  ActionScript 3.0 is used in Flex and Flash 9.

## Programming Hints

- Every Phidget has a unique serial number - this allows you to sort out which device is which at runtime.  Unlike USB devices which model themselves as a COM port, you don't have to worry about where in the USB bus you plug your Phidget in.  If you have more than one Phidget, even of the same type, their serial numbers enable you to sort them out at runtime.
- Each Phidget you have plugged in is controlled from your application using an object/handle specific to that phidget.  This link between the Phidget and the software object is created when you call the .OPEN group of commands.  This association will stay, even if the Phidget is disconnected/reattached, until .CLOSE is called.
- For full performance, the Phidget APIs are designed to be used in an event driven architecture.  Applications that require receiving all the data streaming from the device will have to use event handlers, instead of polling.

## Networking Phidgets

The PhidgetWebService is an application written by Phidgets Inc. which acts as a network proxy on a computer.  The PhidgetWebService will allow other computers on the network to communicate with the Phidgets connected to that computer.  ALL of our APIs have the capability to communicate with Phidgets on another computer that has the PhidgetWebService running.

The PhidgetWebService also makes it possible to communicate with other applications that you wrote and that are connected to the PhidgetWebService, through the PhidgetDictionary object.

## Documentation

### Programming Manual

The Phidget Programming Manual documents the Phidgets software programming model in a language and device unspecific way, providing a general overview of the Phidgets API as a whole. You can find the manual at www.phidgets.com >> Programming.

### Getting Started Guides

We have written Getting Started Guides for most of the languages that we support. If the manual exists for the language you want to use, this is the first manual you want to read. The Guides can be found at www.phidgets.com >> Programming, and are listed under the appropriate language.

### API Guides

We maintain API references for COM (Windows), C (Windows/Mac OSX/Linux), Action Script, .Net and Java. These references document the API calls that are common to all Phidgets. These API References can be found under www.phidgets.com >> Programming and are listed under the appropriate language. To look at the API calls for a specific Phidget, check its Product Manual.

## Code Samples

We have written sample programs to illustrate how the APIs are used.

Due to the large number of languages and devices we support, we cannot provide examples in every language for every Phidget. Some of the examples are very minimal, and other examples will have a full-featured GUI allowing all the functionality of the device to be explored. Most developers start by modifying existing examples until they have an understanding of the architecture.

Go to www.phidgets.com >> Programming to see if there are code samples written for your device. Find the language you want to use and click on the magnifying glass besides "Code Sample". You will get a list of all the devices for which we wrote code samples in that language.

## API for the PhidgetMotorControl

We document API Calls specific to this product in this section. Functions common to all Phidgets and functions not applicable to this device are not covered here. This section is deliberately generic. For calling conventions under a specific language, refer to the associated API manual. For exact values, refer to the device specifications.

### Properties

**int MotorCount() [get] : Constant = 1**

Returns the number of Motors that can be controlled by this PhidgetMotorControl.

**double Velocity (int MotorIndex) [get,set]**

Velocity is the percentage of time the motor is being powered for. The PhidgetMotorControl rapidly switches power to the motor on/off. Velocity can be set between –100 and +100. –100 corresponds to the motor being driven 100% of the time in reverse, +100 driven 100% of the time forward. When velocity is 0, the motor is controlled by the `Braking` property, which defaults to 0%.

**double Acceleration (int MotorIndex) [get,set]**

Returns how fast a motor will be accelerated between given velocities. The valid range is between AccelerationMax and AccelerationMin. Acceleration is in %(change in velocity)/$s^2$.

**double AccelerationMax (int MotorIndex) [get] : Constant = 6250%/$s^2$**

Returns the maximum acceleration that a motor will accept, or return, in %(change in velocity)/$s^2$.

**double AccelerationMin (int MotorIndex) [get] : Constant = 24.51%/$s^2$**

Returns the minimum acceleration that a motor will accept, or return, in %(change in velocity)/$s^2$.

**double Current(int MotorIndex) [get]**

Gets the current usage of a motor, in Amps.

**bool BackEMFSensingState(int MotorIndex) [get,set]**

Gets/Sets the back-EMF sensing state for a motor. When back-EMF sensing is enabled, the motor will coast (free-wheel) 5% of the time while the back EMF measurement is taken (800us every 16ms). Therefore at a velocity of 100%, the motor is only powered 95% of the time. By default, this is disabled.

**double BackEMF(int MotorIndex) [get]**

Gets the back-EMF measurement for a motor, in Volts. This is only available if `BackEMFSensingState` is set to true. Back-EMF generally corresponds roughly to true motor velocity - see technical section for details.

**double Braking(int MotorIndex) [get,set]**

Gets/sets the braking amount for a motor at rest, with a range of 0-100%. Braking is only active when the motor velocity is 0. By default, braking is 0%, allowing the motor to coast (free-wheel). The holding strength of a braked motor depends on the motor, but is generally quite low.

**double SupplyVoltage() [get]**

Gets the board supply voltage, in Volts.

**int InputCount() [get] : Constant = 2**

Returns the number of digital inputs.

**bool InputState(int InputIndex) [get]**

Returns the state of a digital input. True means that the input is activated, and False indicated the default state.

**int EncoderCount() [get] : Constant = 1**

Returns the number of encoder inputs.

**int EncoderPosition(int EncoderIndex) [get,set]**

Sets/gets the current encoder position of an encoder. Note that precision is 1/4 of that supported by the PhidgetEncoders 1047 and 1057 - see the technical section for more information.

**int SensorCount() [get] : Constant = 2**

Returns the number of analog sensor inputs.

**int SensorValue(int SensorIndex) [get]**

Gets the current value for a sensor input. Range is 0-1000.

**int SensorRawValue(int SensorIndex) [get]**

gets the raw 12-bit value for a sensor input. Range is 0-4096.

**bool Ratiometric() [get,set]**

Gets/sets the ratiometric state for the analog sensor inputs. Defaults to true.

## Events

**OnVelocityChange(int MotorIndex, double Velocity) [event]**

An event issued when the velocity a motor is being driven at changes.

**OnCurrentChange(int MotorIndex, double Current) [event]**

An event issued whenever the current consumed by a motor changes.

**OnCurrentUpdate(int MotorIndex, double Current) [event]**

An event containing current consumption information for a motor, which is issued at a set interval of 8ms. This is generally used for PID torque control.

**OnBackEMFUpdate(int MotorIndex, double BackEMF) [event]**

An event containing the back-EMF value for a motor, which is issued at a set interval of 16ms, when back-EMF sensing is enabled. This is generally used for PID velocity control.

**OnInputChange(int InputIndex, bool State) [event]**

An event issued when the state of a digital input changes.

**OnEncoderPositionChange(int EncoderIndex, int Time, int PositionChange) [event]**

An event issued when the position of an encoder changes. `Time` is in 1/3ms and represents the amount of time in which `PositionChange` counts occurred.

**OnEncoderPositionUpdate(int EncoderIndex, int PositionChange) [event]**

An event containing position change information for an encoder, which is issued at a set interval of 8ms, regardless of whether the position has changed. This is generally used for PID velocity and/or position control.

**OnSensorUpdate(int SensorIndex, int SensorValue) [event]**

An event containing sensor value information for sensors plugged into the Analog Inputs, which is issued at a set interval of 8ms. This may be used for PID control loops depending on the type of sensor being used.

**OnError(int ErrorCode, String ErrorDescription) [event]**

The PhidgetMotorControl 1-Motor will throw error events under certain circumstances:

**ErrorCode = EEPHIDGET_WRAP** - The position value for an encoder is wrapping around (between 2147483647 and -2147483648).

**ErrorCode = EEPHIDGET_PACKETLOST** - A packet of data has been lost. This should be taken into consideration when the Update events are being used, as a gap will be present in the data - 8ms/packet lost error event.

**ErrorCode = EEPHIDGET_OVERTEMP** - An over-temperature, or short-circuit condition has occurred. The output will be clamped at 5-8 Amps.

**ErrorCode = EEPHIDGET_BADPOWER** - The supply voltage is too low or too high. This is thrown if the voltage is <=7V, or >=34V. There are several different messages depending on how far out of spec the voltage is. Quick load/velocity changes can cause spike in the supply voltage - if these spikes exceed 40V, the motor is automatically placed in 100% braking - braking ends when the supply voltage falls to <38V.

When error states have ended, there will be an error event with the EEPHIDGET_OK code.

See the ErrorDescription string for specific error details.

# Technical Section

## Brushed DC Motors

Brushed DC Motors are very simple to understand, but very difficult to control precisely. By applying a voltage, or pulsing a voltage rapidly, at the terminals of the motor, current flows through the motor, and it will begin rotating. Depending on the direction of the current, the motor will rotate clockwise or counterclockwise.

The brushes inside the motor automatically flip the direction of current inside the motor, allowing the motor to rotate continuously without the external precise control required for stepper motors. There are two downsides of letting the motor 'sequence' itself.

1. It's not possible to control the timing - the motor will rotate as fast as possible, given it's construction, the voltage applied, and the load it is driving.
2. The brushes generate sparks and Electromagnetic Interference, and have a fairly limited lifetime. On cheaper motors, the brushes are not replaceable, and when the brushes are worn out, the motor is garbage.

Given that the speed of the DC motor depends on its construction, the load it is driving, and the voltage applied, the only practical way to change the speed of the motor is by changing the applied voltage. The 1065 changes the effective voltage by changing the percentage of time the full supply voltage is applied to the motor. By switching the voltage very quickly (a technique called PWM), the controller is made smaller, more efficient, and cheaper.

## Achieving rough control of DC Motors

Depending on your application, you may not need precise control of the motor. If your power supply voltage and load are relatively constant, and you can tolerate some small variation in speed, simply tuning the Velocity in software to get the results you want will work. Using a DC Motor as a fan is an example.

### Back-EMF sensing

Rough control of actual motor speed can be achieved automatically in software by using the Back EMF property. The 1065 very quickly removes power from the motor, and measures the generated voltage. As the motor spins faster, the generated voltage increases, and is quite linear with the rotation speed. The relationship between motor speed and Back EMF depends on the motor construction. A good clue is the rated speed at the rated supply voltage. For example, a DC Motor that is specified for 4000 RPM at 12V will generate roughly 12 Volts when it spins at 4000 RPM. At 2000 RPM, it will generate roughly 6 Volts.

There is a fairly large amount of noise in BackEMF measurements, so backEMF should not be used for precise velocity measurements.

### Current Sensing

Rough control of motor torque can be achieved in software by using the Current Sense property and events. The current in the coils of a DC motor will roughly correspond to the torque it is generating. Torque is difficult for many people to understand - a motor accelerating quickly from stop will experience a huge surge of current as the motor pushes against it's own inertia and accelerates the load. Unless the motor is loaded down, when it is spinning at a stable speed, it is experiencing very little torque, and consuming very little current.

## Achieving precise control of DC Motors

### Encoder Input

The true acceleration, velocity and number of rotations of a DC motor is controlled by using an Encoder to measure the movement of the motor shaft. The encoder rotates with the motor, and produces pulses. An encoder with very high resolution will produces thousands of precisely spaced pulses with a single rotation - an encoder with lower resolution will produce a dozen pulses. Please note we are referring to Quadrature (incremental) Encoders.

By having electronics counting the number of pulses, and the time between pulses, it's straightforward to calculate the velocity of the motor, and how far it has rotated.  The 1065 has an encoder input, and will stream the pulse and timing information to your application, where a software control loop modifies the motor's target velocity.

The 1065 does not support the most accurate method of measuring encoders.  Our PhidgetEncoder products (1047 and 1057) measure PPR(pulses per revolution), which has the ability to count every pulse change in the encoder. The 1065 measures CPR(counts per revolution).  PPR has 4 times the resolution of CPR (ie there are 4 pulses for every 1 count). So the accuracy on the 1065 encoder is 1/4 as precise as 1047/1057. However, CPR are still very accurate, and makes the encoder feedback your best method for velocity/position feedback control. Use high CPR encoders (500 - 1000) for maximum precision.

### System Level Control

Control can also achieved by stepping back from the motor, and viewing the system as a whole.  If the system is a mechanism to close a door, the only feedback required could be a switch on either extreme of the range of motion of the mechanism.

## Control Loops - Overview

In the context of the 1065, A software control loop is an algorithm in your software application that receives sensor data, and uses that data to calculate how hard and in what direction it should drive the motor to achieve it's goal. Control Theory is an extremely complex academic field, and in most cases there are simpler methods.

If your application is simple - opening a door with two limit switches, for instance, no fancy algorithm is required.

### PID Loop

To control a motor's position and/or velocity under varying load conditions - a common application - the PID loop is a place to start.  The performance of the system is determined by tuning three 'magic numbers' for the Proportional, Integral and Derivative terms.  If you understand Control Theory, these magic numbers can be calculated.  For  the rest of us, the typical approach is to first tune the Proportional Term, increase the Integral term if necessary, and usually leave the Derivative term at zero.

The proportional term sets how strongly the system responds immediately to the difference between it's target goal, and it's current measured performance.  For example, if the cruise control on a car is set to 100 kph, and the speedometer indicates the speed is 90 kph, the proportional term sets how much throttle is applied.

The integral term allows the control loop to respondly more strongly over time to situations where the proportional term is unable to reach the target.  The integral term can be thought of as a slowly building impatience within the control loop.

The derivative term is used to prevent the control loop from overshooting the target.  In practice, it requires very precise measurements to calculate how quickly the loop is approaching the target.  In many systems, the measurements are not accurate enough, and the derivative term only causes system instability.

### Kalman Filter

More sophisticated control loops, like Kalman filters, are built with specialized knowledge of the exact application.  In fact, a Kalman filter is not so much a filter as it is a mathematical model of the application, incorporating the laws of physics, and expectations of how the system should behave and respond.  With the cruise control example, if the wheels spin on black ice, the PID loop will not respond appropriately.  An experienced driver, and a much more sophisticated control algorithm like a Kalman filter will realize that something unusual is happening.

## Impact of latency on control loops.

There is a time delay required to measure the system (encoders, Back EMF, current, sensors, etc.), receive this data in your application, and send the new target velocity to the Motor Controller.  If the system being controlled by the control loop can respond as quickly, or quicker than this time delay, the system will be plagued by oscillations.  The 1065 was designed to minimize the delays as much as possible, but very fast responding control systems will require extra care.

## Over Power shutoff

The 1065 has a polarity protection diode built in to protect the controller if the power supply is hooked up backwards.  This diode has the effect of trapping energy generated by the motor in the 1065.  When the motor operates as a generator, the supply voltage begins to increase.  This is continually monitored, and if the Supply Voltage goes above 40 volts, the motor will be put into a braking mode to protect the controller.  This mode is turned off when Supply Voltage drops below 38V.

## High Current Applications

Since 1065 is rated for 5A continuous current, proper heat dissipation techniques should be taken if getting to the upper limit.  The 1065 is protected against over temperature, and there is no problem if over temperature happens occasionally.  If the 1065 runs permanently hot, it’s lifespan will be reduced.

If the 1065 is enclosed, or operated above normal room temperature, heat dissipation will require more effort, or it will have to be de-rated.

## Feedback Example

Here is an example of using the 1065 with a linear actuator providing feedback through an integrated potentiometer. The actuator’s position is fed back through the analog input and can be used by the software application.

## Analog Inputs

### Using the Analog Inputs with Sensors provided by Phidgets

Analogs Inputs are used to interface many different types of sensors. Each Analog Input provides power (Nominal +5VDC), ground, and an analog voltage return wire driven by the sensor to some voltage. The 1065 continuously measures this return voltage and reports it to the application.

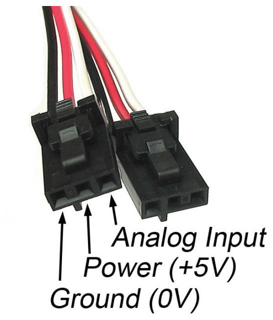
Analog Inputs are used to measure continuous quantities, such as temperature, humidity, position, pressure, etc. Phidgets offers a wide variety of sensors that can be plugged directly into the board using the cable included with the sensor.

### Using the Analog Inputs with your own sensors

For users who wish to interface their own sensors, we describe the Analog Inputs here.

### Mechanical

Each Analog Input uses a 3-pin, 0.100 inch pitch locking connector. Pictured here is a plug with the connections labeled. The connectors are commonly available - refer to the Table below for manufacturer part numbers.



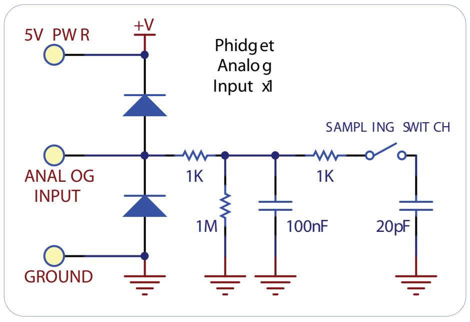
| Cable Connectors | | |
|---|---|---|
| Manufacturer | Part Number | Description |
| Molex | 50-57-9403 | 3 Position Cable Connector |
| Molex | 16-02-0102 | Wire Crimp Insert for Cable Connector |
| Molex | 70543-0002 | 3 Position Vertical PCB Connector |
| Molex | 70553-0002 | 3 Position Right-Angle PCB Connector (Gold) |
| Molex | 70553-0037 | 3 Position Right-Angle PCB Connector (Tin) |
| Molex | 15-91-2035 | 3 Position Right-Angle PCB Connector - Surface Mount |

Note: Most of the above components can be bought at www.digikey.com

### Electrical

The maximum total current consumed by all Analog Inputs should be limited to 400mA.

The analog measurement is represented in the software through the SensorValue as a value between 0 and 1000. A sensor value of 1 unit represents a voltage of approximately 5 millivolts. The RawSensorValue property brings out a 12-bit value (0-4095) for users who require maximum accuracy. Please note that the sampling is actually done with an oversampled 10-bit ADC, but reported as a 12-bit value to allow future expansion.



### Ratiometric Configuration

The group of Analog Inputs can be collectively set to Ratiometric mode from software using the Ratiometric property. If you are using a sensor whose output changes linearly with variations in the sensor's supply voltage level, it is said to be ratiometric. Most of the sensors sold by Phidgets are ratiometric (this is specified on the web product page and in the sensor's product manual).

Setting Ratiometric causes the reference to the internal Analog to Digital Converter to be set to the power supply voltage level. When Ratiometric is enabled, the maximum voltage returned on the Analog Input should be the +5V nominal power provided by the PhidgetInterfaceKit.

## Non-Ratiometric Configuration

If Ratiometric is false, the ADC reference is set to a 5.0V 0.5% stable voltage reference. The maximum voltage returned on the Analog Input should be maximum 5.0V. Note that the Analog Input power supply voltage is not affected by the setting of the Ratiometric property.

## Factors that can affect Accuracy

**High Output Impedance** - Sensors that have a high output impedance will be distorted by the 900K input impedance of the Analog Input. If your output impedance is high, it is possible to correct for this distortion to some extent in your software application.

**Power Consumption** - Sensor cables have some resistance, and the power consumption of the sensor will cause the sensor to have a slightly different ground from the Analog Input on the PhidgetInterfaceKit. The more power consumed by the sensor, and the longer the sensor cable, the more pronounced this effect will be.

**Intrinsic Error In Sensors** - For many sensors, the error is quite predictable over the life of the sensor, and it can be measured and calibrated out in software.

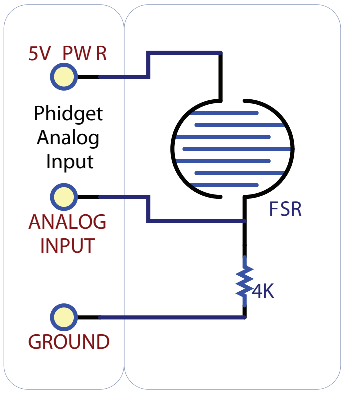
**Non-Ratiometric Configuration** - Voltage Reference error. The 5.0VDC voltage reference is accurate to 0.5%. This can be a significant source of error in some applications, but can be easily measured and compensated for.

## Connecting non-Phidget devices to the Analog Inputs

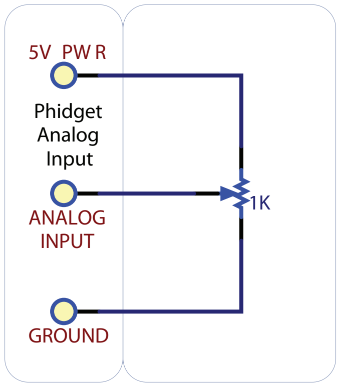
Here are some circuit diagrams that illustrate how to connect various non Phidgets devices to the analog inputs on your Phidget.

### Sensing the value of a variable resistance sensor

In this diagram, an FSR (Force Sensitive Resistor) is shown.

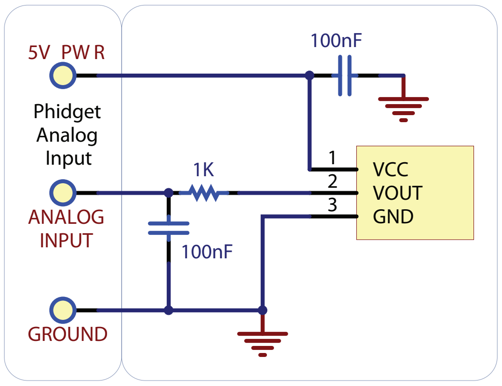


### Sensing the position of a potentiometer



## Interfacing to an arbitrary sensor

Note the use of power supply decoupling and the RC Filter on the output. The RC filter also prevents VOUT from oscillating on many sensors



## Non Phidgets Sensors

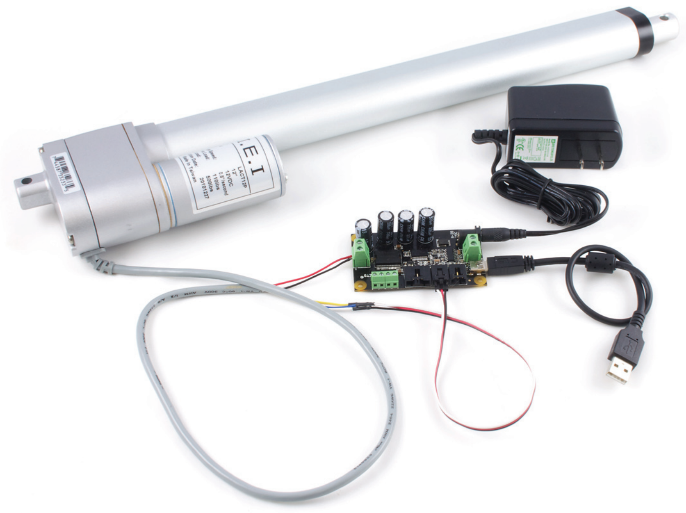
In addition to Phidgets sensors, any sensor that returns a signal between 0 and 5 volts can be easily interfaced. Here is a list of interesting sensors that can be used with the 1065. Note: these sensors are not "plug & play" like the sensors manufactured by Phidgets.

**Analog Sensors**

| **Manufacturer** | **Part Number** | **Description** |
|---|---|---|
| MSI Sensors | FC21/FC22 | Load cells - measure up to 100lbs of force |
| Humirel | HTM2500VB | Humidity sensors |
| Measurement Specialties | MSP-300 | Pressure sensors - ranges up to 10,000 PSI |
| Freescale Semiconductor | MPXA/MPXH | Gas Pressure Sensors |
| Allegro | ACS7 series | Current Sensors - ranges up to 200 Amps |
| Allegro | A1300 series | Linear Hall Effect Sensors - to detect magnetic fields |
| Analog | TMP35 TMP36 TMP37 | Temperature Sensor |
| Panasonic | AMN series | Motion Sensors |
| Honeywell | FS01, FS03 | Small, accurate Piezo-resistive load cells |
| AllSensors-Europe | BARO-A-4V | Barometric Pressure Sensor - 600 to 1,100 mbar |

Note: Most of the above components can be bought at www.digikey.com

Here is an example of using the 1065 with a linear actuator providing feedback through an integrated potentiometer. The actuator's position is fed back through the analog input and can be used by the software application.



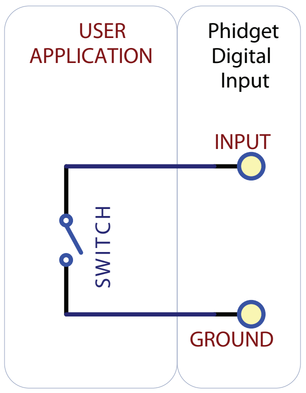
## Digital Inputs

### Using the Digital Inputs

Here are some circuit diagrams that illustrate how to connect various devices to the digital inputs on your Phidget.

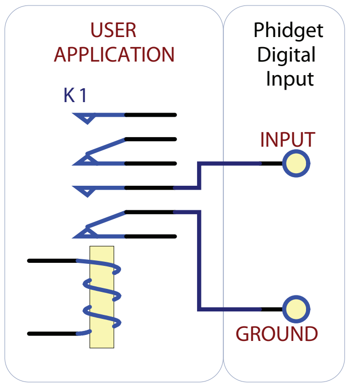
#### Wiring a switch to a Digital Input

Closing the switch causes the digital input to report TRUE.



#### Monitoring the position of a relay

The relay contact can be treated as a switch, and wired up similarly.  When the relay contact is closed, the Digital Input will report TRUE.



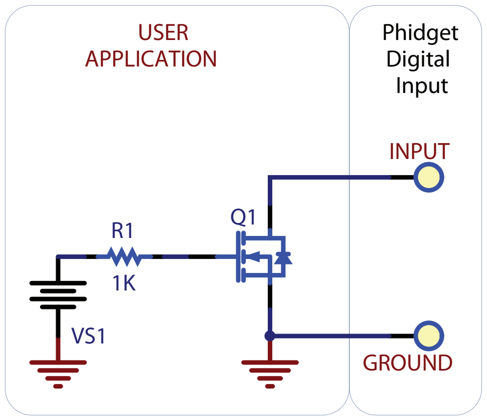
#### Detecting an external Voltage with an N-Channel MOSFET

A MOSFET can be used to detect the presence of an external voltage. The external voltage will turn on the MOSFET, causing it to short the Digital Input to Ground.

If the MOSFET is conducting > 280μA, the Digital Input is guaranteed to report TRUE.

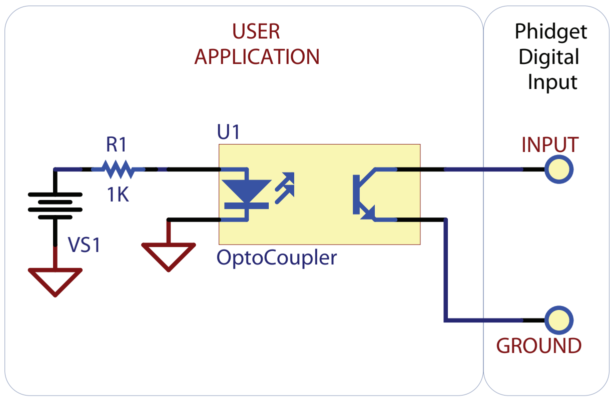
If the MOSFET is conducting < 190μA, the Digital Input is guaranteed to report FALSE.

The voltage level required to turn on the MOSFET depends on the make of of MOSFET you are using.  Typical values are 2V-6V.



## Isolating a Digital Input with an Optocoupler

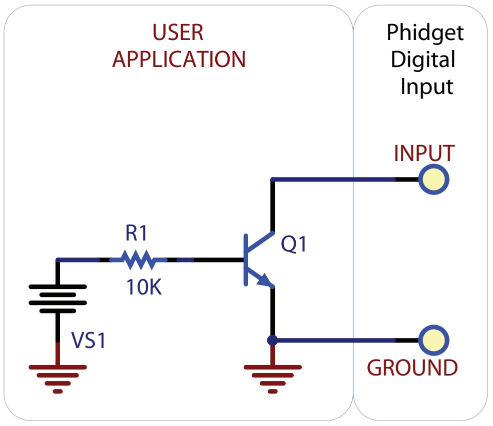
When driving current through the LED, the Digital Input will report TRUE. The amount of current required will depend on the optocoupler used. Design to sink at least 280μA to cause the digital input to report TRUE, and less than 190μA to report FALSE.



## Detecting an external Voltage with an NPN Transistor

This circuit can be used to measure if a battery is connected, or if 12V (for example) is on a wire.

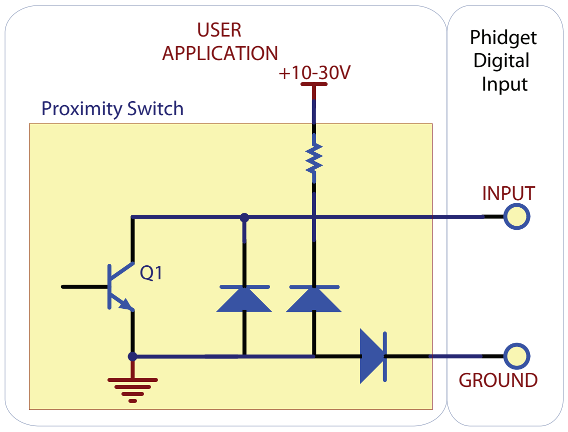
By designing to have Collector-Emitter current > 280μA, the digital input will report TRUE.



## Using a Capacitive or Inductive Proximity Switch

Capacitive proximity switches can detect the presence of nearby non-metallic objects, whereas inductive proximity switches can detect only the presence of metallic objects. To properly interface one of these proximity switches to the digital inputs, a 3-wire proximity switch is required, as well as an external power supply.

We have checked the following switch from Automation Direct to verify that it works with the Digital Inputs. Similar capacitive or inductive proximity switches from other manufacturers should work just as well.



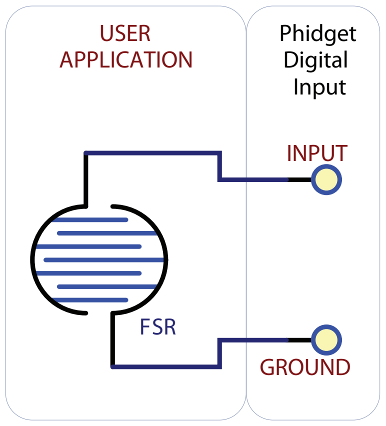
| Manufacturer | Web Page | Capacitive Part No | Inductive Part No |
|---|---|---|---|
| Automation Direct | www.automationdirect.com | CT1 Series | AM1 Series |

### Using an FSR or other variable resistor as a switch

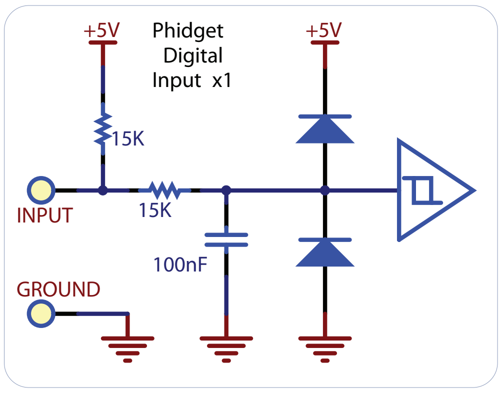
The digital inputs can be easily wired to use many variable resistors as switches.

If the resistance falls below 2.8k Ohms, the Digital Input will go TRUE.

If the resistance rises above 11k Ohms, the Digital Input will go FALSE.



## Functional Block Diagram



The digital inputs have a built in 15K pull-up resistor. By connecting external circuitry, and forcing the input to Ground, the Digital Input in software will read as TRUE. The default state is FALSE - when you have nothing connected, or your circuitry (switch, etc) is not pulling the input to ground.

## Digital Input Hardware Filter

There is built-in filtering on the digital input, to eliminate false triggering from electrical noise. The digital input is first RC filtered by a 15K/100nF node, which will reject noise of higher frequency than 1Khz. This filter generally eliminates the need to shield the digital input from inductive and capacitive coupling likely to occur in wiring harnesses.

## Digital Input Sampling Characteristics

The state of the digital inputs are reported back to the PC periodically. During this sampling period, if a digital input was true for greater than 4.0ms, the digital input is guaranteed to be reported as true in software. This makes the digital input much more sensitive to reporting TRUE state, and makes it useful to watch for short events.

## Encoders

The 1065 can be used with a wide assortment of mechanical and optical encoders. The encoder should be of quadrature output type, indicating that there will be two quadrature output channels (usually labeled A and B). Some encoders have  a third output channel to signal when the index pin (a reference point for zero position or a complete revolution) has been reached - the 1065 does not use this signal.

The 1065 can time the duration between a group of quadrature changes.  The time is returned in 1/3 millisecond. This time value can be used to calculate velocity and acceleration.

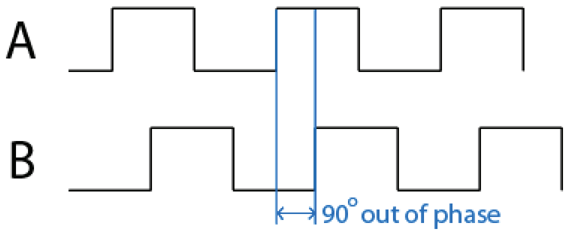
The maximum rate of the 1065 Encoder is specified at 500,000 counts per second.  In your application, this number relates directly to the number of revolutions per second you wish to measure, and the number of counts per revolution specified for your encoder.  If your encoder measures 1000 counts per revolution, then the limit on measurable revolutions per second is 500, or 30,000 RPM.

### Quadrature Encoder Fundamentals

Quadrature encoders are common, using two output channels to dictate both a change and the direction of change. In an encoder system, two parallel mechanical switches or optical slots are offset slightly. This way, as the slots passs by the sensor, the staggered output indicates both the number of pulses that have occurred (the change in position) as well as which output channel is leading the other (direction of change).

### Choosing Encoders

Both mechanical and optical encoders are available, with optical encoders dominating at > 100 counts per second. Review the data sheet for the encoder you are planning to use to ensure it is compatible with the 1065.  Compatible encoders should operate from the +5VDC power provided by the 1065, and use a single wire for the A and B channels.  Some types of encoders will use two wires (differential) for each channel - these are not compatible. Compatible encoders are often advertised as being 'single ended', and will have 4 or 5 connections.

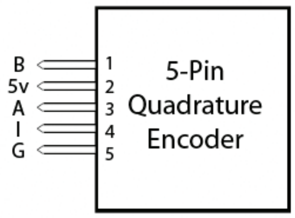


Absolute encoders will not work with this device.

Mechanical encoders are effectively just two switches, and often have a push button switch on the shaft.  This push button switch can be wired into a digital input on the 1065. Mechanical encoders do not have to be connected to +5V.

**Warning:** The 1065 incorporates a 2.4 kOhm pull-up resistor on the line from the encoder input connector.  If your encoder is mechanical, this pull-up resistor eliminates the requirement to add your own external pull-up resistor.

Some optical encoders will have a simple photo-transistor / open-collector output. The 2.4 kOhm pull-up resistor may have to be augmented with a stronger parallel resistor if your optical encoder datasheet calls for it.  Some open-collector outputs will not be strong enough to pull this resistor to ground.  These encoders are not compatible with the 1065, and may only work initially, or not at all.  If you have any doubts, please contact us.



Most optical encoders have a push-pull output, and the pull-up resistor is irrelevant, but weak enough not cause problems.

We have reviewed the following encoders, and found that they can be used with the 1065.  This is not meant to be a comprehensive list but should be used as examples of the type of encoders that can be used with the 1065.

## Connectors

Each Input uses a 3-pin, 0.100 inch pitch locking connector.  The connectors are commonly available - refer to the

| Manufacturer | Web Page | Part Number |
|---|---|---|
| Grayhill | www.Grayhill.com | Series 63R, Series 61R Series 63Q TTL Output |
| US Digital (recommended) | www.USDigital.com | S4, S5, E2, E3, E4, E4P, etc. |
| Avago Technologies (Formerly Agilent) | www.avagotech.com | HEDS 5500 |
| CUI Inc. | www.cui.com | AMT103-V |

Table below for manufacturer part numbers.

Note: Most of the above components can be bought at www.digikey.com

| Manufacturer | Part Number | Description |
|---|---|---|
| Molex | 50-57-9405 | 5 Position Cable Connector |
| Molex | 16-02-0102 | Wire Crimp Insert for Cable Connector |
| Molex | 70543-0004 | 5 Position Vertical PCB Connector |
| Molex | 70553-0004 | 5 Position Right-Angle PCB Connector (Gold) |
| Molex | 70553-0039 | 5 Position Right-Angle PCB Connector (Tin) |
| Molex | 15-91-2055 | 5 Position Right-Angle PCB Connector - Surface Mount |

## Device Specifications

| Characteristic | Value |
|---|---|
| **Motor Controller** | |
| Output Controller Update Rate | 42 Updates per second |
| Response Time | 30 milliseconds |
| Velocity Resolution | ±8-bit (~0.39%) |
| Velocity Range | ±100% |
| Acceleration Resolution | 8-bit (~0.39%) |
| Acceleration Range | 24.52 - 6250 %/s$^2$ |
| Acceleration Time Range (-100% to +100% velocity) | 32 - 8160 milliseconds |
| Braking Resolution | 8-bit (~0.39%) |
| Braking Range | 0 - 100% |
| Motor PWM Frequency | Varies with motor speed |
| **Analog Input** | |
| Impedance | 900K ohms |
| 5V Reference Error | Max 0.5% |
| Update Rate | 125 samples/second |
| **Digital Inputs** | |
| Pull-Up Resistance | 15K ohms |
| Low Voltage (True) | 0.8V Max |
| High Voltage (False) | 2.1V Min |
| Maximum Voltage | ±15V |
| Update Rate | ~125 samples/second |
| Recommended Wire Size | 16 - 26 AWG |
| Wire Stripping | 5-6mm strip |
| **Encoder** | |
| Maximum Count Rate | 500,000 Counts per second |
| Internal Output Pull-Up Resistance | 2.4k Ohms |
| Software Update Rate (typical) | 8 milliseconds |
| USB Update Rate | 125 samples/second |
| Time Resolution | 1/3 ms |
| **Board** | |
| Minimum Power Supply Voltage | 9 VDC |
| Maximum Power Supply Voltage | 28 VDC |
| Continuous Motor Current | 5 A |
| Motor Overcurrent Trigger | 8 A |
| Power Supply Overvoltage Trigger | 40 V |
| USB-Power Current Specification | 100 mA max |
| Device Quiescent Current Consumption | 20 mA |
| Operating Temperature | 0 - 70°C |

***Note:*** Current from USB supply is not available for motors.

## Product History

| Date | Board Revision | Device Version | Comment |
|---|---|---|---|
| June 2011 | 0 | 100 | Product Release, requires phidget21 2.1.8 |

## Support

Call the support desk at 1.403.282.7335 9:00 AM to 5:00 PM Mountain Time (US & Canada) - GMT-07:00

or

E-mail us at: support@phidgets.com