Hawk

USB Control Boards

Installation and Users Manual
(Including Hawkeye Software)

Available exclusively from
PC Control Ltd.
www.pc-control.co.uk

Revision 2.3

Contents

1. Introduction to Hawk Range of Boards and Hawkeye Software

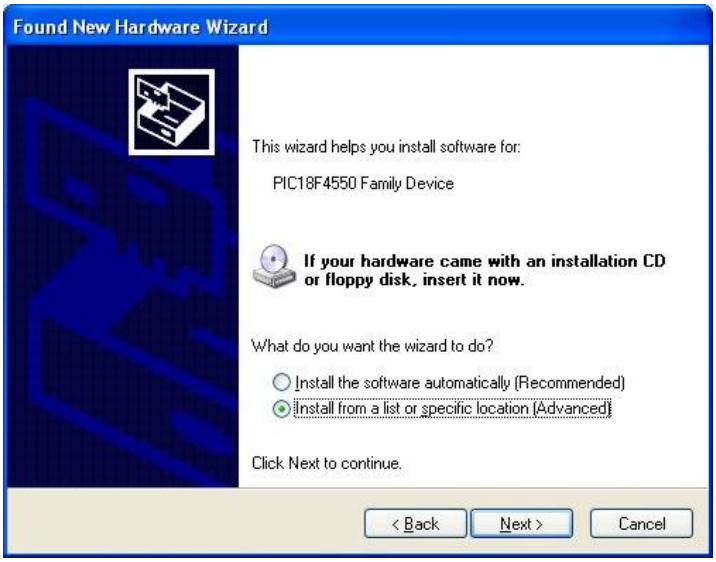
The Hawk range of boards are versatile USB adaptors which allow the PC user to explore the world of real time control and automation. All boards in the range are part of a new generation of control boards from PC-Control which provide the full speed USB 2.0 interface. They are tools, which are attractive to both the novice and experienced user. **BEFORE** connecting any of the Hawk range of boards to your USB ports you must first install the Hawkeye software and associated drivers.

2. Software Installation

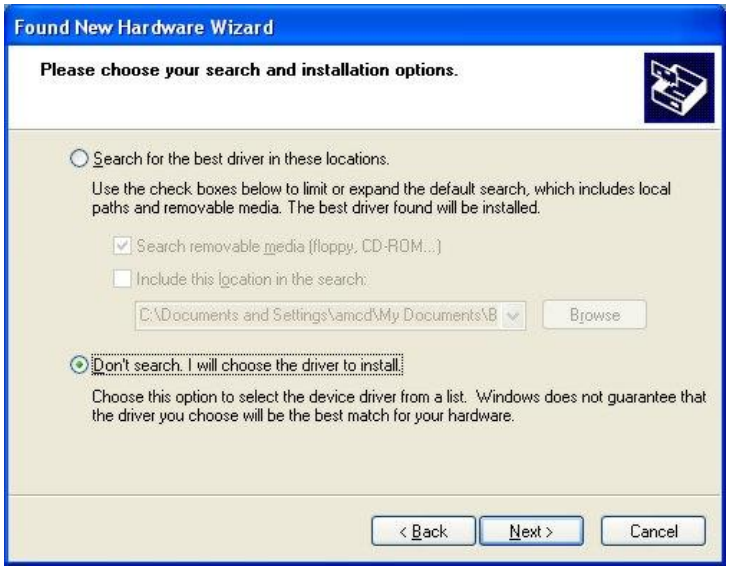
To install "Hawkeye" software simply insert the supplied CD into your CD ROM drive and the installation menu should start up automatically. If it does not then use windows explorer to navigate to the CD drive and find the file called "Setup". Double clicking on this file will start the installation. Note that all installation procedures described should be done having logged on as Administrator with full administrator privileges. Also note that use of the driver is restricted to those users with Admin privileges and you should therefore login to windows as administrator whenever actually using this software and associated drivers.

Installation is very simple and only requires following the on-screen instructions. Once "Hawkeye" installation has been completed you should then connect the Hawk board to a free USB port. The first time you connect it, Windows will let you know it has found new hardware with the following window…

Select "No, not this time" and click "Next". You will then see…

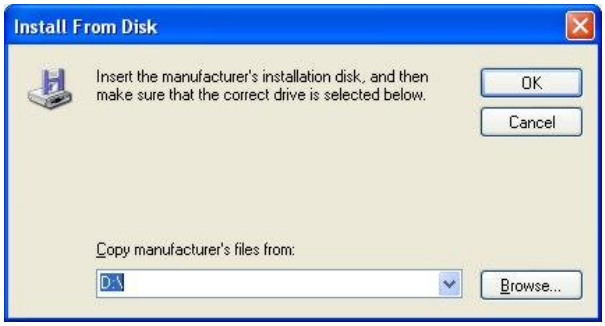


Select “Install from a list or specific location (Advanced)” and click “Next” which will lead to the screen below…



Choose “Don’t search. I will choose the best driver to install” and then click “Next”

When the above screen appears click on the “Have Disk” button. Your installation CD should be in your CD drive at this time.



When presented with the above screen, make sure the drive selected “eg D:\” is the correct one on your system that has the installation CD in it. Click “OK”

The wizard should find the “PC Control Board” and clicking next should begin the installation. If you are presented with the following box….

… simply click “Continue Anyway”.

The installation should then complete automatically with a final screen showing..

.. which requires you to click on Finish. You may also see an information bubble appear informing you that your new hardware is “installed and ready for use”.

Windows 7/8 Additional Notes for Windows 7/8 Installations:

When the board is first connected to a USB port windows may try to automatically install a driver for it. This will fail and windows will inform you of this. Simply close the message box and proceed to install the driver manually driver manually as follows...

Go to the control panel and click on "View devices and printers".

In the list of devices you will see the attached board. Double click on this and choose the "hardware" tab. Then choose the properties to see the device properties dialog box.

Click on the "Driver" tab and choose "Update Driver".

Choose “Browse my computer for driver software” and then select the drive which contains the installation CD (followed by "Next").

Windows should then install the driver and confirm it has done so..

The board is now ready to be used with its supplied application software or your own programs using the DLL supplied.

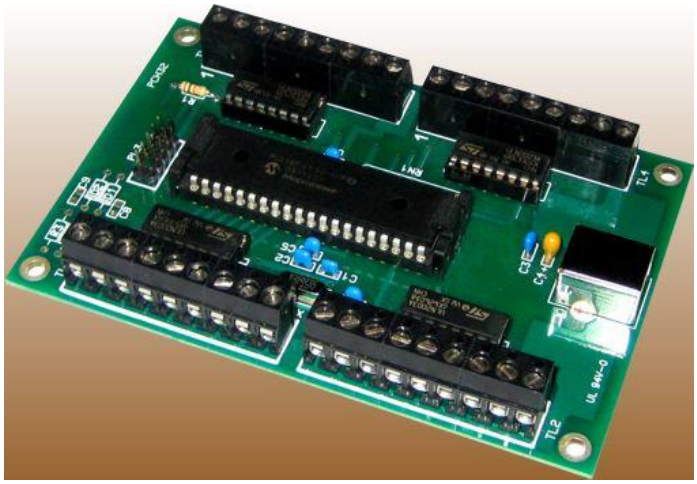
3. Hawk Hardware

The Hawk range of USB boards provides a consistent interface between the PC and a wide range of "real world" devices to be controlled or monitored. Although the current range of Hawk board types is limited, this range will continue to expand to incorporate as many types of device as possible whilst still maintaining an "easy to use" and "easy to program" development environment. This will provide users with access to the tools to create their own PC controlled application in the shortest time with the least amount of effort.

The specific hardware and connection details for each of the boards in the current range will now be presented. Please remember to check our website for new additions to the range and updated software and manual which will be made available for free download.

4. Switching Hawk (28 Switching Type Outputs from a Single USB Port)

With the Switching Hawk it is possible to switch on and off up to 28 external devices under the control of a PC. The external devices can be any DC powered unit which uses less than 500mA of current at less than 24v. For devices requiring higher voltages or currents external "slave" relays can be used giving unlimited potential for control.



The ease of connectivity of USB, makes connection to the PC simple. Screw terminals mean that external devices to be controlled can be attached without any need for soldering. The included "**Hawkeye"** application software allows the user to quickly verify that everything is connected correctly by providing full manual controls for all board facilities.

A DLL is provided to make it easy for the programmer to construct his own software applications and take full advantage of the Switching Hawk hardware. The DLL provides a set of functions which allows the user to specify the on/off pattern of the outputs without having to know or even try to understand the details of USB communication protocols etc..

4.1 Hardware Installation

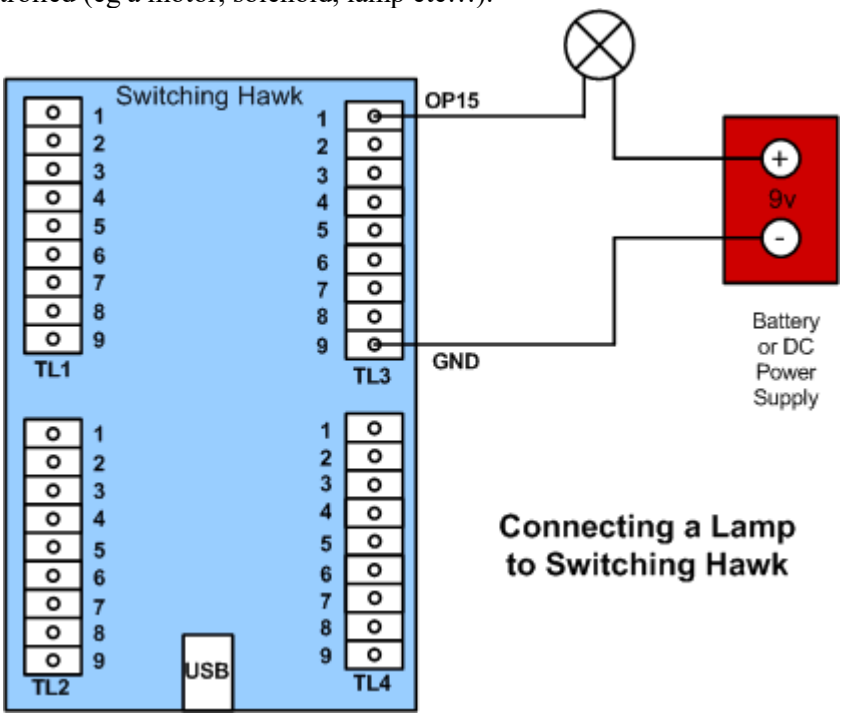
Simply connect the Switching Hawk to any available USB port (*This will require a standard USB cable*). Although it will operate from bus powered hubs it is recommended that you connect it to a primary USB socket or a self powered hub. This allows Switching Hawk to take full advantage of the available 500mA from such a connection. Bus powered hubs are limited to 100mA.

It is recommended that you do not connect Switching Hawk to a USB port until you have installed the "Hawkeye" software and have the installation disk in your CDrom drive. This will make it easy to install the required Windows drivers. See previous software section.

4.2. Connecting Devices to Switching Hawk

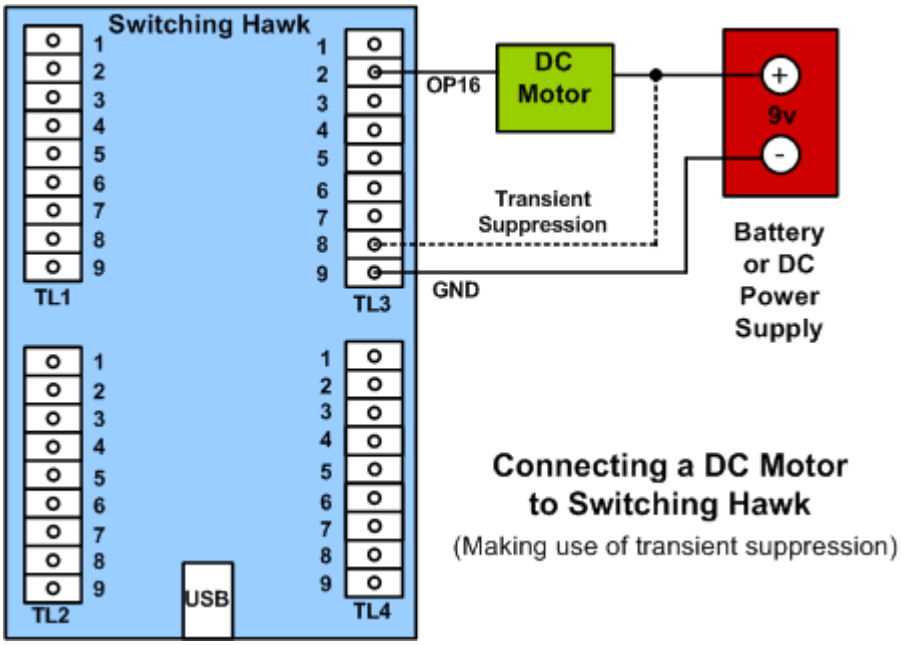
The **Switching Hawk** board is designed to be flexible in its uses and has a number of electrical features that make it easy to use in many applications. The following descriptions are merely suggestions on suitable ways to use the unit.

There are 28 *high voltage capable* DC switching outputs available on the screw terminals with each output taking the form of an "open collector" driver. For these outputs to operate correctly it is necessary to link the 0v (or Ground) connection of the Switching Hawk (screw terminals TL1-1, TL2-1, TL3-9, TL4-9) to the 0v connection of the external power supply which is being used to "drive" the device to be controlled (eg a motor, solenoid, lamp etc…).



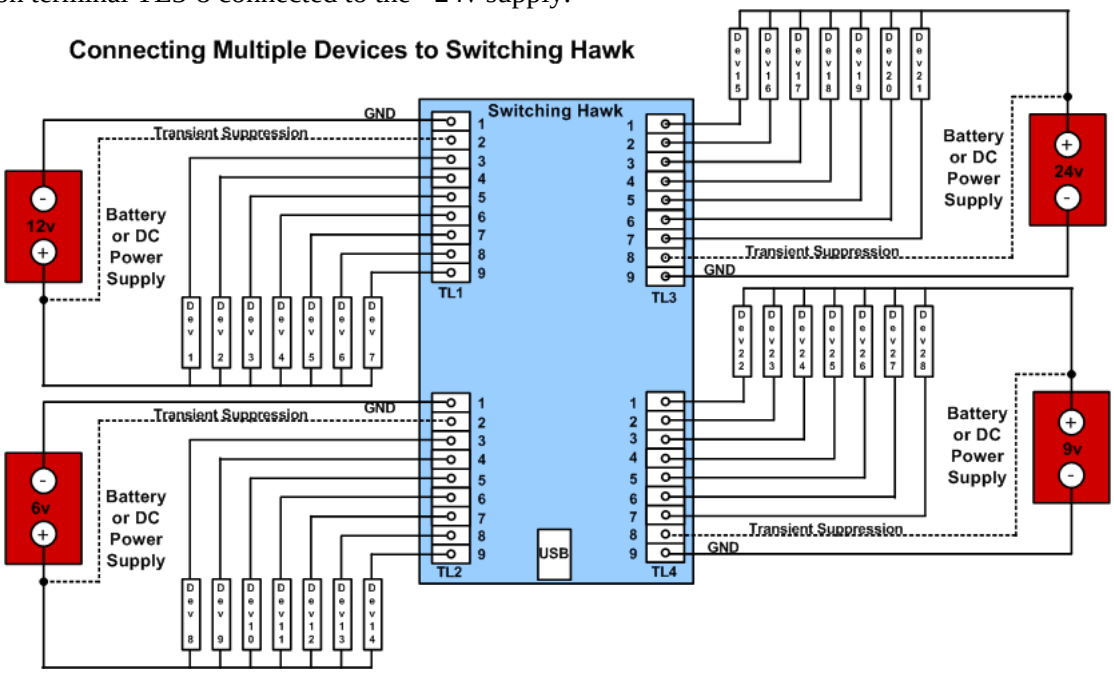
The device being controlled is then connected between one of the Switching Hawk control outputs (e.g. OP15 on Screw Terminal TL3-1) and the external positive end of the supply (e.g. +9v).

When the Switching Hawk output is turned on (by the manual control button in Hawkeye or by your own software) the terminal becomes a low impedance path to ground and current flows in the external circuit through the attached device. For low power non-inductive devices such as lamps, that is all that is necessary. If you are directly attaching an inductive load (such as a motor or relay) it is advisable to take precautions against switching transients.



Switching transients are spikes in the voltage that occur when an inductive load is turned off and can be high enough to cause damage to attached circuitry. Switching Hawk has built in suppressor diodes connected to four of the screw terminals (TL1-2, TL2-2, TL3-8, TL4-8) to provide a means of suppressing these transients. To make use of these simply connect the terminal associated with the "group of 7" outputs needing suppression to the external positive supply being used to "drive" your inductive load. i.e. Terminal TL1-2 is intended for inductive devices connected to terminals TL1-3 to TL1-9, TL2-2 for terminals TL2-3 to TL2-9, TL3-8 for terminals TL3-1 to TL3-7, TL4-8 for terminals TL4-1 to TL4-7.

NOTE: Care should be exercised if using different supply voltages for inductive loads. For example to operate both a 12v and a 24v dc motor using the Switching Hawk and using the transient suppression technique just described, you must ensure that you don't have two different supplies on the same group of 7 outputs. i.e. in this case you could have the 12v motor on terminal TL1-3 with its protection terminal TL1-2 connected to the +12v supply, and the 24v motor on terminal TL3-1 with it's protection terminal TL3-8 connected to the +24v supply.



In summary , when using transient protection ensure you do not have more than one supply voltage on a given group of 7 outputs. The diagram above shows the Switching Hawk connected to its maximum number of independent devices and with 4 different power supplies. It is, of course , quite acceptable and simple to use all outputs with just a single supply. There is no need to have a different voltage supply on each set of 7 outputs, and, without the need for any transient suppression (eg non-inductive devices), there is no restriction on how many different voltage supplies that can be used as long as the 0v from all supplies are connected together.

The output switching components used by Switching Hawk are ULN2003 High Current / High Voltage Darlington Drivers. These devices are capable of being switched up to 50v and 500mA. However, although capable of these operating limits, the recommended application of the Switching Hawk is for voltages up to 24v. The current capability on each output is 500mA. As the ULN2003 data sheet suggests, this can be extended by connecting outputs in parallel. If doing this, it is obviously essential that you ensure that all outputs that are paralleled are always in the same state (on/off) at the same time or you run the risk of one output taking all of the current and exceeding maximum limits. Obviously it is not possible to do this using the manual controls of Hawkeye, it must be done in your own software. You must also take account of the overall power handling capability of the ULN2003 when using multiple outputs each with high current. Refer to the graph of “peak collector current vs duty cycle and number of outputs” in the ULN2003 data sheet included on the installation CD.

4.3 Connector Pinouts

Pinout of the High Voltage Switching Outputs On Screw Terminals(TL1)

Pin	Signal description
1	GND
2	Transient Suppression
3	Switching Output 1
4	Switching Output 2
5	Switching Output 3
6	Switching Output 4
7	Switching Output 5
8	Switching Output 6
9	Switching Output 7

Pinout of the High Voltage Switching Outputs On Screw Terminals(TL2)

Pin	Signal description
1	GND
2	Transient Suppression
3	Switching Output 8
4	Switching Output 9
5	Switching Output 10
6	Switching Output 11
7	Switching Output 12
8	Switching Output 13
9	Switching Output 14

Pinout of the High Voltage Switching Outputs On Screw Terminals(TL3)

Pin	Signal description
1	Switching Output 15
2	Switching Output 16
3	Switching Output 17
4	Switching Output 18
5	Switching Output 19
6	Switching Output 20
7	Switching Output 21
8	Transient Suppression
9	GND

Pinout of the High Voltage Switching Outputs On Screw Terminals(TL4)

Pin	Signal description
1	Switching Output 22
2	Switching Output 23
3	Switching Output 24
4	Switching Output 25
5	Switching Output 26
6	Switching Output 27
7	Switching Output 28
8	Transient Suppression
9	GND

For all terminals, terminal number **1** is marked on the board as a white Number '**1**' and terminal **9** is the one nearest to the terminal number label (TL1, TL2, TL3, TL4)

PL3 Connector.

The connector labelled PL3 is reserved for facilities used during manufacturing and production testing and **no connections should be made to it**. A small link is fitted between pins 2 and 4 of PL3: this should **NOT** be removed.

5. Motor Hawk (Bi-polar Stepper Motor or Twin DC Motor Control)

With the Motor Hawk it is possible to control either a 4-phase Bi-Polar stepper motor **or** two DC motors in both speed and direction. There are also 8 digital inputs and 5 digital outputs available for general use, with 4 of those inputs configurable as automatic limit switch inputs for motion control applications.

5.1 Hardware Installation

Simply connect the Motor Hawk to any available USB port (*This will require a standard USB cable*). Although it will operate from bus powered hubs it is recommended that you connect it to a primary USB socket or a self powered hub. This allows Motor Hawk to take full advantage of the available 500mA from such a connection. Bus powered hubs are limited to 100mA.

5.2 Stepper Motor Control

The Motor Hawk has outputs capable of "driving" one bi-polar (or hybrid) stepper motor.

Stepper motors are one of the most useful devices in the world of control, automation and robotics. They form the most convenient and versatile bridge between a set of motion rules in a controller (computer) and the motion itself. They can be made to move slowly, quickly, in reverse, pause, complete revolutions, partial revolutions and even individual steps of less than a degree of rotation. With this flexibility of movement coupled with an abundance of torque for relatively little power applied, the stepper motor finds many suitable applications.

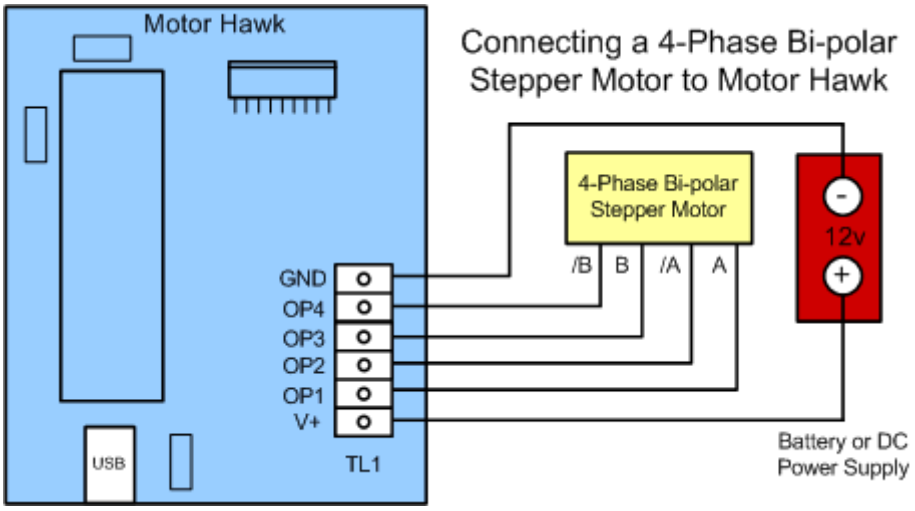
The downside to stepper motors is that they are not as simple to "drive" (electrically speaking) as simple DC motors, which just need the required volts and amps to do their bit. Stepper motors need a precise sequence of pulses delivered to the correct winding at the correct time in order to perform their required task. It would be nice if all you had to do was specify the number of steps to take , in what direction and at what speed and the stepper motor obliged. This is the function of the Motor Hawk.

From the PC, these requirements can be specified and "sent" to the Motor Hawk via USB, and the Motor Hawk on board microcontroller then generates the precise sequence of pulses on the appropriate winding to move the stepper motor accordingly.

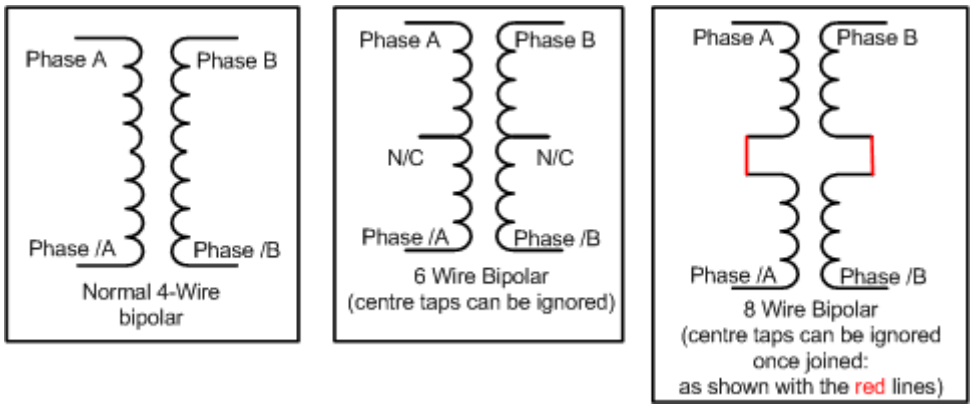
5.3 Connecting a Stepper Motor to Motor Hawk

Most bi-polar stepper motors have just 4 wires to be connected although some will have more than this. Some stepper motors are capable of being connected in both bi-polar or uni-polar mode and it is these that will have more than 4 wires available. It is still possible to use those with more than 4 wires and the way to connect them will be discussed later. The Motor Hawk uses a row of screw terminals for making all connections to the stepper motor. This is labelled as TL1 on the board. The individual terminal connections are labelled as OP1, OP2, OP3, OP4 for the motor wires and GND and V+ for the motor power supply. All stepper motors need an external DC power supply with a voltage and current capability suited to the motor requirements. It is important to consider the specifications of the Motor Hawk when choosing a stepper motor and associated power supply for your application. The

most important considerations are the maximum voltage and current capability of the Motor Hawk. This is a maximum of 36v and 2A per phase. Bi-polar stepper motors come in a wide variety of specifications differing in power, step resolution, torque, voltage and current requirements. A typical medium torque stepper motor requires 12v at 200mA per phase winding to function correctly. Connection details for a stepper motor are shown below.

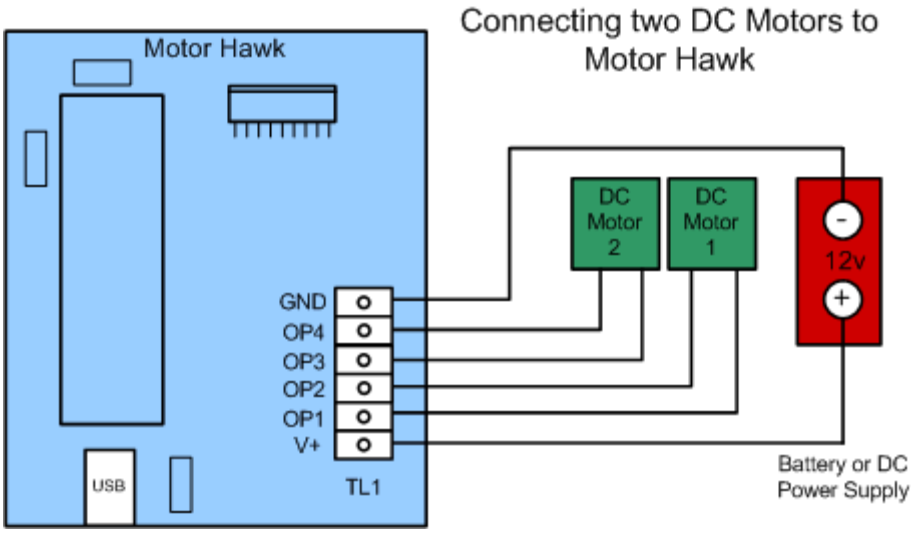


As mentioned above it is possible to use motors which have more than 4 wires when the motor has been designed to be operated in bi-polar mode. These motors generally have 6 or 8 wires. When the motor has 6 wires then you can simply ignore (i.e. do not connect to anything) the extra wires. These wires will be connected to the centre taps of the main two phase coils. These extra connections are only used when operating the motor in unipolar mode. Similarly with the 8 wire version. The centre taps can be ignored but they must first be connected together as shown below (in red) in order to complete the two phase coils.



5.4 Connecting DC Motors to Motor Hawk

The Motor Hawk has the option of either driving a stepper motor **or** , up to two DC motors. This option is selectable in software (see software section). The DC Motors can be any simple DC motor with voltage and current requirements within the specifications of the Motor Hawk (i.e. less than 36v and less than 2A each). The motors which can be driven in both directions are connected as shown below….



Note: the DC motors cannot be connected at the same time as a stepper motor.

The Motor Hawk has four outputs for driving two motors independently. The output terminals are labelled as OP1 and OP2 for motor 1 and OP3 and OP4 for motor 2.

When Motor 1 is set to drive in the forward direction terminal 'OP1' is positive with respect to 'OP2' and vice-versa when set to go in reverse.(similarly for OP3/OP4 and Motor 2). As well as direction control the motor outputs control the speed of the attached motor by using PWM (pulse width modulation) to deliver a variable power based on the external motor supply. i.e. the voltage it produces is always exactly equal to the voltage applied to the motor supply terminals (VM+ and GND) but it is constantly turned on and off at a high rate. The power transferred to the motor (and hence the resulting speed) is varied by changing the amount of time the output spends 'ON'. i.e. if the output is only on for 5ms out of every 100ms then the resulting speed would be about 5% of full speed. If it is on for 50ms out of every 100ms then you would have approx half full speed.

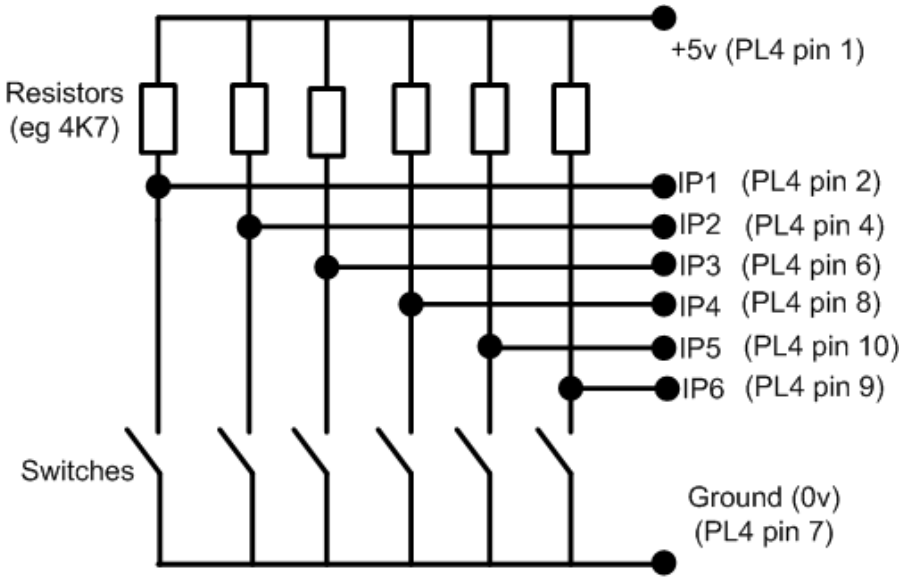
The speed can be varied over the full range from less than 1% to more than 99% in 255 pre-defined steps. This gives very fine control over the speed of the motor. Remember that it is the power being delivered to the motor which is being varied which in turn causes a speed change. If the motor is turning a heavy load then the speed will be proportionally less for the same power output. Each motor output is capable of delivering up to 2A to the connected motor. The Multi-pin driver chip on the Motor Hawk board will tend to be warm when driving significant currents into one or both motors. This is perfectly normal. Most applications will not require the use of maximum power in both motors for long periods. However, if your particular application is intended to run close to maximum limits you may prefer to fit the optional heatsink (part no. ACC001) to the driver chip to reduce its temperature. This is available in the accessories section of the Control Shop (*www.pc-control.co.uk).*

5.5 Connecting Digital Inputs to Motor Hawk

There are 6 digital inputs on a Motor Hawk provided on PL4. This is a standard 10 way header connector suitable for use with standard 10 way ribbon cable type plugs available from most good electronics stockists. (*the terminator 10s (part No ADP002) from The Control Shop at www.pc-control.co.uk is also compatible with this connector and makes the inputs available on screw terminals*).

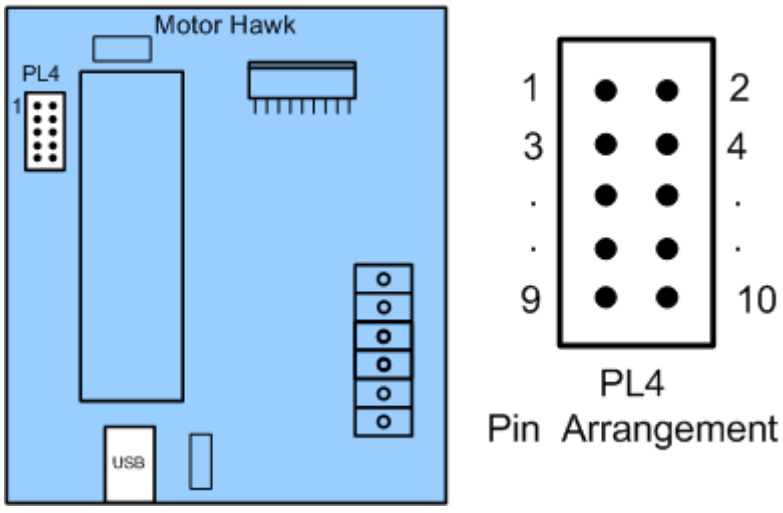
These inputs have characteristics compatible with standard 5v logic devices. i.e. when the input is at +5v it will be read as a logic '1' or "High". When the input is at 0v or GND it will be read as logic '0' or "Low. These are standard CMOS type inputs. This means that they are relatively high impedance. When not connected to any signal source, the inputs can behave like small aerials "picking up" electrical noise from the immediate environment and can appear to be randomly changing without anything connected. When using these inputs you should ensure that any unused inputs are connected to GND (0v) to prevent this.

If you are using an input to monitor a switch, the correct way to connect it is to have the input "pulled up" to +5v via a resistor (eg 4K7) and use the switch to connect it to GND when applied. For convenience a 5v supply is provided for this purpose on PL4 pin 1. The diagram below shows all six inputs connected to show the status of 6 external switches.



Using the Hawk Digital Inputs for Reading Switches

PL4 Digital Input Connector



It should be noted that the 5v supply provided on PL4 pin 1 can only be used for low current pull up use. (less than 50mA total) since it is derived from the main USB 5v. It should not be used to power higher current devices of any kind as this may damage the USB system.

Although all six are available for general use as inputs, four of them can be configured (in software) for an automatic function in the control of the motors. They are designed to act as range limits. A range limit is a mechanism to prevent a moving object moving beyond its safe operating range. For example, the motor you are controlling may be moving a drill on an X/Y drilling table along the X axis. At some point it will reach the end of its available travel and, without limits, presumably hit an end stop. If the motor continues to operate in this condition it will probably overheat and may even have enough power to damage the mechanism it is moving. To prevent this, a limit switch may be fitted

near the end of travel in such a way that it is closed when reached by the moving part. The closure of this switch is used to switch off the motor automatically. However, it would be impractical to just leave the motor "dead" against the end stop with no possibility of reversing it back into the working range, so the automatic stop only stops the forward motion. When the signal is, at some point, changed to reverse, the motor is then allowed to reverse back from the end stop. Similarly at the other end of travel, the reverse will be inhibited automatically when the other limit switch is reached but forward motion would then be allowed. This is the function of the inputs on PL4 pins 2,4,6 and 8.

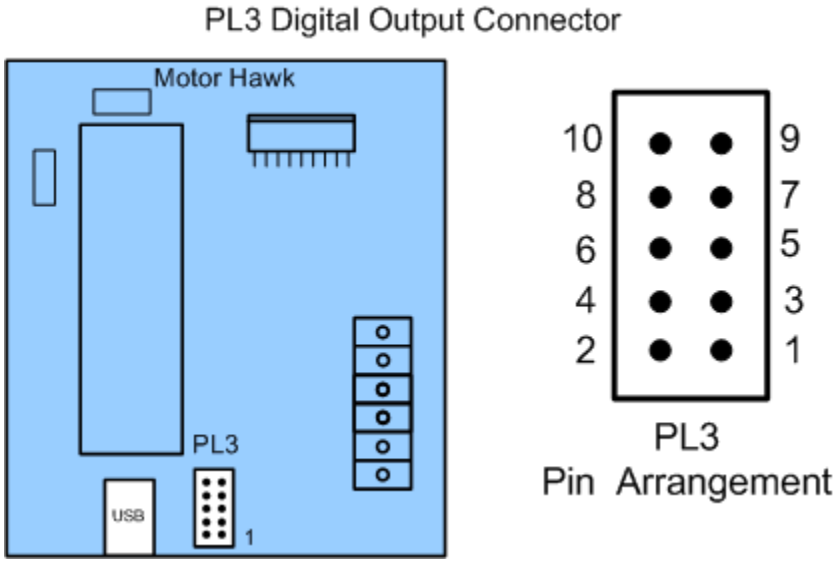
When being used for control of two DC Motors the following applies: When 2 is connected to ground it inhibits forward motion of the Motor 1 immediately and automatically without any intervention by the controlling program. Reverse would still be allowed. Similarly Pin 4 inhibits reverse motion but allows forward when connected to ground. Pins 6 and 8 are the corresponding limits for motor 2 (*6 for forward and 8 for reverse*). This makes the fitting of range limit switches very easy for both motors. When the Motor Hawk is being used for stepper motor control only inputs 1 and 2 (pins 2 and 4) can be used in this way. Since the limit switch inputs can be read like any other input it can be determined whether or not the moving device has operated any of the limit switches from within the control program allowing suitable additional remedial action to be taken, if required.

It is not necessary to use these inputs as range limits if not required. This facility can be enabled or disabled within the software.

5.6 Connecting to the Motor Hawk Digital Outputs

There are 8 digital outputs on a Motor Hawk provided on PL3. Like PL4 this is also a standard 10 way header connector suitable for use with standard 10 way ribbon cable type plugs. Each output provides either +5v or 0v under program control. These can be used for a wide variety of tasks but it should be remembered that the current capability of each output is limited to no more than 25mA.

Connection details are shown in the connector pinouts section below



5.7 Motor Hawk Heatsink

Although the Motor Hawk does not need a heatsink fitted to the main driver component for normal operation, it can be useful to fit one in cases where the device is operating for prolonged periods at near to its maximum rated output or if it is preferred to run the driver cooler.

In these circumstances an optional heatsink (OPT003) can be fitted simply by attaching the heatsink to the L298D driver IC using the screw , washer and nut provided with the heatsink. The picture below shows the board with the heatsink fitted.


Motor Hook
V 1.1
GND
OP4
OP3
OP2
OP1
V+
TL 1

5.8 Motor Hawk Connector Pinouts

Pinout of the Motor Control Outputs On Screw Terminals(TL1)

	Signal description	
Pin	Stepper Motor Option	DC Motors Option
GND	GND (0v) (Motor Supply Negative)	GND (0v) (Motor Supply Negative)
OP4	Stepper /B	Motor 2 (-)
OP3	Stepper B	Motor 2 (+)
OP2	Stepper /A	Motor 1 (-)
OP1	Stepper A	Motor 1 (+)
V+	Motor Supply Positive	Motor Supply Positive

Pinout of the Digital Inputs (PL4)

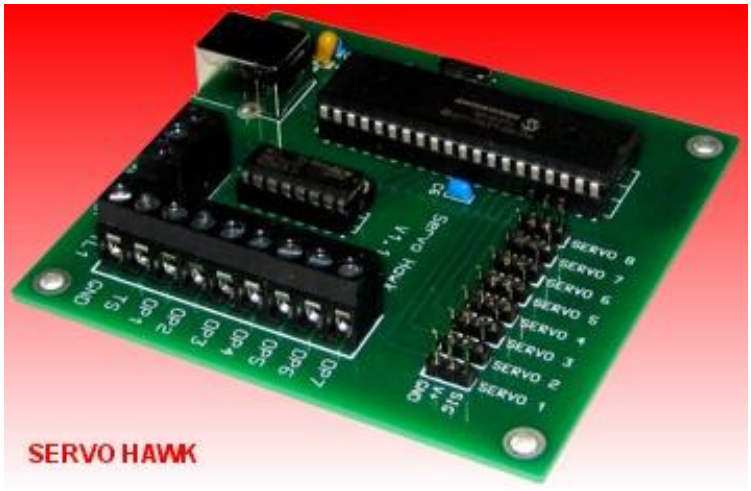
	Signal description	
Pin	Limits Function Disabled	Limits Function Enabled
1	+5v (50mA limit)	+5v (50mA limit)
2	Input 1	Inhibit Forward when 0v (Stepper Motor or DC Motor 1)
3	Not Used	Not Used
4	Input 2	Inhibit Reverse when 0v (Stepper Motor or DC Motor 1)
5	Not Used	Not Used
6	Input 3	Inhibit Forward when 0v (DC Motor 2)
7	GND (0v)	GND (0v)
8	Input 4	Inhibit Reverse when 0v (DC Motor 2)
9	Input 6	Input 6
10	Input 5	Input 5

Pinout of the Digital Outputs (PL3)

Pin	Signal description
1	Digital Output 1 (max 25mA sink/source)
2	Digital Output 2 (max 25mA sink/source)
3	Digital Output 8 (max 25mA sink/source)
4	Digital Output 3 (max 25mA sink/source)
5	Digital Output 7 (max 25mA sink/source)
6	Digital Output 4 (max 25mA sink/source)
7	Digital Output 6 (max 25mA sink/source)
8	Digital Output 5 (max 25mA sink/source)
9	GND (0v)
10	GND (0v)

6. Servo Hawk (8 Servo Control)

Directly and independently control up to 8 standard servos with the Servo Hawk . There are also 7 switching type outputs capable of switching devices up to 30v and 500mA including transient suppression facilities for those inductive devices (such as motors, solenoids and relays)



6.1 Hardware Installation

Simply connect the Servo Hawk to any available USB port (*This will require a standard USB cable*). Although it will operate from bus powered hubs it is recommended that you connect it to a primary USB socket or a self powered hub. This allows Servo Hawk to take full advantage of the available 500mA from such a connection. Bus powered hubs are limited to 100mA.

6.2 Servo Control

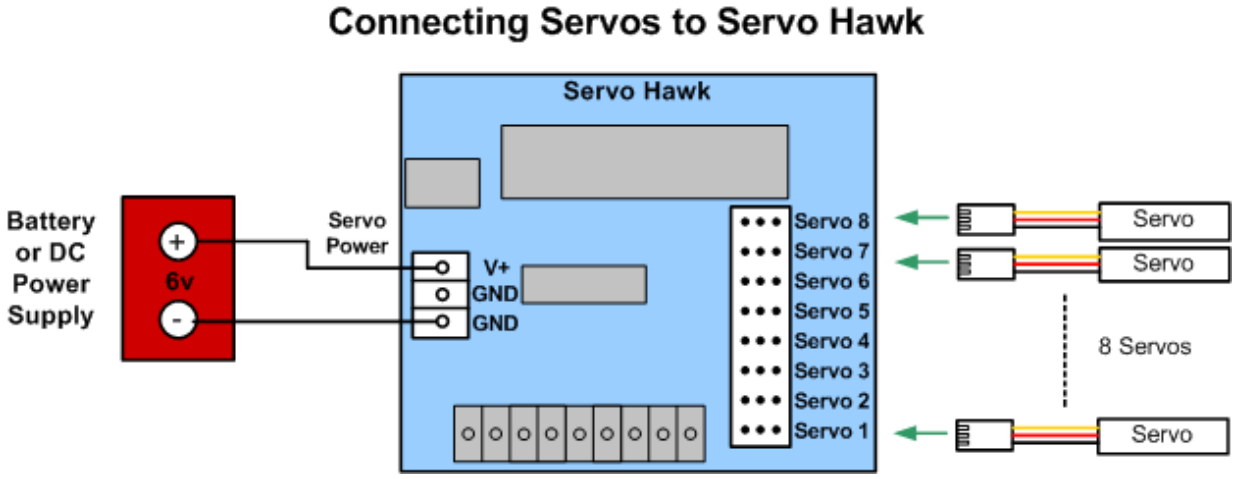
Servos are rotational devices that allow the angular position of rotation to be specified. Once specified the servo will "hold" that position against external forces. The way the position is specified is by supplying a variable width pulse at regular intervals. The width of the pulse determines the angular position. The accepted standard for the pulse width is to have the centre position defined by a width of 1.5ms , the extreme anti-clockwise position as 1.0ms and the extreme clockwise position as 2.0ms. The vast majority of servos adhere to this standard although the amount of rotation corresponding to extreme clockwise or anti-clockwise varies according to the servo design. Typically this ranges from +/-90 degrees to almost +/- 180 degrees with the most common ones falling somewhere in between.

Servos are also fairly standard in the type of connector they use for connecting the servo to the controller (eg servo hawk), especially the hobbyist RC servo type. The connector is a three way socket as shown on the right. The three wires are usually (but not always) coloured black, red and yellow. The black wire is the ground (or common), the yellow is the pulse width modulated signal for position control and the red is the power supply positive terminal connection. i.e. apart from the pulse width signal that tells the servo where to move to, it also requires its own power source for driving its internal motor and control circuitry. The voltage and current requirements of the servo power supply will vary from servo to servo with the ones producing the highest torque having the highest current requirements. To accommodate this variability, servo hawk does not supply any power to the servo. It simply provides a convenient way of connecting an external power source to all eight servos that it controls. Servo Hawk provides 8 connection points for taking the standard three way connector from each of the eight servos that it will control. The control signals are "driven" by the on-board microcontroller whilst the power connections (the black and red wires) are simply routed to a pair of screw terminals where the external servo power supply can be connected. Although most servos only require a few hundred milli-amps the power

connections can handle up to 5A, which would accommodate even very high specification servos.

6.3 Connecting Servos to Servo Hawk

As mentioned above, connecting a servo to the Servo Hawk is simply a matter of plugging in the standard three way socket attached to the servo to one of the eight on board three pin header connections. The header connections are labelled Servo 1 to Servo 8. Up to 8 servos can be connected to the Servo hawk at any one time, with identification of an individual servo on a given board by the number of the header it is connected to. Always check the specifications of the servo you are connecting to make sure the socket has the same pin configuration as the on board header (i.e the standard servo connection).

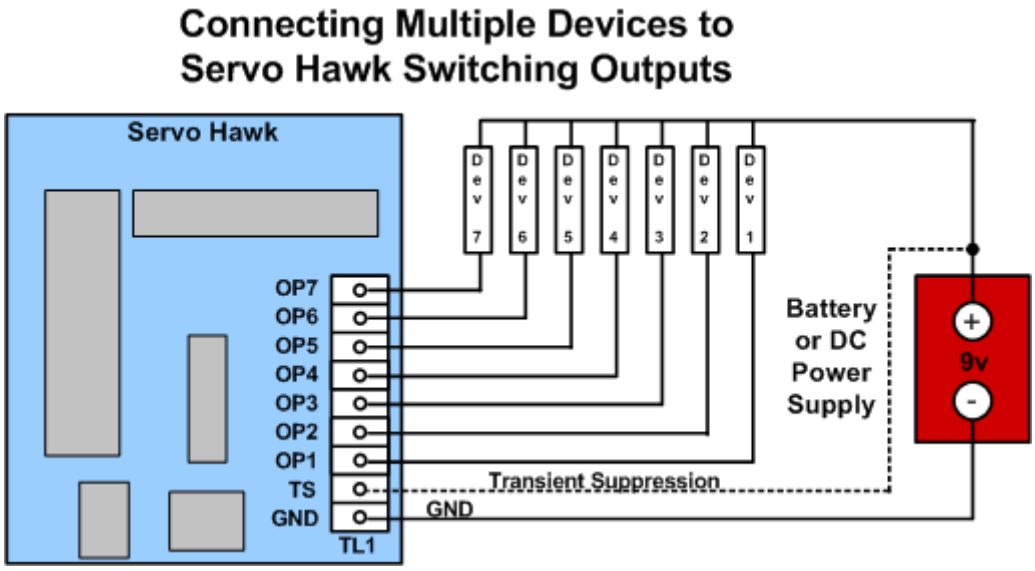


Once your servos are connected you also need to connect the servo power supply to the terminal block labelled “Servo Power”. This will distribute the power to each of the connected servos. The servo power supply ground (or negative) terminal should be connected to one of the terminals marked “GND” and the positive terminal of the power supply to the “V+” terminal.

6.4 Connecting Devices to the Servo Hawk Switching Outputs

In addition to the servo outputs on the servo hawk there are 7 high voltage switching outputs. These allow a wide range of relatively low power DC devices to be controlled in addition to the servos. These devices can be any DC operated device with a current requirement of less than 500mA and a voltage of less than 30v. Many useful devices can be used here including DC motors, solenoids and the very useful relays, allowing switching of even larger and more powerful devices.

Connecting these devices simply requires a common ground connection with the device being connected between the output and the positive terminal of an external power supply chosen to match the devices characteristics for voltage and current. This is illustrated below.



For illustration purposes, the diagram above is shown with a 9v supply. This supply should be chosen to suit the particular devices connected to the servo hawk but must be less than 30v.

6.5 Servo Hawk Connector Pinouts

Pinout of the Switching Outputs On Screw Terminals(TL1)

Pin	Signal Description
OP7	Switching Output for Device 7
OP6	Switching Output for Device 6
OP5	Switching Output for Device 5
OP4	Switching Output for Device 4
OP3	Switching Output for Device 3
OP2	Switching Output for Device 2
OP1	Switching Output for Device 1
TS	Transient Suppression Option
GND	GND (0v) (External Supply Negative)

Pinout of the Servo Power Connector

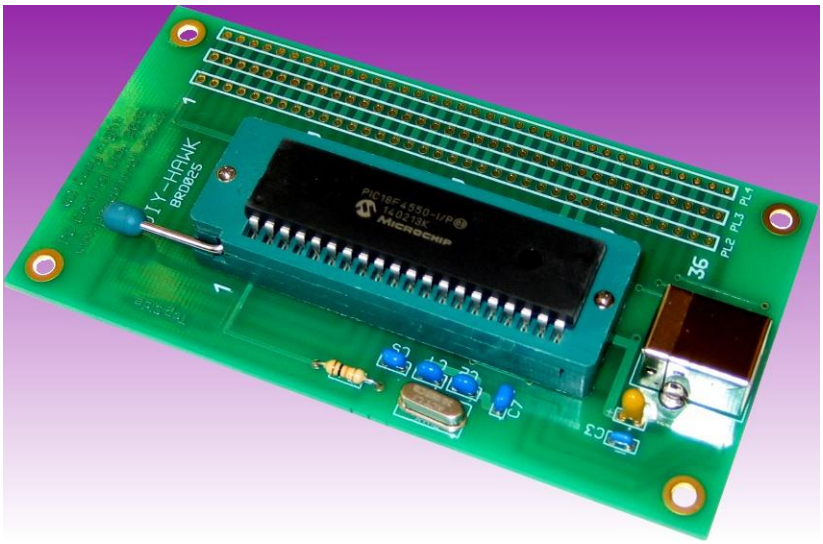
Pin	Signal Description
V+	Servo Power (External Servo Power Supply Positive)
GND	GND (0v) (External Servo Power Supply Negative)
GND	GND (0v) (External Servo Power Supply Negative)

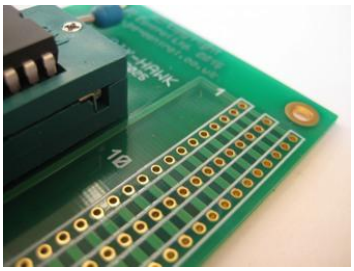
Pinout of the Each Servo Connector (1 -8)

Pin	Signal Description
SIG	5v Pulse width modulated signal for servo positioning
V+	Servo Power
GND	GND (0v)

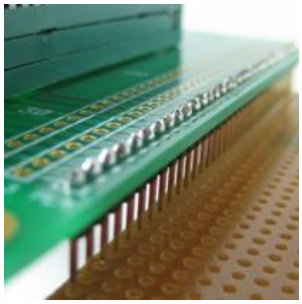
7. DIY Hawk

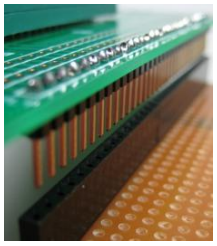
When you need, or prefer, more control over the "add on" electronics required by your project, the DIY hawk provides an excellent solution. It provides a full USB interface to the PC complete with software and DLL function library and brings out the 30 available input/output pins of the PIC microcontroller to solder pads ready for your connections.





For many applications you would simply connect your external "apparatus" directly to the solder pads with wires. For others, there is a very flexible way to connect the DIY Hawk to prototyping board (stripboard) ready for your own circuitry or interface electronics. The solder pads have been arranged in three single rows of 36 with each pad being capable of being soldered from above or below. This allows a standard 36 way "header strip" to be soldered to the board on the underside of the DIY Hawk. A corresponding stripboard can then be soldered directly to this header giving a versatile prototyping area for your own electronics. If you need a less permanent arrangement allowing you to change your stripboard then a simple 36 way socket can be used allowing the board to be easily connected and separated as required. (*Note: The 36 way header and stripboard shown are not supplied with this board but are readily available from many sources*)





The input/output arrangement on the solder pads (or header) provides for 13 digital inputs, 13 digital outputs and 4 analogue inputs with 10bit resolution(1024 levels). There are also five GND (0v) and one +5v power connection pads to make up the 36. This mixture will accommodate a very wide range of designs for automation and control.

The "ready to run" application software provided within the "Hawkeye" application will allow you to test your initial designs using manual control of all of the inputs and outputs. Once you have your basic hardware up and running you will then be looking to use the DLL (Dynamic Link Library) which provides a range of functions to be called directly from your own software. These functions provide full control over all inputs and outputs and can be used with the vast majority of programming languages, including Visual Basic, C, C++, C# and many others. The only requirement is that the programming language is capable of calling functions contained in an external DLL. This is almost always the case.

For the ultimate in flexibility, the PIC Microcontroller on the board is housed in a ZIF (zero insertion force) socket. This would allow it to be easily changed for your own PIC chip programmed with your own firmware. Although this is really for the more experienced programmers, it is not as

difficult as you would think. The PIC is a standard 18F4550 microcontroller manufactured by Microchip and commonly available from most good electronic stockists around the world. Development software and chip programmers can be found on Microchip's website together with extensive documentation and application examples. Please note that we do not directly support such developments and will not field any questions about this area of development. We feel that the Microchip support documentation in this area is more than adequate.

7.1 Hardware Installation

Simply connect the DIY Hawk to any available USB port (*This will require a standard USB cable*). Although it will operate from bus powered hubs it is recommended that you connect it to a primary USB socket or a self powered hub. This allows DIY Hawk to take full advantage of the available 500mA from such a connection. Bus powered hubs are limited to 100mA.

7.2 Making Connections to your Own Electronics

The main connection points for your own electronics or devices is via the solder pads on the right hand side of the board. These are arranged in three rows of 36 pads. The pads are numbered from top to bottom as 1 to 36 with numbering shown on the board every 10 pads. The three rows are connected together so that there are three connection points for every numbered pad. The pads have been prepared for soldering both on the top side and bottom side of the board. This provides added flexibility in your connection options. For example: As mentioned above, you can fit a 36 way header strip to the underside of the board for subsequent soldering (or plug/socket connect) to prototyping stripboard. This is also a suitable way for connection to many of the non solder type prototype boards. The spacing of the pads is on a standard 0.1" / 2.54mm pitch.

Please note that the 5v DC supply made available to your own electronics via the connection points on the board should only be used for very low power devices such as sensors and CMOS logic circuits etc.. It should ideally be limited to less than 50mA if possible but no more than the maximum allowable of 100mA. This supply is derived directly from your PC's USB supply and should not be misused. If you need extra power from the 5v, use an external supply for your own boards use and remember to link the GND connections but NOT the 5v ones.

Details of the connector pinout is shown below. This includes the actual names of the pins (port bits) used on the 18F4550 PIC chip which will be relevant to those wishing to do their own PIC programming , but for those planning on the simpler approach using our DLL and software, they can be ignored.

Terminal Pin Assignments for DIY Hawk

Solder Pad	18F4550 Port Bit	Input/Output
1	GND	GND
2	RE2	Digital Output 1
3	RE1	Digital Output 2
4	RE0	Digital Output 3
5	RA5	Digital Output 4
6	RA4	Digital Output 5
7	RA0	Analogue Input 1
8	RA1	Analogue Input 2
9	RA2	Analogue Input 3
10	RA3	Analogue Input 4
11	RB7	Digital Output 6
12	RB6	Digital Output 7
13	RB5	Digital Output 8
14	RB4	Digital Output 9
15	RB3	Digital Output 10
16	RB2	Digital Output 11
17	RB1	Digital Output 12
18	RB0	Digital Output 13
19	+5v	+5v
20	GND	GND
21	RD7	Digital Input 1
22	RD6	Digital Input 2
23	RD5	Digital Input 3
24	RD4	Digital Input 4
25	RC7	Digital Input 5
26	RC6	Digital Input 6
27	GND	GND
28	GND	GND
29	RD3	Digital Input 7
30	RD2	Digital Input 8
31	RC1	Digital Input 9
32	RC2	Digital Input 10
33	RD1	Digital Input 11
34	RD0	Digital Input 12
35	RC0	Digital Input 13
36	GND	GND

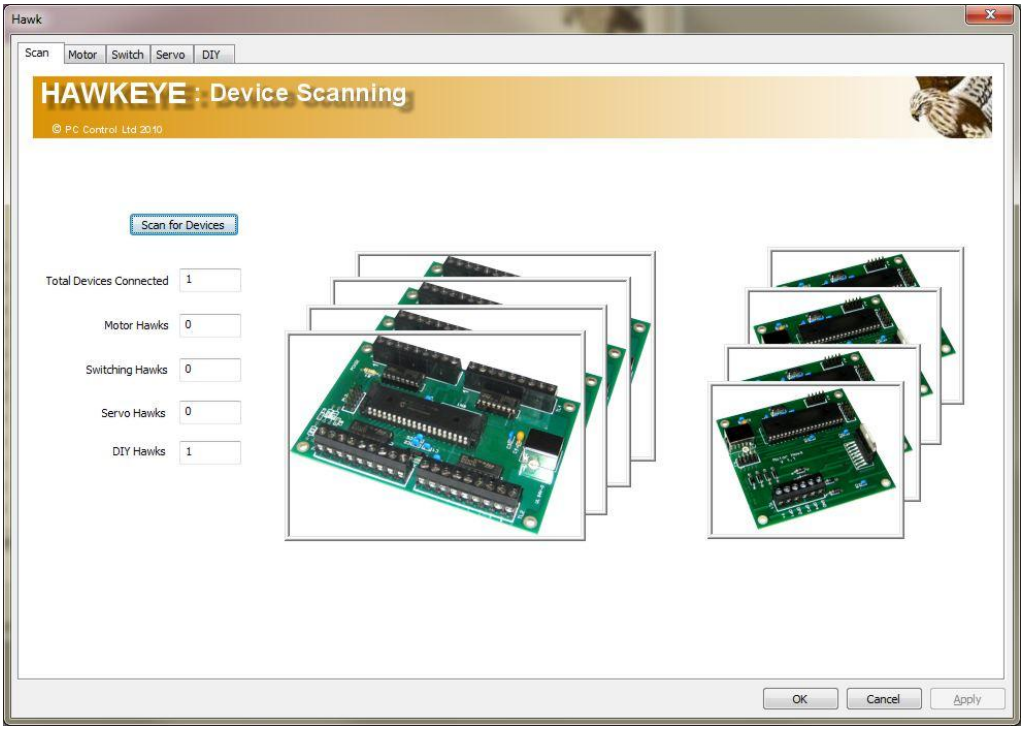
8 Hawkeye Application Software

8.1 Overview

When the Hawkeye application is started, click on the “Run” menu option to initiate the main application. The first time Hawkeye is run the customary disclaimer agreement will ask for your agreement. Click “I Agree” to never see it again and to start Hawkeye.

8.2 Hawkeye Main Screen

When you first run Hawkeye you will start from the Device Scanning screen as shown below.



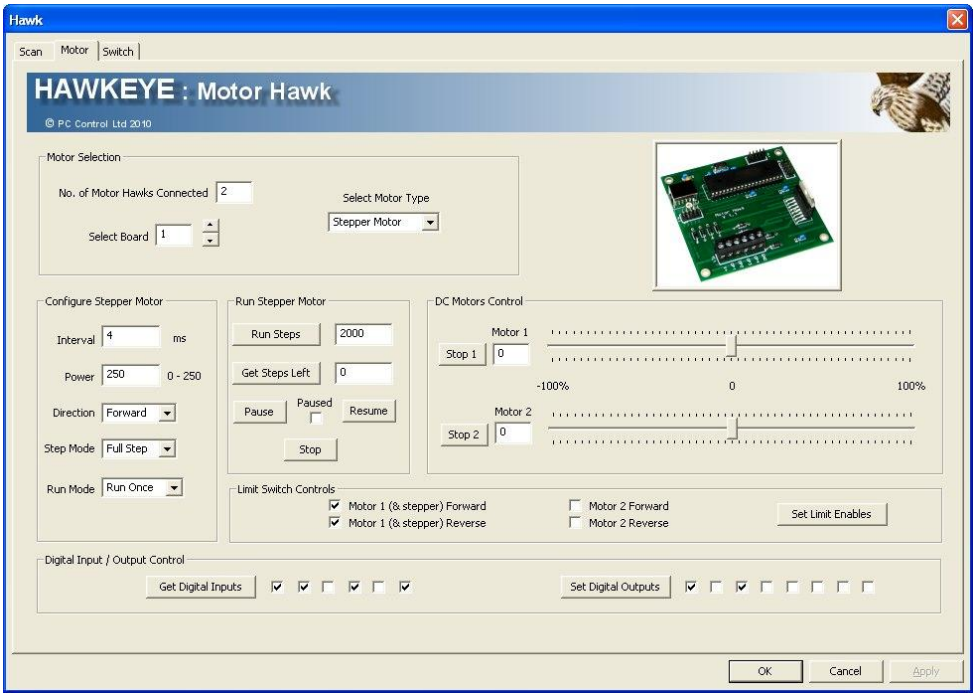
As can be seen, the main application is divided into a range of tabbed screens. The first screen is for scanning the USB ports to identify all attached hawk boards and the other tabs are specific screens for control of each type of hawk board present.

It is necessary to use the “Scan For Devices” button on the scanning page **before** attempting to use any of the controls on any of the other pages. The scanning process not only identifies what boards from the hawk range are present and how many there are of each, but also creates program links to them to make them available for control by the specific tabbed pages associated with that type of board. It is also important to note that you need to return to this page and use the “Scan For Devices” button again if you have added or removed any Hawk board whilst the Hawkeye application is running.

The total number of Hawk boards connected and the number of each type of board will be shown in their respective boxes. To begin controlling one of your attached boards you can now select the appropriate tab for that board.

8.3 Motor Hawk Screen

On selecting the “Motor” tab you will see the following control screen for any attached Motor Hawks.



In the first box you will see how many Motor Hawks are currently connected to the computer. This should be the same as the figure shown on the device scanning page. If there is only one board of this type then the “Select Board” option will already be set to ‘1’ and need not be changed. If you have more than one Motor Hawk then you should select the particular board you want to control using the “Select Board” option. As you change the number in the “Select Board” box , the current settings on the new board selected will automatically be retrieved from the board and displayed on the page, ready for your changes.

As an aside, it is worth mentioning here the board numbering identification system that is used to identify individual boards: Although there is not a specific number attributed to a Motor Hawk board before it is attached to a USB port, it is still easy to be sure which board is which within a group of the same type of board. The numbering system is derived from the way Windows enumerates the USB devices in any system .i.e. normal plugging and unplugging of a USB board will lead to different number assignments by windows. However, if you dedicate a small USB hub to your group of Hawk Boards, you can be assured that the same relative number will be assigned to the same board each time the computer is turned on. i.e. although other USB devices may be plugged and unplugged in your computer system, the hub will still be treated separately and the boards will still receive the same relative numbers. It is the relative numbering that Hawkeye uses to assign its numbers to the boards making it consistent, easy to use and avoiding the usual USB number assignment issues with windows systems.

Before using any other controls you need to specify which type of control you require on your Motor Hawk. i.e. Stepper Motor or Twin DC Motors. Once selected, only the controls associated with that type of motor will be functional. On first “power-up” the motor hawk will have settings that will have the attached motors in an “off” configuration. Once you have chosen the type of motor you are controlling (Stepper or DC Motors) and changed the settings to be active, you should not change the motor type again. Changing the motor type while the current type is active may result in unexpected speed/direction/power changes in the attached motor(s), which, obviously, has the potential to cause damage.

8.3.1 Stepper Motor Control

In order to make the stepper motor perform your specific movement command you must first specify its movement configuration using the "**Configure Stepper Motor**" group of controls. As soon as a change has been made to the relevant option it is immediately sent to the Motor hawk board to update its configuration. A change is recognised by completing and moving the cursor to another entry (e.g. by pressing the "tab" key.

The **Interval** is the time delay between successive motor steps and can be any number in the range 1 to 30000 milliseconds. It should be noted that , although Motor Hawk can deliver the necessary pulse sequences at the fastest rate (i.e. 1ms) the stepper motor may not be able to move in that time. In other words the specifications and capabilities of the specific stepper motor connected need to be taken into consideration when selecting the fastest speeds (shortest intervals). As a benchmark, you can consider the standard stepper motor (MOT003) available in the Control Shop (www.pc-control.co.uk) can successfully operate down to a 3ms interval.

The **Power** is a measure of the relative power delivered to the phase windings of the stepper motor. This can be a number in the range 0 to 250 with 0 corresponding to "power off" and 250 to "maximum power". Motor Hawk uses a principle of pulse width modulation (pwm) to control the power being delivered to the motor from its standard DC power supply. Most of the time you will use either "maximum power" (250) or "power off" (0). Maximum power is used for normal operation and ensures the stepper motor receives a fixed DC voltage equal to the external power supply used. Power off is typically used when the motor is idle and waiting for the next command. If the idle time (time spent stationary) is prolonged, the motor can begin to overheat if the power is constantly on maximum. This is due to the unusual nature of stepper motors where the current consumed while stationary is significantly more than when moving (*ref inductance and electromagnetic theory*). Fractional power settings can be used especially where the movement is slow (large step intervals) and the torque required to move the load is relatively low.

The **Direction** control is simply the choice of direction of rotation (forward or reverse).

Step Mode selection allows the stepper motor to be operated in two different modes. In "Full Step" mode the stepper motor will move one complete step in the chosen direction corresponding to its specified step size. For example a 200 step per revolution motor will move one 200^{th} of a revolution for each step. However, if you select "Half Step", the step size will be half the normal step size of the motor. i.e. the 200 steps per rev motor would then only move one 400^{th} of a revolution for each step. This can be useful for increasing the resolution of the attached motor for a particularly demanding application.

Normally the stepper motor will execute the specified number of steps and then stop and wait for the next command. By using the "**Run Mode**" option this can be changed to "Continuous" operation. In continuous mode the stepper motor will execute the specified number of steps and then immediately re-start the same number in a continuously repeating loop.

The "**Run Stepper Motor**" control section allows you to send a "**Run**" command to the selected Motor Hawk and monitor its progress. To execute a number of steps simply enter that number in the box adjacent to the "Run Steps" button and then click on that button. The stepper motor will start immediately and will use the settings already specified by the configuration already specified in the "Configure Stepper Motor" controls. The number of steps can be any number in the range 1 to 30000. At any time you can view the number of steps remaining by clicking on the "Get Steps Left" button.

The "**Pause**" button will cause the motor to stop running immediately but will leave the steps remaining unchanged. A subsequent click on the "**Resume**" button will re-start the motor with exactly the same settings and number of steps remaining. Pause and resume can be used at any time.

The "**Stop**" button will also stop the motor immediately but it will also clear any remaining steps to zero. To re-start the motor a new "Run Steps" command would then need to be issued.

There are two digital inputs which can be used with the stepper motor to perform a range limiting function. These can be enabled using the "Limit Switch Controls" section. To enable these simply tick the appropriate boxes and then click on the "Set Limit Enables" button. A full description of how these range limit inputs operate can be found in the hardware section of this manual.

8.3.2 DC Motor Control

Up to two DC motors can be used and controlled independently with the Motor Hawk. Once you have selected the motor type as "DC Motors" in the drop down selector box the attached motors can be controlled very simply by moving the slider controls in the "**DC Motors Control**" section of the screen. With the slider in the central '0' position the motor has no power applied to it and will be stationary. To make the motor rotate clockwise (forward) move the slider to the right and anti-clockwise (reverse) by moving it to the left. The clockwise and anti-clockwise are arbitrary. The important point is that the polarity of the voltage applied to the motor is reversed depending on moving the slider to the left or right of centre. The further the slider is moved from the centre the more power is delivered to the motor and the faster it will rotate. Two additional buttons are provided to allow the motor to be stopped quickly without altering the slider. These will immediately stop the associated motor and reduce the slider power to zero.

The **Limit Switch Controls** can also be applied to the DC motors in the same way they were applied to the stepper motor. For example: if the Motor Forward Limit Enable is ticked and the "Set Limit Enables" button is clicked then Motor 1 will automatically stop if digital input 1 is connected to GND (by an appropriately positioned limit switch). A full description of how these range limit inputs operate can be found in the hardware section of this manual.

8.3.3 Digital Outputs

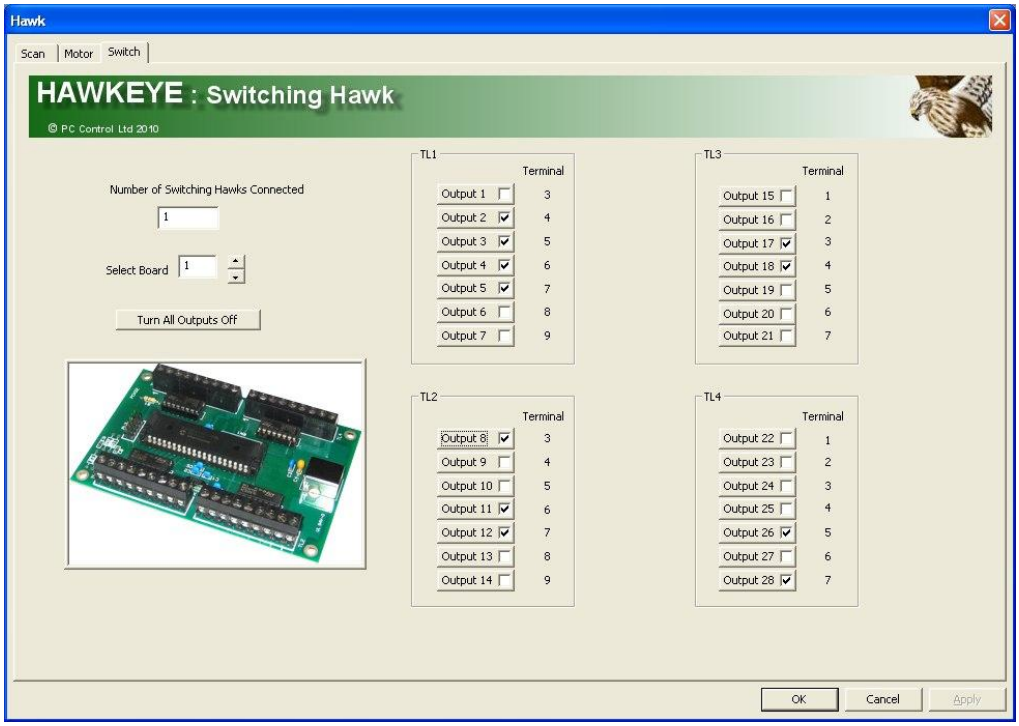
There are 6 independent digital outputs on the Motor Hawk which can be used for controlling external devices. By ticking the appropriate boxes a new pattern of on/off of these outputs will be sent to the Motor Hawk when the "Set Digital Outputs" button is clicked. When a box is ticked the associated output is on (i.e. at +5v , Logic '1') and is off (i.e. at 0v, Logic '0') when unticked.

8.3.4 Digital Inputs

The five digital inputs available on the Motor hawk can be checked at any time by clicking on the "Get Digital Inputs" button. This will show a tick if the input is high / +5v / Logic '1' and unticked when the input is Low / 0v /Logic '0'. This will always be the case even if the input is also being used as a range limit (as discussed above).

8.4 Switching Hawk Screen

On selecting the “Switch” tab you will see the following control screen for any attached Switching Hawks.



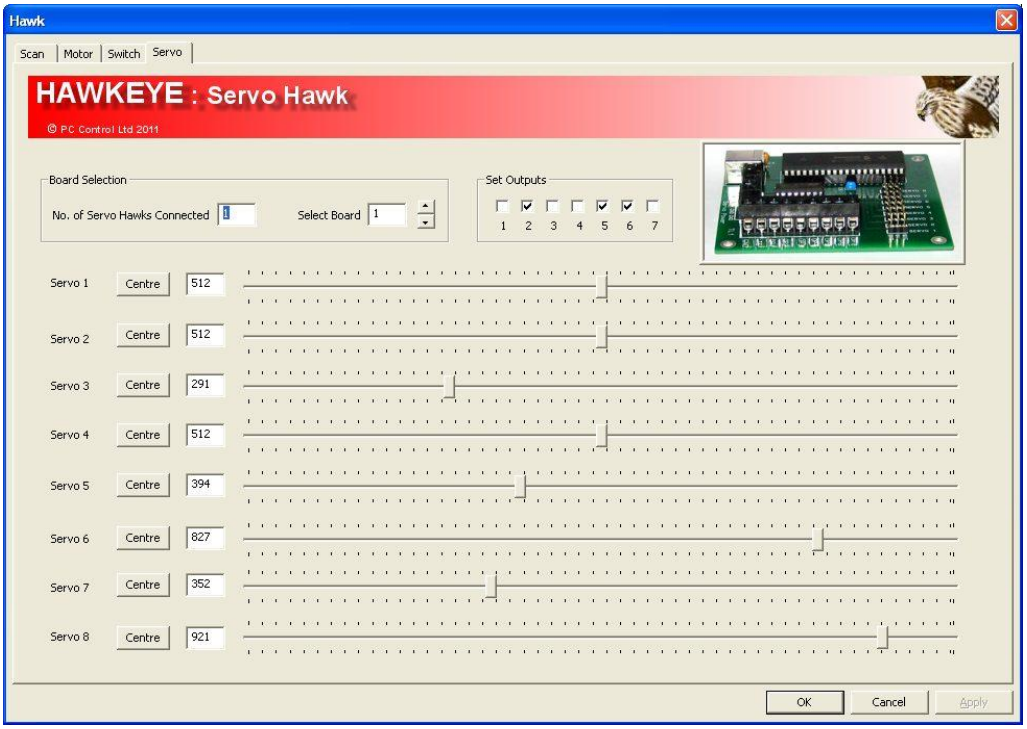
In the first box you will see how many Switching Hawks are currently connected to the computer. This should be the same as the figure shown on the device scanning page. If there is only one board of this type then the “Select Board” option will already be set to ‘1’ and need not be changed. If you have more than one Switching Hawk then you should select the particular board you want to control using the “Select Board” option. As you change the number in the “Select Board” box , the current settings on the new board selected will automatically be retrieved from the board and displayed on the page, ready for your changes.

The switching hawk is a very simple board to control. It has 28 switching type outputs with each of them represented on this page by an appropriately named button with a tick box. An individual output can be turned on or off by simply clicking on the button. The tick will show the current state of the output (tick present indicates the output is on). An additional button is available to turn all of the outputs off at the same time.

Adjacent to each output button is the number of the terminal on the switching hawk that corresponds to that output, making wiring a little easier.

8.5 Servo Hawk Screen

On selecting the “Servo” tab you will see the following control screen for any attached Servo Hawks.



In the first box you will see how many Servo Hawks are currently connected to the computer. This should be the same as the figure shown on the device scanning page. If there is only one board of this type then the “Select Board” option will already be set to ‘1’ and need not be changed. If you have more than one Servo Hawk then you should select the particular board you want to control using the “Select Board” option. As you change the number in the “Select Board” box , the current settings on the new board selected will automatically be retrieved from the board and displayed on the page, ready for your changes.

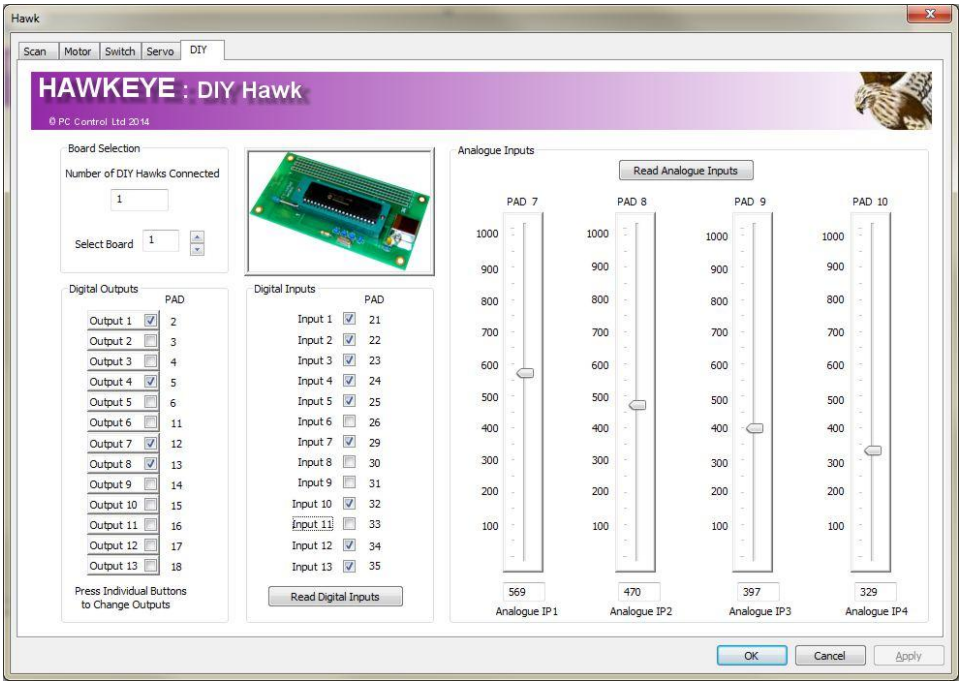
The Servo Hawk is a very simple board to control. It has 8 servo outputs where the position of the servo is controlled by simply moving the associated slider control left and right. When the slider control is in the central position the servo is also in its central position. Specifically, the positioning signal is set to 1.5ms. A numerical indication of this position is shown in the box to the left of each slider control. This number is in the range 0 – 1024 with 512 being the central position. This number range is due to the resolution of the positioning signal being10 bits. i.e. a 10 bit binary number is used to specify each possible position of the servo. When the servo is in its furthest anti-clockwise position (pulse width 1.0ms) the number will be 0 and when furthest clockwise (pulse width 2.0ms) it will be 1024. For convenience, a centre button is provided to allow for quick repositioning to the central (neutral) position.

The 7 available switching outputs on the servo hawk can be turned on and off using the tick boxes in the “Servo Outputs” section. The tick will show the current state of the output (tick present indicates the output is on).

When you have more than one servo hawk connected to the PC the current settings on the board just selected will be retrieved automatically from the board and used to update the slider controls and output settings.

8.6 DIY Hawk Screen

On selecting the “DIY” tab you will see the following control screen for any attached DIY Hawks.



In the first box you will see how many DIY Hawks are currently connected to the computer. This should be the same as the figure shown on the device scanning page. If there is only one board of this type then the “Select Board” option will already be set to ‘1’ and need not be changed. If you have more than one DIY Hawk then you should select the particular board you want to control using the “Select Board” option. As you change the number in the “Select Board” box , the current settings on the new board selected will automatically be retrieved from the board and displayed on the page, ready for your changes.

The DIY hawk has 13 standard digital type outputs with each of them represented on this page by an appropriately named button with a tick box. An individual output can be turned on or off by simply clicking on the button. The tick will show the current state of the output. A tick present indicates the output is "ON" (logic 1 / high / +5v). Adjacent to each output button is the number of the terminal pad on the DIY hawk that corresponds to that output, making connections a little easier.

It also has 13 standard digital type inputs. Each of these is represented by a tick box. When the tick is present the corresponding input is logic 1 / high / +5v. When no tick present the input is logic 0 / low / 0v. The inputs are only updated when the "Read Digital Inputs" button is pressed.

The board has four analogue inputs. These are designed to work with voltages in the range 0 to +5v and will digitise this voltage into a number in the range 0 to 1023. i.e. it is a 10bit converter. The digitised analogue voltage is represented both numerically and on a varying vertical slider. The analogue inputs are only updated when the "Read Analogue Inputs" button is pressed.

The controls and facilities on this screen are only intended to allow you to verify your connected devices are working correctly and, as such, it only provides manual type controls. Obviously your application will probably require your own software to provide full functionality. The supplied DLL function library provides an easy way for you to write your own software for this. Details for this is provided in later sections of this manual.

9. Writing your own software for the Hawk range of boards

To use any of the "Hawk" boards in manual only mode straight from the box does not require any programming other than installing the Hawkeye software and operating the outputs manually. However, once you have verified your system and device connections using the manual facilities of Hawkeye you will want to write your own application specific software. The following information will be of use.

Provided with all Hawk boards is a DLL (dynamic link library) called "hawk.dll". This provides a comprehensive range of functions that allow full program control over all facilities on all hawk boards. The DLL takes care of all of the complexities of USB communications internally and avoids the need for the programmer to get involved in this area. This means that the programmer can concentrate on his particular application without worrying about USB protocols and the mechanisms for handling them. Simple functions such as InitHawk(), SetOutputs() and RunMotor() and many other similar functions, make it very easy to construct powerful and easy to read control programs. Although the DLL was written in 'C' it can be used (called) by programs written in a number of popular languages including (but not limited to) C++, BASIC and Visual BASIC. On the installation disk , you will find two fully working example programs that will help you get "up and running" very quickly with your own program. One is written in Visual Basic and the other in Visual C++. They were both written within the Microsoft Visual Studio 2008 Development platform. If you have this development software, or later versions, you should find that the full workspace supplied will simply load immediately and be ready to compile and run. However, even if you have other development platforms you will find the source code readable in most editors, including the ubiquitous "notepad". In this way, cut and paste operations can be used to populate your own program with the relevant code fragments from the examples.

The following sections describe the DLL functions in more detail and how to use them.

9.1 General Program Structure

Before describing the full range of DLL functions in detail, it is worth looking at the basic structure of an application program intending to use the hawk DLL and associated hawk control boards.

Since one of the major aims of the DLL is to hide the complexities of the USB communications protocols from the control programmer, it is necessary to set up the required handling structure to achieve this prior to using the hawk functions. This is achieved very simply by initialising the hawk environment using the

Sys_Initialise()

function. It takes no parameters and returns a number corresponding to the total number of hawk boards (of any type) connected to USB ports. This should be run at the head of your program and, obviously, only after you have connected your hawk boards to free USB ports. If boards are disconnected or connected while the program is running you will need to call this function again to re-initialise the system (or simply re-run the program).

As a general point to note, the names of the DLL functions reflect their association using the first three letters. i.e. Sys_.... are general system functions, Motor_... are functions to be used with the Motor hawks, Switching_... for switching hawks etc…etc…

To enable an individual hawk board to be subsequently addressed and controlled individually it is necessary to obtain a unique number for it. This is the function of the system function Sys_GetXxxxHawkCount where Xxxx corresponds to the specific board type concerned. For example it would be

Sys_GetMotorHawkCount()

for Motor hawks. These functions have no parameters but return the number of those specific boards connected. This number can then be used to control the specific board. For example , if you only have one board connected then this function will return a count of '1' and , in later functions you will use that number to specify that particular board. If, however, you have 3 boards connected and receive the number '3' from this function, you will control individual boards using either 1, 2 or 3 as the board number in the control functions (described later). There is sometimes confusion over how to identify which board is number 1 and which is number 3 in the above example. This point is best explained by looking, for a moment at

how Windows handles USB devices which may be connected and disconnected at any time from multiple USB ports.

Windows keeps a register of numbers associated with USB devices currently connected to its ports. These numbers can dynamically change as devices are disconnected and re-connected. However, when a number of devices are connected to a particular USB hub, they are always enumerated in the same way and the same order during each and every start up of windows. The absolute numbers may change but the relative order is always maintained.

In order to simplify this system and permit consistent use, the hawk DLL keeps its own record of absolute USB device numbers and translates them into simplified ranges starting from '1' for each type of board connected. In this way you can always be sure that your board numbered '1' will always be the one connected to the first port in a particular hub and '2' the second etc.. etc… It is for this reason that we strongly recommend that you use a small (and generally inexpensive) USB HUB just for the hawk boards when you are using multiple hawk boards in the one system. Although you can connect multiple hawks to the main computer USB ports, it is much harder to recognise which one is which and can also lead to mistakes when you connect a new USB device (eg mouse or disk drive) within the same internal hub serving those boards.

Once you have the board number to use for your connected board, the full range of control functions are then available to you by using this board number as one of the parameters in the relevant function. For example: to set the outputs on the second switching hawk connected to the PC would use….

```
Switching_SetOutputs(2, outputs);
```

Where '2' is the board number and 'outputs' is an integer corresponding to the desired on-off "bit" pattern of the outputs.

With all of the specific board control functions available there is a rigorous check made on the success/failure of the control message reaching the specific board. i.e. internally the DLL USB messaging works two ways. For each command sent to a board there is a USB reply message sent back to the PC confirming successful reception of the message. If there has been any problem with this process the return value of the function will indicate this. The functions will return zero for success or any other number to indicate an error.

The other system function which is required in your program is

```
Sys_CloseAllDevices()
```

This should be used at the end of your program just prior to termination. (this closes all open USB device paths etc…).

It is also important to note that , whatever development system or programming language you are using, you should specify in your compiler settings that the type of functions being called in the DLL follow the standard calling convention. This is usually denoted as "__stdcall" when changing your compiler options. (i.e. __stdcall should be used rather than __fastcall or __cdecl). Your compiler should also be configured to use 32bit integer values (rather than 16 bit).

9.2 Making use of DLL functions from your own program

Before you can make use of the DLL functions described in the next section it is necessary to "link" the DLL to your own program. The first step in this process is to copy the DLL file ("hawk.dll") from the installation disk to a suitable directory on your hard drive. It is recommended that you copy it to the "Windows/System32" directory where it will always be found by your compiler. The DLL can be found in the installation disk subdirectory called "DLL" together with a header file containing some useful type definitions for the functions.

You should consult your own program development environment for specific details of how to call external DLL's but, the two most commonly used techniques are described below. The first is used in C/C++ programs and the second in Basic/Visual Basic programs.

9.2.1 Using the Hawk DLL with C/C++

In C/C++ programs it is necessary to obtain a "handle" to the DLL by "Loading" it prior to use. This is done by using the windows LoadLibrary( ) function as follows.

```
HINSTANCE           HawkDll;        // create a variable to hold the handle for the dll

HawkDll = LoadLibrary("hawk.dll");  // load the DLL and return a valid handle in HawkDll

if(HawkDll != NULL)                 // a NULL value would indicate a problem
{
        …. Dll handle is valid and can be used
}
```

It is always good practice to make sure a valid handle has been obtained to the DLL before using it. If there was a problem loading the DLL (eg. Could not find the DLL file) then the value returned from LoadLibrary() would be NULL (i.e. zero).

Once a valid handle has been obtained you then need to obtain the function pointers to the individual DLL functions using the windows GetProcAddress() function as shown below…

```
// make ProcAddress pointers for each function (here we illustrate with the sys_initialise() function)
Type_Sys_Initialise     Sys_Initialise;  // Type definitions are contained in supplied
                                         // header file "hawdll.h"

// obtaining the function address of the Sys_Initialise dll function
Sys_Initialise = (Type_Sys_Initialise)GetProcAddress( HawkDll, "Sys_Initialise");

if(Sys_Initialise != NULL)
{
  // Dll function has been found and can be used via its pointer

  DeviceCount = Sys_Initialise();       // initialise system and find how many Hawk boards of
                                        // any type are currently connected by calling the DLL
                                        // Sys_Initialise function
}
```

9.2.2 Using the Hawk DLL with Basic / Visual Basic

In Basic /VB programs it is necessary to “declare” the DLL function prior to using it in your program. The function declaration has the following format…

```
Declare Function Sys_Initialise Lib "hawk.dll" () As Integer
```

This also tells the compiler where to find the “hawk.dll” file. i.e. a full pathlist to the hawk.dll file should be used following the Lib part of the declaration unless the file is in one of the special directories where the program will always look (eg the “Windows/System32” directory).

Using the DLL function within the program once it has been declared as shown above is then very simple….

```
DeviceCount = Sys_Initialise()   ‘ get the number of devices connected and initialise the Hawk system
```

Similarly…

```
Declare Function Motor_SetDigitalOutputs Lib "hawk.dll" (ByVal outputs As Integer) As Integer

Dim Outputs As Integer
Dim BoardNumber As Integer

BoardNumber = 1                                  ‘ Use motor hawk number 1
Outputs = 7                                      ‘ 7 is 00000111 in binary

Motor_SetOutputs(BoardNumber, Outputs)           ‘ turn on the first three digital
                                                 ‘ outputs of  the motor hawk
```

9.3 DLL Functions Reference

This section will explain the details of each function provided within the DLL library and how to use them from within your own program. Starting with the systems functions not specific to any board followed by a list of the DLL functions grouped by applicable board type. It should be noted that the declaration statement for each function has not been shown but all follow the conventions described in section 7.2

Function Name: Sys_Initialise
Applicable to: General System
Syntax 'C / C++': int Sys_Initialise()
Syntax 'Basic/VB': Sys_ Initialise () As Integer
Return Value: Returns the total number of Hawk boards (of any type) currently connected to USB ports.

Description: This function must be called prior to any other DLL function being used. Its function is to create the DLL USB communications environment and prepare the system to accept all other function calls. It should only be called once in a program unless one of the devices being controlled is disconnected or another board has been connected to a USB port while the program is running. If this happens it is necessary to call it again (*and to also call the individual device count functions described later*).

Usage "C / C++":

```
int error;

error = Sys_Initialise();
if(error == 0)
{
        ……  Main Program
}
else
{
        ……. Report error and stop
}
```

Usage "Basic / Visual Basic":

```
Dim error As Integer

error = Sys_Initialise()

If error = 0 Then
…….. Main Program
Else
…….. Report error
End If
```

Function Name: Sys_GetMotorHawkCount()
Applicable to: General System
Syntax 'C / C++': int Sys_GetMotorHawkCount()
Syntax 'Basic/VB': Sys_ GetMotorHawkCount () As Integer

Return Value: The number of motor hawk boards currently connected to USB ports

Description: Once the system has been initialised this function can be called to find out how many motor hawk boards are currently connected to USB ports. The number returned is important for later identifying individual boards. For example: If the number returned is 3 (i.e. 3 motor hawks connected) then the only valid board numbers that can be used in the Motor Hawk specific functions (described later) are numbers 1, 2 and 3. Assuming a static USB configuration within a hub, each of these numbers will then always refer to the same specific board.

Note: If a board is disconnected (or an additional one is connected) while the main program is running then this function needs to be called again to establish the new numbers applicable (after also calling the Sys_Initialise() function again).

Usage "C / C++":

```
int count;

count = Sys_ GetMotorHawkCount ();
        ……  Main Program
```

Usage "Basic / Visual Basic":

```
Dim count As Integer

count = Sys_ GetMotorHawkCount ()
…….. Main Program
```

Function Name: Sys_GetSwitchingHawkCount()
Applicable to: General System
Syntax 'C / C++': int Sys_GetSwitchingHawkCount()
Syntax 'Basic/VB': Sys_ GetSwitchingHawkCount () As Integer

Return Value: The number of switching hawk boards currently connected to USB ports

Description: Once the system has been initialised this function can be called to find out how many switching hawk boards are currently connected to USB ports. The number returned is important for later identifying individual boards. For example: If the number returned is 3 (i.e. 3 switching hawks connected) then the only valid board numbers that can be used in the Switching Hawk specific functions (described later) are numbers 1, 2 and 3. Assuming a static USB configuration within a hub, each of these numbers will then always refer to the same specific board.

Note: If a board is disconnected (or an additional one is connected) while the main program is running then this function needs to be called again to establish the new numbers applicable (after also calling the Sys_Initialise() function again).

Usage "C / C++":

```
int count;

count = Sys_ GetSwitchingHawkCount ();
        …… Main Program
```

Usage "Basic / Visual Basic":

```
Dim count As Integer

count = Sys_ GetSwitchingHawkCount ()
…….. Main Program
```

Function Name: Sys_GetServoHawkCount()
Applicable to: General System
Syntax 'C / C++': int Sys_GetServoHawkCount()
Syntax 'Basic/VB': Sys_ GetServoHawkCount () As Integer

Return Value: The number of servo hawk boards currently connected to USB ports

Description: Once the system has been initialised this function can be called to find out how many servo hawk boards are currently connected to USB ports. The number returned is important for later identifying individual boards. For example: If the number returned is 3 (i.e. 3 servo hawks connected) then the only valid board numbers that can be used in the Servo Hawk specific functions (described later) are numbers 1, 2 and 3. Assuming a static USB configuration within a hub, each of these numbers will then always refer to the same specific board.

Note: If a board is disconnected (or an additional one is connected) while the main program is running then this function needs to be called again to establish the new numbers applicable (after also calling the Sys_Initialise() function again).

Usage "C / C++":

```
int count;

count = Sys_ GetServoHawkCount ();
        …… Main Program
```

Usage "Basic / Visual Basic":

```
Dim count As Integer

count = Sys_ GetServoHawkCount ()
…….. Main Program
```

Function Name: Sys_GetDiyHawkCount()
Applicable to: General System
Syntax 'C / C++': int Sys_GetDiyHawkCount()
Syntax 'Basic/VB': Sys_ GetDiyHawkCount () As Integer

Return Value: The number of DIY hawk boards currently connected to USB ports

Description: Once the system has been initialised this function can be called to find out how many DIY hawk boards are currently connected to USB ports. The number returned is important for later identifying individual boards. For example: If the number returned is 3 (i.e. 3 DIY hawks connected) then the only valid board numbers that can be used in the DIY Hawk specific functions (described later) are numbers 1, 2 and 3. Assuming a static USB configuration within a hub, each of these numbers will then always refer to the same specific board.

Note: If a board is disconnected (or an additional one is connected) while the main program is running then this function needs to be called again to establish the new numbers applicable (after also calling the Sys_Initialise() function again).

Usage "C / C++":

```
int count;

count = Sys_ GetDiyHawkCount ();
        ……  Main Program
```

Usage "Basic / Visual Basic":

```
Dim count As Integer

count = Sys_ GetDiyHawkCount ()
…….. Main Program
```

Function Name: Sys_CloseAllDevices
Applicable to: General System
Syntax 'C / C++': int Sys_CloseAllDevices()
Syntax 'Basic/VB': Sys_ CloseAllDevices () As Integer
Return Value: The number of access paths to hawk boards (of any type) that were successfully closed.

Description: This function should be called just prior to terminating the application or when access to the attached hawk boards is no longer needed. The return value indicates the number of hawk boards that have had their access paths successfully closed. This number should be the same as the number returned by the Sys_Initialise( ) function.

Usage "C / C++":

```
int count;

…… Main Program
count = Sys_ CloseAllDevices ();
```

Usage "Basic / Visual Basic":

```
Dim count As Integer

…….. Main Program
count = Sys_ CloseAllDevices ()
```

Function Name: Motor_SetType
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_SetType (int BoardNumber, int Type)
Syntax 'Basic/VB': Motor_SetType (ByVal BoardNumber As Integer,
ByVal Type As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Type A positive integer with a value of 0 or 1 to configure the board for either stepper motor operation or DC motors: '0' for stepper motor and '1' for DC motors.

Description: This function configures the on board electronics of the motor hawk to drive either stepper motors or DC motors. This should be called near the start of your program but after the initialisation and obtaining the hawk board numbers. i.e. the motor type needs to be specified before subsequent calls to functions that define the speed, direction, steps etc. of any motors connected. NOTE: A change to the motor type configuration after one or motors is active could lead to unexpected speed/direction/power changes which has the potential to cause damage.

Usage "C / C++":

```
int BoardNumber, Type;
int Error;

BoardNumber = 3;
Type = TYPE_STEPPER;        // set for stepper motors (TYPE_STEPPER defined in hawkdll.h)

Error = Motor_SetType ( BoardNumber, Type);

If(Error == 0)
{
…. All other Motor Hawk functions for stepper motors for this board can now be used

}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Type As Integer
Dim Error As Integer

BoardNumber = 2
Type = TYPE_DCMOTORS     ' set for DC motors (TYPE_DCMOTORS defined in hawkdll.h)

Error = Motor_SetType ( BoardNumber, Type)

If Error = 0 Then
…. All other Motor Hawk functions for DC motors for this board can now be used
End If
```

Function Name: Motor_RunSteps
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_RunSteps (int BoardNumber, int Steps)
Syntax 'Basic/VB': Motor_ RunSteps (ByVal BoardNumber As Integer,
ByVal Steps As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Steps A positive 32 bit integer with a value in the range 1 to 0x7fffffff (Hex) corresponding to the number of steps to be executed.

Description: Sends a command to the Motor Hawk specified by BoardNumber to begin running the specified number of steps. The direction of rotation and interval between steps would have already been set using the SetStepper() function. This function returns immediately it completes the delivery of the command to the Motor Hawk; i.e. it does not wait until the steps are completed. It assumes the board has already been configured as a Stepper Motor controller (using SetType()). The command will be ignored by the Motor Hawk if it has been configured for DC motor operation.

Usage "C / C++":

```
int BoardNumber;
int Steps;
int Error;

BoardNumber = 2;        // may be any valid board number of those available

Steps = 100;            // any number in the range 1 – 2,147,483,647
                        // (i.e. largest positive 32 bit number 0x7FFFFFFF)

Error = Motor_ RunSteps( BoardNumber, Steps);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Steps As Integer
Dim Error As Integer

BoardNumber = 1         ' may be any valid board number of those available

Steps = 100             ' any number in the range 1 – 2,147,483,647
                        ' (i.e. largest positive 32 bit number 0x7FFFFFFF)

Error = Motor_ RunSteps( BoardNumber, Steps)
```

Function Name: Motor_SetDCMotors
Applicable to: Motor Hawk
Syntax 'C/C++':
```
int Motor_SetDCMotors (     int BoardNumber,
                            int M1Speed, Int M1Dir,
                            int M2Speed, Int M2Dir  )
```

Syntax 'Basic/VB':
```
Motor_ SetDCMotors ( ByVal BoardNumber As Integer,
                     ByVal M1Speed As Integer,
                     ByVal M1Dir As Integer,
                     ByVal M2Speed As Integer,
                     ByVal M2Dir As Integer ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Speed A positive integer in the range 0 – 250 which corresponds to the speed of rotation of the motor

Dir A positive integer specifying the direction of rotation with a value of '0' for Forward, '1' for Reverse.

Description: Sends a command to the Motor Hawk specified by BoardNumber to set the speed and direction of both DC motors connected. . It should be noted that the speed setting is more accurately called a power setting as it controls the delivered power to the motor. The varying power then varies the speed approximately in proportion.

It assumes the board has already been configured as a DC Motor controller using SetType( ). The command will be ignored by the Motor Hawk if it has been configured for stepper motor operation.

Usage "C / C++":
```
int BoardNumber;
int M1Speed, M1Dir, M2Speed, M2Dir;
int Error;

BoardNumber = 1;            // may be any valid board number of those available

M1Dir = MHK_FORWARD;        // direction constants defined in hawkdll.h
                            // (0 is stopped, 1 is forward and 2 is reverse)
M1Speed = 128;              // speed is in the range 0 – 255

M2Dir = MHK_STOPPED;        // motor 2 will be stationary here even though
                            // the speed is set to 128 below
M1Speed = 128;

Error = Motor_ SetDCMotors( BoardNumber, M1Speed, M1Dir, M2Speed, M2Dir);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim M1Speed As Integer
Dim M1Dir As Integer
Dim M2Speed As Integer
Dim M2Dir As Integer
Dim Error As Integer

BoardNumber = 1              ‘ may be any valid board number of those available

M1Dir = MHK_REVERSE          ‘ direction constants defined in hawkdll.h
                             ‘ (0 is stopped, 1 is forward and 2 is reverse)
M1Speed = 100                ‘ speed is in the range 0 – 255

M2Dir = MHK_STOPPED          ‘ motor 2 will be stationary here even though
                             ‘ the speed is set to 200 below
M1Speed = 200

Error = Motor_ SetDCMotors( BoardNumber, M1Speed, M1Dir, M2Speed, M2Dir)
```

Function Name: Motor_GetDCMotors
Applicable to: Motor Hawk
Syntax 'C/C++':
```
int Motor_GetDCMotors (     int BoardNumber,
                            int *M1Speed, Int *M1Dir,
                            int *M2Speed, Int *M2Dir  )
```

Syntax 'Basic/VB':
```
Motor_ GetDCMotors ( ByVal BoardNumber As Integer,
                     ByRef M1Speed As Integer,
                     ByRef M1Dir As Integer,
                     ByRef M2Speed As Integer,
                     ByRef M2Dir As Integer ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

M1Speed A positive integer in the range 0 – 250 which corresponds to the speed of rotation of the motor 1

M1Dir A positive integer indicating the direction of rotation of motor 1with a value of '0' for Forward, '1' for Reverse.

M2Speed A positive integer in the range 0 – 250 which corresponds to the speed of rotation of the motor 2

M2Dir A positive integer indicating the direction of rotation of motor 2with a value of '0' for Forward, '1' for Reverse.

Description: Get the speed and direction of both DC motors connected, previously set using the SetDCMotors() function.

It assumes the board has already been configured as a DC Motor controller using SetType( ). Iif it has been configured for stepper motor operation the returned values are undefined.

Usage "C / C++":

```
int BoardNumber;
int M1Speed, M1Dir, M2Speed, M2Dir;
int Error;

BoardNumber = 1;                // may be any valid board number of those available

Error = Motor_ GetDCMotors( BoardNumber, &M1Speed, &M1Dir, &M2Speed, &M2Dir);

if(Error == 0)
{
   M1Speed, M1Dir, M2Speed and M2Dir will now contain valid
  indications of speed and direction of both motors

}
else
{
 process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim M1Speed As Integer
Dim M1Dir As Integer
Dim M2Speed As Integer
Dim M2Dir As Integer
Dim Error As Integer

BoardNumber = 1                 ‘ may be any valid board number of those available


Error = Motor_ GetDCMotors( BoardNumber, M1Speed, M1Dir, M2Speed, M2Dir)

If Error = 0 Then
   M1Speed, M1Dir, M2Speed and M2Dir will now contain valid
  indications of speed and direction of both motors
Else
  process the error
Endif
```

Function Name: Motor_GetType
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_GetType (int BoardNumber, int *Type)
Syntax 'Basic/VB': Motor_GetType (ByVal BoardNumber As Integer,
ByRef Type As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards currently connected.

Type A positive integer with a value of 0 or 1 indicating the current setting for motor type. '0' indicates a stepper motor and '1' indicates DC motors.

Description: This function obtains the current motor type configuration (stepper motor or DC motors)

Usage "C / C++":

```
int BoardNumber, Type;
int Error;

BoardNumber = 3;

Error = Motor_GetType ( BoardNumber, &Type);

if(Error == 0)
{
.... Type now contains the current motor type configuration (TYPE_DCMOTORS or
TYPE_STEPPER) defined in hawkdll.h
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Type As Integer
Dim Error As Integer

BoardNumber = 2

Error = Motor_GetType ( BoardNumber, Type)

If Error = 0 Then
.... Type now contains the current motor type configuration (TYPE_DCMOTORS or
TYPE_STEPPER) defined in hawkdll.h
Else
  process the error
Endif
```

Function Name: Motor_GetStepper
Applicable to: Motor Hawk
Syntax 'C/C++':

```
int Motor_GetStepper (int BoardNumber,
                      int *Direction,
                      int *Interval,
                      int *StepMode,
                      int *Power,
                      int *Steps,
                      int *RunMode,
                      int *Paused          )
```

Syntax 'Basic/VB':

```
Motor_GetStepper (    ByVal BoardNumber As Integer,
                      ByRef Direction As Integer,
                      ByRef Interval As Integer,
                      ByRef StepMode As Integer,
                      ByRef Power As Integer,
                      ByRef Steps As Integer,
                      ByRef RunMode As Integer,
                      ByRef Paused As Integer
                                          ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards currently connected.

Direction has one of the following integer values: '0' for reverse and '1' for forward

Interval is the time interval in ms and is an integer in the range 1 to 30000

StepMode has the following integer values: '1' for "full step mode" and '2' for "half step mode"

Power is a measure of the power delivered to the stepper motor phase coils and is an integer in the range 0 to 250. The higher the power setting the higher the achievable torque

Steps This is the number of steps still remaining to be completed and is a full positive 32 bit integer in the range 0 to 2,147,483,647 (0x7fffffff Hex).

RunMode has the following integer values: '0' executes the specified number of steps then stops, '1' executes the specified number of steps then restarts the same number of steps (effectively causing the motor to run continuously)

Paused has the following integer values: '0' for "Running Normally" and '1' for "Paused"

Description: This function obtains the current settings for the stepper motor from the specified board.

Usage "C / C++":

```
int BoardNumber;
int Error, Direction, Interval, StepMode, Power, Steps, RunMode, Paused;

BoardNumber = 3;

Error = Motor_GetStepper (BoardNumber, &Direction, &Interval, &StepMode, &Power, &Steps, &RunMode, &Paused );


if(Error == 0)
{
.... Direction, Interval, StepMode, Power, Steps, RunMode and Paused now contain the current stepper motor settings
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Direction As Integer
Dim Interval As Integer
Dim StepMode As Integer
Dim Power As Integer
Dim Steps As Integer
Dim RunMode As Integer
Dim Paused As Integer
Dim Error As Integer

BoardNumber = 2

Error = Motor_GetStepper (BoardNumber,  Direction,  Interval,  StepMode,  Power,  Steps, RunMode, Paused)

If Error = 0 Then
.... Direction, Interval, StepMode, Power, Steps, RunMode and Paused now contain the current stepper motor settings
Else
  process the error
Endif
```

Function Name: Motor_GetStepsRemaining
Applicable to: Motor Hawk
Syntax 'C/C++':
```
int Motor_GetStepsRemaining(int BoardNumber,
                            int *StepsRemaining  )
```

Syntax 'Basic/VB':
```
Motor_GetStepper (      ByVal BoardNumber As Integer,
                        ByRef StepsRemaining As Integer
                                                ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards currently connected.

StepsRemaining A 32 bit integer number in the range 0 to 2,147,483,647 (0x7fffffff Hex) corresponding to the number of steps still to be executed following the last "RunSteps" command.

Description: This function obtains the number of steps still to be executed following the last "RunSteps" function call.

Usage "C / C++":
```
int BoardNumber;
int Error, StepsRemaining ;

BoardNumber = 4;

Error = Motor_ GetStepsRemaining (BoardNumber,  &StepsRemaining  );

if(Error == 0)
{
.... StepsRemaining now contains the number of steps still to be completed
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":
```
Dim BoardNumber As Integer
Dim StepsRemaining As Integer

BoardNumber = 7

Error = Motor_GetStepsRemaining (BoardNumber, StepsRemaining)

If Error = 0 Then
.... StepsRemaining now contains the number of steps still to be completed
Else
  process the error
Endif
```

Function Name: Motor_SetDigitalOutputs
Applicable to: Motor Hawk

Syntax 'C/C++': int Motor_SetDigitalOutputs (int BoardNumber, int Outputs)
Syntax 'Basic/VB': Motor_ SetDigitalOutputs (ByVal BoardNumber As Integer,
ByVal Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x000000ff (Hex) corresponding to the on/off pattern of digital outputs. Only the first 8 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit7 to output 8. When the appropriate bit is a '1' the output is on (logic 1 / high / +5v), when it is a '0' the output is off (logic 0 / low / 0v).

Description: Sends a command to the Motor Hawk specified by BoardNumber to set the digital outputs to the on/off pattern specified.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;

BoardNumber = 2;          // may be any valid board number of those available

Outputs  = 0x0F;          // any number in the range 0 – 255  (0 – 0xFF Hex)

Error = Motor_ SetDigitalOutputs ( BoardNumber, Outputs);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer

BoardNumber = 1           ' may be any valid board number of those available

Outputs = 15              'any number in the range 0 – 255  (0 – 0xFF Hex)

Error = Motor_ SetDigitalOutputs ( BoardNumber, Outputs)
```

Function Name: Motor_GetDigitalOutputs
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_GetDigitalOutputs (int BoardNumber, int *Outputs)
Syntax 'Basic/VB': Motor_ GetDigitalOutputs (ByVal BoardNumber As Integer,
ByRef Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x000000ff (Hex) corresponding to the on/off pattern of digital outputs. Only the first 8 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit7 to output 8. When the appropriate bit is a '1' the output is on (logic 1 / high / +5v), when it is a '0' the output is off (logic 0 / low / 0v).

Description: Sends a command to the Motor Hawk specified by BoardNumber to get the current on/off state of the digital outputs.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;

BoardNumber = 2;                // may be any valid board number of those available

Error = Motor_ GetDigitalOutputs ( BoardNumber, &Outputs);

if(Error == 0)
{
.... Outputs now contains the 8 bit binary pattern (on/off)  of the digital outputs
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer

BoardNumber = 1                 ' may be any valid board number of those available

Error = Motor_ GetDigitalOutputs ( BoardNumber, Outputs)

If Error = 0 Then
.... Outputs now contains the 8 bit binary pattern (on/off)  of the digital outputs
Else
  process the error
Endif
```

Function Name: Motor_GetDigitalInputs
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_GetDigitalInputs (int BoardNumber, int *Inputs)
Syntax 'Basic/VB': Motor_ GetDigitalInputs (ByVal BoardNumber As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Inputs A positive 32 bit integer with a value in the range 0 to 0x0000003f (Hex) corresponding to the on/off pattern of digital inputs. Only the first 6 bits of the inputs parameter are used. Bit 0 corresponds to input1, Bit1 to input 2 …. Bit5 to input 6. When the appropriate bit is a '1' the input is at logic 1 / high / +5v, when it is a '0' the input is at logic 0 / low / 0v.

Description: Sends a command to the Motor Hawk specified by BoardNumber to get the current state of the digital inputs.

Usage "C / C++":

```
int BoardNumber;
int Inputs;
int Error;

BoardNumber = 2;                // may be any valid board number of those available

Error = Motor_ GetDigitalInputs ( BoardNumber, &Inputs);

if(Error == 0)
{
…. Inputs now contains the 6 bit binary pattern of the digital inputs
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Inputs As Integer
Dim Error As Integer

BoardNumber = 1                 ' may be any valid board number of those available

Error = Motor_ GetDigitalInputs ( BoardNumber, Inputs)

If Error = 0 Then
…. Inputs now contains the 6 bit binary pattern of the digital inputs
Else
  process the error
Endif
```

Function Name: Motor_SetLimitEnables
Applicable to: Motor Hawk
Syntax 'C/C++':
```
int Motor_SetLimitEnables (     int BoardNumber,
                                int M1Forward, int M1Reverse,
                                int M2Forward, int M2Reverse  )
```

Syntax 'Basic/VB':
```
Motor_ SetLimitEnables ( ByVal BoardNumber As Integer,
                         ByVal M1Forward As Integer,
                         ByVal M1Reverse As Integer,
                         ByVal M2Forward As Integer,
                         ByVal M2Reverse As Integer
                                                ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

M1Forward A positive integer with a value of 0 or 1 to configure the "Motor 1 Forward" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

M1Reverse A positive integer with a value of 0 or 1 to configure the "Motor 1 Reverse" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

M2Forward A positive integer with a value of 0 or 1 to configure the "Motor 2 Forward" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

M2Reverse A positive integer with a value of 0 or 1 to configure the "Motor 2 Reverse" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

Description: Sends a command to the Motor Hawk specified by BoardNumber to configure the operation of the limit switch inputs. When a limit switch input is active, the motor is "stopped" whenever the input is switched to GND (0v). There are separate limit switch inputs for forward and reverse operation for each motor.

It assumes the board has already been configured as a DC Motor controller using SetType( ). The command will be ignored by the Motor Hawk if it has been configured for stepper motor operation.

Usage "C / C++":

```
int BoardNumber;
int M1Forward, M1Reverse, M2Forward, M2Reverse;
int Error;

BoardNumber = 1;        // may be any valid board number of those available

M1Forward = 1;          // Motor 1 forward limit switch input active
M1Reverse = 0;          // Motor 1 reverse limit switch input ignored
M2Forward = 0;          // Motor 2 forward limit switch input ignored
M2Reverse = 1;          // Motor 2 reverse limit switch input active

Error = Motor_ SetLimitEnables( BoardNumber, M1Forward, M1Reverse, M2Forward, M2Reverse );
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim M1Forward As Integer
Dim M1Reverse As Integer
Dim M2Forward As Integer
Dim M2Reverse As Integer
Dim Error As Integer

BoardNumber = 1          ‘ may be any valid board number of those available

M1Forward = 1            ‘ Motor 1 forward limit switch input active
M1Reverse = 0            ‘ Motor 1 reverse limit switch input ignored
M2Forward = 0            ‘ Motor 2 forward limit switch input ignored
M2Reverse = 1            ‘ Motor 2 reverse limit switch input active

Error = Motor_SetLimitEnables( BoardNumber, M1Forward, M1Reverse, M2Forward, M2Reverse)
```

Function Name: Motor_GetLimitEnables
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_GetLimitEnables (int BoardNumber,
int *M1Forward, int *M1Reverse,
int *M2Forward, int *M2Reverse)

Syntax 'Basic/VB': Motor_ GetLimitEnables (ByVal BoardNumber As Integer,
ByRef M1Forward As Integer,
ByRef M1Reverse As Integer,
ByRef M2Forward As Integer,
ByRef M2Reverse As Integer
) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

M1Forward A positive integer with a value of 0 or 1 to configure the "Motor 1 Forward" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

M1Reverse A positive integer with a value of 0 or 1 to configure the "Motor 1 Reverse" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

M2Forward A positive integer with a value of 0 or 1 to configure the "Motor 2 Forward" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

M2Reverse A positive integer with a value of 0 or 1 to configure the "Motor 2 Reverse" limit switch input operation. '0' for limit switch input "ignored" and '1' for limit switch input "active".

Description: Sends a command to the Motor Hawk specified by BoardNumber to get the current configuration of the limit switch inputs. When a limit switch input is active, the motor is "stopped" whenever the input is switched to GND (0v). There are separate limit switch inputs for forward and reverse operation for each motor.

It assumes the board has already been configured as a DC Motor controller using SetType( ). The command will be ignored by the Motor Hawk if it has been configured for stepper motor operation.

Usage "C / C++":

```
int BoardNumber;
int M1Forward, M1Reverse, M2Forward, M2Reverse;
int Error;

BoardNumber = 1;                // may be any valid board number of those available
Error = Motor_ GetLimitEnables( BoardNumber, &M1Forward, &M1Reverse, &M2Forward,
&M2Reverse );

if(Error == 0)
{
.... M1Forward, M1Reverse, M2Forward and M2Reverse now contain the current configuration of the
limit switch enables
}
else
{
 process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim M1Forward As Integer
Dim M1Reverse As Integer
Dim M2Forward As Integer
Dim M2Reverse As Integer
Dim Error As Integer

BoardNumber = 1                ‘ may be any valid board number of those available

Error = Motor_GetLimitEnables( BoardNumber, M1Forward, M1Reverse, M2Forward, M2Reverse)

If Error = 0 Then
.... M1Forward, M1Reverse, M2Forward and M2Reverse now contain the current configuration of the
limit switch enables
Else
  process the error
Endif
```

Function Name: Motor_SetStepper
Applicable to: Motor Hawk
Syntax 'C/C++':

```
int Motor_SetStepper (int BoardNumber,
                      int Direction,
                      int Interval,
                      int StepMode,
                      int Power
                      )
```

Syntax 'Basic/VB':

```
Motor_SetStepper (    ByVal BoardNumber As Integer,
                      ByVal Direction As Integer,
                      ByVal Interval As Integer,
                      ByVal StepMode As Integer,
                      ByVal Power As Integer
                      ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards currently connected.

Direction has one of the following integer values: '0' for reverse and '1' for forward

Interval is the time interval in ms and is an integer in the range 1 to 30000

StepMode has the following integer values: '1' for "full step mode" and '2' for "half step mode"

Power is a measure of the power delivered to the stepper motor phase coils and is an integer in the range 0 to 250. The higher the power setting the higher the achievable torque

Description: This function configures the stepper motor prior to the use of RunSteps(). It sets the direction of rotation, the interval between steps, the step mode (full or half) and the delivered power. Note: It does not cause the stepper motor to execute any steps: this is the function of RunSteps(). This function can be used prior to any movement of the stepper motor or while the stepper motor is in operation. For example the speed of movement can be changed by changing the Interval whilst rotating. It has no effect on the number of steps remaining to be executed.

Usage "C / C++":

```
int BoardNumber;
int Error, Direction, Interval, StepMode, Power;

BoardNumber = 3;

Direction = 1;          // forward
Interval = 100;         // 100ms between steps
StepMode = 0;           // full step mode
Power = 250;            // maximum power


Error = Motor_SetStepper (BoardNumber, Direction, Interval, StepMode, Power );
```

Usage “Basic / Visual Basic”:

```
Dim BoardNumber As Integer
Dim Direction As Integer
Dim Interval As Integer
Dim StepMode As Integer
Dim Power As Integer
Dim Error As Integer

BoardNumber = 2
Direction = 1            ‘ forward
Interval = 100           ‘ 100ms between steps
StepMode = 0             ‘ full step mode
Power = 250              ‘ maximum power

Error = Motor_SetStepper (BoardNumber, Direction, Interval, StepMode, Power)
```

Function Name: Motor_PauseStepper
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_PauseStepper (int BoardNumber)
Syntax 'Basic/VB': Motor_ PauseStepper (ByVal BoardNumber As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Description: Pauses the rotation of the stepper motor and leaves the steps remaining to be executed unchanged. If the MotorResumeStepper() function is used after this, the stepper will resume rotation with the same number of steps remaining. No changes are made to the stepper configuration (i.e. interval, direction or power)

Usage "C / C++":

```
int BoardNumber;
int Error;

BoardNumber = 3;

Error = Motor_ PauseStepper ( BoardNumber);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Error As Integer

BoardNumber = 2

Error = Motor_ PauseStepper ( BoardNumber)
```

Function Name: Motor_ResumeStepper
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_ResumeStepper (int BoardNumber)
Syntax 'Basic/VB': Motor_ ResumeStepper (ByVal BoardNumber As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Description: Resumes the rotation of the stepper motor following a "Motor_PauseStepper()" function call. The stepper will resume rotation with the same number of steps remaining. No changes are made to the stepper configuration (i.e. interval, direction or power). If the stepper motor had already completed any steps remaining prior to this call then the function call will have no effect. However, the pause would have to be removed before any subsequent RunSteps() commands are effective.

Usage "C / C++":

```
int BoardNumber;
int Error;

BoardNumber = 3;

Error = Motor_ ResumeStepper ( BoardNumber);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Error As Integer

BoardNumber = 2

Error = Motor_ ResumeStepper ( BoardNumber)
```

Function Name: Motor_StopStepper
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_StopStepper (int BoardNumber)
Syntax 'Basic/VB': Motor_ StopStepper (ByVal BoardNumber As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Description: Stops the stepper motor immediately and clears any remaining steps to be executed. No changes are made to the stepper configuration (i.e. interval, direction or power)

Usage "C / C++":

```
int BoardNumber;
int Error;

BoardNumber = 3;

Error = Motor_ StopStepper ( BoardNumber);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Error As Integer

BoardNumber = 2

Error = Motor_ StopStepper ( BoardNumber)
```

Function Name: Motor_SetRunMode
Applicable to: Motor Hawk
Syntax 'C/C++': int Motor_SetRunMode (int BoardNumber, int RunMode)
Syntax 'Basic/VB': Motor_ SetRunMode (ByVal BoardNumber As Integer,
ByVal RunMode As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

RunMode A positive integer with a value of 0 or 1 to configure the "Run Mode". '0' for "run once and '1' for "run continuous.

Description: Sends a command to the Motor Hawk specified by BoardNumber to set "Run Mode" to "Once" or "Continuous". When set "Run Once" the stepper motor will complete its required number of steps and then stop. When set to "Run Continuous" the stepper will complete its required number of steps and then re-start them automatically; effectively running continuously

Usage "C / C++":

```
int BoardNumber;
int RunMode;
int Error;

BoardNumber = 2;          // may be any valid board number of those available

RunMode  = 0;             // run once

Error = Motor_ SetRunMode ( BoardNumber, RunMode);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim RunMode As Integer
Dim Error As Integer

BoardNumber = 1           ' may be any valid board number of those available

RunMode = 1               ' run continuous

Error = Motor_ SetRunMode ( BoardNumber, RunMode)
```

Function Name: Switching_SetOutputs
Applicable to: Switching Hawk
Syntax 'C/C++': int Switching_SetOutputs (int BoardNumber, int Outputs)
Syntax 'Basic/VB': Switching_ SetOutputs (ByVal BoardNumber As Integer,
ByVal Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x0fffffff (Hex) corresponding to the on/off pattern of switching outputs. Only the first 28 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit27 to output 28. When the appropriate bit is a '1' the output is on, when it is a '0' the output is off.

Description: Sends a command to the Switching Hawk specified by BoardNumber to set the switching outputs to the on/off pattern specified.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;

BoardNumber = 2;            // may be any valid board number of those available

Outputs  = 0x0F;            // outputs 1,2,3 and 4 on

Error = Switching_SetOutputs ( BoardNumber, Outputs);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer

BoardNumber = 1             ' may be any valid board number of those available

Outputs = 15                'outputs 1,2,3 and 4 on

Error = Switching_SetOutputs ( BoardNumber, Outputs)
```

Function Name: Switching_GetOutputs
Applicable to: Switching Hawk
Syntax 'C/C++': int Switching_GetOutputs (int BoardNumber, int *Outputs)
Syntax 'Basic/VB': Switching_ GetOutputs (ByVal BoardNumber As Integer,
ByRef Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x0fffffff (Hex) corresponding to the on/off pattern of switching outputs. Only the first 28 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit27 to output 28. When the appropriate bit is a '1' the output is on, when it is a '0' the output is off.

Description: Sends a command to the Switching Hawk specified by BoardNumber to get the current on/off settings of the switching outputs.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;

BoardNumber = 2;                // may be any valid board number of those available

Error = Switching_GetOutputs ( BoardNumber, &Outputs);

if(Error == 0)
{
…. Outputs now contains the current on/off pattern of the switching outputs
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer
BoardNumber = 1                 ' may be any valid board number of those available

Error = Switching_GetOutputs ( BoardNumber, Outputs)
If Error = 0 Then
…. Outputs now contains the current on/off pattern of the switching outputs
Else
  process the error
Endif
```

Function Name: Servo_SetServos
Applicable to: Servo Hawk
Syntax 'C/C++':

```
int Servo_SetServos (int BoardNumber,
                     int Servo1,
                     int Servo2,
                     int Servo3,
                     int Servo4,
                     int Servo5,
                     int Servo6,
                     int Servo7,
                     int Servo8      );
```

Syntax 'Basic/VB':

```
Servo_SetServos (    ByVal BoardNumber As Integer,
                     ByVal Servo1 As Integer,
                     ByVal Servo2 As Integer,
                     ByVal Servo3 As Integer,
                     ByVal Servo4 As Integer,
                     ByVal Servo5 As Integer,
                     ByVal Servo6 As Integer,
                     ByVal Servo7 As Integer,
                     ByVal Servo8 As Integer        ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards currently connected.

Servo1 – Servo8 is the position of the servo between min and max positions. A figure of 0 will cause the servo to take up the minimum position (usually fully anti-clockwise) and 1024 will cause it to take up to max position (usually fully clockwise) with all other intermediate values corresponding to relative intermediate positions.

Description: This function sets the positions of all 8 servos. The servo position parameter (0 – 1024) corresponds to a timing signal in the range 1.0ms to 2.0ms. This , by convention, sets the servo position to a relative position within its specified range of movement between its min and max positions. The angular position for a given figure will depend on the characteristics of the servo being used. Some servos only operate from -90 degrees to +90 degrees. In that case a figure of 0 would correspond to the -90 and the 1024 to +90. A figure of 512 will always centralise the servo (in the example case at 0 degrees).

Usage "C / C++":

```
int BoardNumber;
int Error, Servo1, Servo2, Servo3, Servo4, Servo5, Servo6, Servo7, Servo8;

BoardNumber = 1;

Servo1= 0;          // min position
Servo2 = 100;       // intermediate position
Servo3 = 300;       // intermediate position
Servo4 = 500;       // intermediate position
Servo5 = 600;       // intermediate position
Servo6 = 700;       // intermediate position
Servo7 = 900;       // intermediate position
Servo8 = 1024;      // max position
```

Error = Servo_SetServos (BoardNumber, Servo1, Servo2, Servo3, Servo4, Servo5, Servo6, Servo7, Servo8);

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Servo1 As Integer
Dim Servo2 As Integer
Dim Servo3 As Integer
Dim Servo4 As Integer
Dim Servo5 As Integer
Dim Servo6 As Integer
Dim Servo7 As Integer
Dim Servo8 As Integer
Dim Error As Integer

BoardNumber = 5
Servo1= 0              // min position
Servo2 = 150           // intermediate position
Servo3 = 350           // intermediate position
Servo4 = 530           // intermediate position
Servo5 = 670           // intermediate position
Servo6 = 760           // intermediate position
Servo7 = 930           // intermediate position
Servo8 = 1024          // max position
```

Error = Servo_SetServos (BoardNumber, Servo1, Servo2, Servo3, Servo4, Servo5, Servo6, Servo7, Servo8)

Function Name: Servo_GetServos
Applicable to: Servo Hawk
Syntax 'C/C++':

```
int Servo_GetServos (int BoardNumber,
                     int *Servo1,
                     int *Servo2,
                     int *Servo3,
                     int *Servo4,
                     int *Servo5,
                     int *Servo6,
                     int *Servo7,
                     int *Servo8      );
```

Syntax 'Basic/VB':

```
Servo_GetServos (    ByVal BoardNumber As Integer,
                     ByRef Servo1 As Integer,
                     ByRef Servo2 As Integer,
                     ByRef Servo3 As Integer,
                     ByRef Servo4 As Integer,
                     ByRef Servo5 As Integer,
                     ByRef Servo6 As Integer,
                     ByRef Servo7 As Integer,
                     ByRef Servo8 As Integer        ) As Integer
```

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards currently connected.

Servo1 – Servo8 is the position of the servo between min and max positions. A figure of 0 will indicate that the servo is at the minimum position (usually fully anti-clockwise) and 1024 will indicate max position (usually fully clockwise) with all other intermediate values corresponding, in proportion, to intermediate positions.

Description: This function gets the positions of all 8 servos. The servo position parameter (0 – 1024) corresponds to a timing signal in the range 1.0ms to 2.0ms. This , by convention, corresponds to a relative position within its specified range of movement between its min and max positions. The angular position for a given figure will depend on the characteristics of the servo being used. Some servos only operate from 0 degrees to +180 degrees. In that case a figure of 0 would correspond to 0 degrees and the 1024 to +180. A figure of 512 always indicates that the servo is centralised (in the example it would indicate 90 degrees).

Usage "C / C++":

```
int BoardNumber;
int Error, Servo1, Servo2, Servo3, Servo4, Servo5, Servo6, Servo7, Servo8;

BoardNumber = 1;

Error = Servo_GetServos (BoardNumber, &Servo1, &Servo2, &Servo3, &Servo4, &Servo5, &Servo6,
&Servo7, &Servo8);

if(Error == 0)
{
   Servo1 to Servo8 will now contain valid
  indications of the positions of the servos
}
else
```

```
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Servo1 As Integer
Dim Servo2 As Integer
Dim Servo3 As Integer
Dim Servo4 As Integer
Dim Servo5 As Integer
Dim Servo6 As Integer
Dim Servo7 As Integer
Dim Servo8 As Integer
Dim Error As Integer

BoardNumber = 5

Error = Servo_GetServo (BoardNumber, Servo1, Servo2, Servo3, Servo4, Servo5, Servo6, Servo7, Servo8)

If Error = 0 Then
    Servo1 to Servo8 will now contain valid
   indications of the positions of the servos
Else
  process the error
Endif
```

Function Name: Servo_SetOutputs
Applicable to: Servo Hawk
Syntax 'C/C++': int Servo_SetOutputs (int BoardNumber, int Outputs)
Syntax 'Basic/VB': Servo_ SetOutputs (ByVal BoardNumber As Integer,
ByVal Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x07f (Hex) corresponding to the on/off pattern of switching outputs. Only the first 7 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit6 to output 7. When the appropriate bit is a '1' the output is on, when it is a '0' the output is off.

Description: Sends a command to the Servo Hawk specified by BoardNumber to set the switching outputs to the on/off pattern specified.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;

BoardNumber = 2;          // may be any valid board number of those available

Outputs  = 0x0F;          // outputs 1,2,3 and 4 on

Error = Servo_SetOutputs ( BoardNumber, Outputs);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer

BoardNumber = 1           ' may be any valid board number of those available

Outputs = 15              'outputs 1,2,3 and 4 on

Error = Servo_SetOutputs ( BoardNumber, Outputs)
```

Function Name: Servo_GetOutputs
Applicable to: Servo Hawk
Syntax 'C/C++': int Servo_GetOutputs (int BoardNumber, int *Outputs)
Syntax 'Basic/VB': Servo_ GetOutputs (ByVal BoardNumber As Integer,
ByRef Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x07f (Hex) corresponding to the on/off pattern of switching outputs. Only the first 7 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit6 to output 7. When the appropriate bit is a '1' the output is on, when it is a '0' the output is off.

Description: Sends a command to the Servo Hawk specified by BoardNumber to get the current on/off settings of the switching outputs.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;
BoardNumber = 2;                // may be any valid board number of those available

Error = Servo_GetOutputs ( BoardNumber, &Outputs);

if(Error == 0)
{
…. Outputs now contains the current on/off pattern of the switching outputs
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer
BoardNumber = 1                 ' may be any valid board number of those available

Error = Servo_GetOutputs ( BoardNumber, Outputs)
If Error = 0 Then
…. Outputs now contains the current on/off pattern of the switching outputs
Else
  process the error
Endif
```

Function Name: Diy_SetOutputs
Applicable to: DIY Hawk
Syntax 'C/C++': int Diy_SetOutputs (int BoardNumber, int Outputs)
Syntax 'Basic/VB': Diy_SetOutputs (ByVal BoardNumber As Integer,
ByVal Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x00001FFF (Hex) corresponding to the on/off pattern of digital outputs. Only the first 13 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit12 to output 13. When the appropriate bit is a '1' the output is on (logic 1 / high / +5v), when it is a '0' the output is off (logic 0 / low / 0v).

Description: Sends a command to the DIY Hawk specified by BoardNumber to set the digital outputs to the on/off pattern specified.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;

BoardNumber = 2;          // may be any valid board number of those available

Outputs  = 0x0F;          // any number in the range 0 – 8191  (0 – 0x1FFF Hex)

Error = Diy_SetOutputs ( BoardNumber, Outputs);
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer

BoardNumber = 1           ' may be any valid board number of those available

Outputs = 15              'any number in the range 0 – 8191  (0 – 0x1FFF Hex)

Error = Diy_SetOutputs ( BoardNumber, Outputs)
```

Function Name: Diy_GetOutputs
Applicable to: DIY Hawk
Syntax 'C/C++': int Diy_GetOutputs (int BoardNumber, int *Outputs)
Syntax 'Basic/VB': Diy _ GetOutputs (ByVal BoardNumber As Integer,
ByRef Outputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Outputs A positive 32 bit integer with a value in the range 0 to 0x1FFF (Hex) corresponding to the on/off pattern of digital outputs. Only the first 13 bits of the Outputs parameter are used. Bit 0 corresponds to output1, Bit1 to output 2 …. Bit12 to output 13. When the appropriate bit is a '1' the output is on, when it is a '0' the output is off.

Description: Sends a command to the DIY Hawk specified by BoardNumber to get the current on/off settings of the digital outputs.

Usage "C / C++":

```
int BoardNumber;
int Outputs;
int Error;

BoardNumber = 2;                // may be any valid board number of those available

Error = Diy _GetOutputs ( BoardNumber, &Outputs);

if(Error == 0)
{
…. Outputs now contains the current on/off pattern of the digital outputs
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Outputs As Integer
Dim Error As Integer
BoardNumber = 1                 ' may be any valid board number of those available

Error = Diy _GetOutputs ( BoardNumber, Outputs)
If Error = 0 Then
…. Outputs now contains the current on/off pattern of the digital outputs
Else
  process the error
Endif
```

Function Name: Diy_GetDigitalInputs
Applicable to: DIY Hawk
Syntax 'C/C++': int Diy_GetDigitalInputs (int BoardNumber, int *Inputs)
Syntax 'Basic/VB': Diy _ GetDigitalInputs (ByVal BoardNumber As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.

Inputs A positive 32 bit integer with a value in the range 0 to 0x1FFF (Hex) corresponding to the on/off pattern of digital inputs. Only the first 13 bits of the Inputs parameter are used. Bit 0 corresponds to input 1, Bit1 to input 2 …. Bit12 to input 13. When the appropriate bit is a '1' the input is "ON" (logic 1 / high / +5v), when it is a '0' the input is "OFF" (logic 0 / low / 0v).

Description: Sends a command to the DIY Hawk specified by BoardNumber to get the current on/off settings of the digital inputs.

Usage "C / C++":

```
int BoardNumber;
int Inputs;
int Error;

BoardNumber = 2;                // may be any valid board number of those available

Error = Diy _ GetDigitalInputs ( BoardNumber, &Inputs);

if(Error == 0)
{
.... Inputs now contains the current on/off pattern of the digital inputs
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Inputs As Integer
Dim Error As Integer
BoardNumber = 1                 ' may be any valid board number of those available

Error = Diy _ GetDigitalInputs ( BoardNumber, Inputs)
If Error = 0 Then
.... Inputs now contains the current on/off pattern of the digital inputs
Else
  process the error
Endif
```

Function Name: Diy_GetAnalogueInputs
Applicable to: DIY Hawk
Syntax 'C/C++': int Diy_GetAnalogueInputs (int BoardNumber, int *Inputs)
Syntax 'Basic/VB': Diy _ GetAnalogueInputs (ByVal BoardNumber As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or '1' if failed.

Parameters: BoardNumber An integer corresponding to a board number within the range of the number of boards, of this type, currently connected.
Inputs An array of 4 positive 32 bit integers, each with a value in the range 0 to 1023 (0 - 0x3FF Hex) corresponding to the digitised values of the analogue inputs. Note: The "inputs" pointer supplied should point to an array of 4 integers since this function will return 4 integers starting at the pointer address.
Description: Sends a command to the DIY Hawk specified by BoardNumber to get the current digitised values of the analogue inputs.

Usage "C / C++":

```
int BoardNumber;
int Inputs[4];
int Error;

BoardNumber = 2;                // may be any valid board number of those available

Error = Diy _ GetAnalogueInputs ( BoardNumber, Inputs);

if(Error == 0)
{
....Inputs[0] now contains the current digitised value of analogue  input 1
....Inputs[1] now contains the current digitised value of analogue  input 2
....Inputs[2] now contains the current digitised value of analogue  input 3
....Inputs[3] now contains the current digitised value of analogue  input 4
}
else
{
  process the error
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
Dim Inputs(4) As Integer
Dim Error As Integer
BoardNumber = 1                 ' may be any valid board number of those available

Error = Diy _ GetAnalogueInputs ( BoardNumber, Inputs(0) )
If Error = 0 Then
....Inputs(0) now contains the current digitised value of analogue  input 1
....Inputs(1) now contains the current digitised value of analogue  input 2
....Inputs(2) now contains the current digitised value of analogue  input 3
....Inputs(3) now contains the current digitised value of analogue  input 4
Else
  process the error
Endif
```

10. Minimum PC System Requirements

Motor Hawk, Switching Hawk and Hawkeye software do not require a high spec PC for correct operation, but the following system is suggested as a sensible minimum

Processor	500MHz Pentium
Memory	64MB
HDD	10MB free space required
Screen Resolution	1024x768 (256 colours)
Interface	One free USB socket (1.0 or 2.0)
Operating System	Windows XP , Win 7 and Win 8

WARNING: All Hawk adaptor boards are intended for DC voltages less than 50v. They should not be connected directly to mains voltages under any circumstances.

Hawk Products
Regulatory Compliance and Safety Information

Product Name:
Hawk Product Range

Part Nos.
BRD020, BRD021, BRD022


IMPORTANT PLEASE RETAIN THIS INFORMATION FOR FUTURE REFERENCE

Warnings

- These products should not be connected to mains voltages under any circumstances.
- These products should be placed on a stable, flat, non-conductive surface in use and should not be contacted by conductive items.
- The connection of non CE compliant devices may affect overall compliance or result in damage to the unit and invalidate the warranty.

Instructions for safe use

- To avoid malfunction or damage to your board please observe the following:
- Do not expose it to water, moisture or place on a conductive surface whilst in operation.
- Do not expose it to heat from any source; these boards are designed for reliable operation at normal ambient room temperatures.
- Take care whilst handling to avoid mechanical or electrical damage to the printed circuit board and connectors.
- Avoid handling the board while it is powered. Only handle by the edges to minimize the risk of electrostatic discharge damage.
- All peripherals used with the boards should comply with relevant standards for the country of use and be marked accordingly to ensure that safety and performance requirements are met.

Compliance Information

- The boards comply with the relevant provisions of the RoHS Directive for the European Union.

WEEE Directive Statement for the European Union

- In common with all Electronic and Electrical products the boards should not be disposed of in household waste. Alternative arrangements may apply in other jurisdictions.

EMC Compliance Statements

European Union (EU) Electromagnetic Compatibility Directive Compliance Statement

All boards within this product range conform with the protection requirements of EU Council Directive 2004/108/EC on the approximation of the laws of the Member States relating to electromagnetic compatibility.
Warning: This is equivalent to an EN 55022 Class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

PC Control Ltd. 18 Beech Close, Desborough, Kettering, Northants NN14 2XQ, UK
www.pc-control.co.uk

Terms of Use for all Goods Supplied

Definitions

'Supplier' shall mean PC Control Ltd.

'Buyer' shall mean the person, company or any other body that purchases or agrees to purchase Goods.

'Goods' shall mean all goods and services which the Buyer agrees to buy from the Supplier including replacements for defective Goods, hardware, documentation and software products licensed for use by the Buyer.

Use of the Goods in any way by the Buyer constitutes acceptance of these terms and conditions.

Terms and Conditions

1. The Goods are intended to be part of the buyer's own design of apparatus and not a finished product in their own right.
2. The Goods supplied are not to be used in any design where there is a risk, however small, either directly or indirectly, of death or personal injury.
3. The Buyer will be responsible for ensuring the fitness for purpose of the Goods for the Buyer's application.
4. To the extent permitted by law, the Supplier accepts no liability whatsoever or howsoever arising in respect of loss, damage or expense arising from errors in information or advice provided whether or not due to the Supplier's negligence or that of its employees, agents or sub-contractors save for any loss or damage arising from death or personal injury.
5. To the extent permitted by law, the Supplier shall not be liable to the Buyer by reason of any representation (unless fraudulent), or any implied warranty, condition or other term, or any duty at common law, or under the express terms of any Contract with the Buyer, for any indirect, special or unforeseen loss or damage (whether for loss of profit or otherwise), costs, expenses or other claims for compensation whatsoever (whether caused by the negligence of the Supplier, its employees or agents or otherwise) which arise out of or in connection with the supply of the Goods or their use or resale by the Buyer.
6. The entire liability of the Supplier under or in connection with the Contract with the Buyer shall not exceed the price of the Goods except as expressly provided in these terms and conditions.
7. These terms are an important part of the full terms and conditions of business as published on the website at www.pc-control.co.uk/general-terms.htm which also apply.

If you cannot agree to the terms and conditions of use of the boards(s) from the Hawk range then you should return it to the supplier within 7 days of receipt to receive a refund. Your use of the board or the associated software in any way whatsoever will be regarded as an acceptance of these terms and conditions.