CAIRN OPTOSPIN IV INSTRUCTION MANUAL

Updated January 2019, for PIC firmware versions 4.3 onwards

INTRODUCTION

The Cairn Optospin IV acquired its name for two reasons. It is our fourth filter wheel design, and up to four filter wheels can be operated by a single controller. These wheels can spin continuously in perfect synchrony, under the control of an internally programmed or externally supplied reference frequency, or they can be simultaneously or independently stepped between specified filter positions. One may think that a product capable of operating in two different ways would be beset by various compromises, but in fact the design requirements are surprisingly similar, and this has enabled us to achieve exemplary performance in both modes. The underlying physics doesn't seem to be very well known, but it's actually very straighforward, so we include discussion of it in the Appendix. Actually the Optospin IV design bears more than a passing resemblance to the one that we introduced in the 1990s, but the readier availability of better magnetic materials since those days has resulted in the development of more powerful electric motors suitable for this application - a fact that we have taken full use of!

In order to extract the best possible performance from these motors, their operation is controlled by a type of processor called a PIC, which stands for Programmable Interrupt Controller. There is a whole family of these devices, and we have needed to use one of the more powerful ones (specifically an 18F4620 for those of you who may care about such things!) in order to obtain the performance levels we require. The control software is programmed into the device itself, and hence is actually referred to as “firmware”, but the devices can readily be reprogrammed. This manual is for firmware version 4.2 onwards, but if by any chance you have an earlier version, then a quick reprogram (which you might even be able to do yourself, see the list of USB commands) should be all you need to support the new features for this version.

Each wheel has six filter positions, and is enclosed within a very compact housing, of dimensions 100x100x35mm. A very powerful design feature is that two housings can be connected together in an interlocking manner, putting the two wheels in series while maintaining the same overall thickness (i.e. optical path length) of 35mm. This feature has at least three major uses. First, one of the wheels can be used for auxiliary filters, such as neutral density ones. Second, if one filter position in each wheel is left open, then any of ten filters can now be selected; the controller allows this configuration to appear as a single ten-position wheel. We term this form of operation “combined mode” This combination can step between filter positions very much faster than the equivalent single wheel ever could, both because there are fewer filter positions to traverse in the worst case (three compared with five), and because the inevitably larger size of a ten-position wheel gives it a substantially higher inertia - see the physics primer in the Appendix for further information. Third, a stepping

filter wheel generates a countertorque, which can cause vibration. If that is likely to be a problem, it can be cancelled by using the second wheel to generate an opposite countertorque. Furthermore, spreading six filters over two wheels, e.g. so that odd numbered positions are used in one wheel and even numbers in the other, reduces the moving mass per wheel, and hence further reduces the stepping time. Where both wheels go to the same numeric position, we term this form of operation "slave mode", but they can also be driven to different positions of course.

The Optospin IV uses standard 25mm diameter filters of up to 6mm thickness. They can be accessed for inspection or changing via the removeable upper section of the housing. As an additional feature, each housing contains some EEPROM memory, which the controller can use to read and write information about the filters currently installed, as well as for remembering various configuration parameters.

Our original filter wheel was designed for photometry on a millisecond timescale. It could spin continuously at speeds of up to at least 100Hz in order to change filter positions sufficiently rapidly for this application, and its successors retained this ability. In spite of the subsequent emergence of digital imaging cameras as a viable alternative for some applications, our modular photometry system remains in demand, so the Optospin IV provides the necessary control signals for these modules as standard. In addition, it has both a USB and a TTL (parallel digital) interface for interconnection with and possible control by other equipment. The module interface is also of use for other purposes, so it is also described here. Note that many functions can be controlled by either the TTL or the USB interfaces, and when there is a possible conflict, it is generally the USB commands that take precedence; in fact. there is one that causes the TTL interface to be ignored altogether. However, for time-critical commands, use of the TTL interface is preferred, because the USB communication standard incorporates unpredictable time delays.

Note also that the Optospin IV has two distinct operating modes. In spin mode, any combination of filter wheels can be spun, while the others can remain stationary at specified positions. In step mode, any combination of wheels can be stepped to any positions simultaneously. The only things that a single controller cannot do is to have different wheels spinning at different speeds, or to have some wheels stepping while others are spinning. But to support those applications you would just need another controller, and the interfacing facilities that we shall now describe would make it a relatively straightforward task to operate two or more controllers together.

For precise positional control, a combination of magnetic and optical sensors is used. The magnetic sensors give absolute positional information at points that correspond to the actual filter positions, so the system always "knows" whether the correct filter is in the light path, and can automatically correct accordingly if anything has gone wrong (although that is likely to be caused by some external event than an internal error). For the finer positional information that is required during a step, a pair of optical sensors is used. These use infrared illumination, which will normally be of no

consequence. However, where very low light levels may be measured in conjunction with particularly sensitive cameras, there is a potential interference risk, so we avoid this by illuminating the sensors only when that filter wheel is actually in the process of stepping. Data acquisition can then take place under conditions of guaranteed darkness.

I**NTERFACING**

TTL INTERFACE

This interface is via a 25 way male D connector. By convention on this type of connector, pins 1-13 form the top (longer) row, and pins 14-25 form the bottom row, with the numbering running from left to right in both cases. Unfortunately, this is not the most logical scheme when it comes to wiring up these connectors (especially to a ribbon cable), for which the logical sequence is 1, 14, 2, 15, etc. Therefore we have gone for the logical sequence and have listed the pin functions in that same order.

CONNECTOR PIN	FUNCTION	ALTERNATE FUNCTION
1	Wheel 1 bit0 in	“Shutter” input for data acquisition
14	Wheel 1 bit1 in	
2	Wheel 1 bit2 in	
15	Wheel 2 bit0 in	
3	Wheel 2 bit1 in	
16	Wheel 2 bit2 in	
4	Spin in	
17	Extref in	Step Command in
5	Read wheels 3 and 4 out if high	
18	Wheel 3 bit0 in	
6	Wheel 3 bit1 in	
19	Wheel 3 bit2 in	
7	Wheel 4 bit0 in	
20	Wheel 4 bit1 in	
8	Wheel 4 bit2 in	
21	Wheel 1 (3) bit0 out	
9	Wheel 1 (3) bit1 out	
22	Wheel 1 (3) bit2 out	
10	Wheel 2 (4) bit0 out	
23	Wheel 2 (4) bit1 out	High during filter 1 when spinning
11	Wheel 2 (4) bit2 out	
24	High if spinning	High if ready to step again
12	Sync out	“Ready” (high when step finished)
25	Ground	
13	+5V out	

The “Spin” input on pin 4 determines whether the controller is in spin (logic high) or step (logic low) mode. Like all the other inputs, this defaults to low (i.e. step mode in this case) if nothing is connected to it.

The various “bit” inputs are used in step mode only (apart from the first one), in conjunction with the “Step Command” input on pin 17. In normal use, the required position for each wheel is specified by the status of three inputs, to give a binary encoded filter position, where bit0 is the least significant one. Filter positions 1 to 6 are encoded by bit values ranging from 001 to 110 respectively. When the “Step Command” input goes high, any wheel that is not currently in the specified position will step to it. A bit value of 000 (where logic 0 is the default state of any unconnected input) is interpreted as “stay at current filter position”. A bit value of 111 also encodes filter position 6, and is used by our remote controller to signal that (in conjunction with the bit pattern sent to the other wheel) that two wheels are being used together in combined mode. Further information on this and other points is given in the section that describes the stepping mode.

Current filter positions can be read from the corresponding “bit” outputs, using the same formats. To reduce the number of digital connections, a toggling input has been provided on pin 5. When low, the positions of wheels 1 and 2 can be read, and when high, the positions of wheels 3 and 4. Of particular potential use is the “Ready” output on pin 12. This goes low while the wheels are actively stepping, and high once they are all in their specified positions. It is supplemented by an additional output on pin 24. A variable delay (set via USB) can be programmed, to set a minimum further time before the system will respond to further step commands, and this output will go high once the system is ready to step again. Setting such a delay will prevent users from attempting to set unrealistically short dwell times for any filter position. However, the stepping code enhancements introduced in version 4.2 of the PIC firmware mean that you now *can* have dwell times of just a few tens of milliseconds if you really want to. In our opinion this type of requirement is better handled by spinning the wheels continuously, the control interfacing for which is comprehensively supported in firmware version 4.0 onwards, but as they say, the customer is always right!

TTL interfacing is rather more straightforward in spin mode (pin 4 high), since many functions are (necessarily) controlled via the USB. The main choice to make here is whether the spin speed is determined within the controller via the relevant USB commands, or whether it is determined by an external reference frequency. In the external case, the reference is connected to pin 17, and the wheel(s)' rotation will normally synchronise so that at the rising edge of the reference, they are midway between the positions for filter 6 and filter 1. However, as will be described in the module interface section, synchronisation can instead be with respect to successive filter positions or multiples thereof, so there if considerable flexibility here. By default, the wheels will synchronise to an external reference if one is provided, but

this preference can be overriden by (yet) another USB command. Whatever the reference source, the wheels will progressively accelerate or decelerate to the requested frequency, and when they are synchronised to it (in phase as well as frequency), then the "Sync" output on pin 12 will go high.

In spin mode, the filter position will of course be continuously changing, so in this case it is much more useful to know which filter is currently in the light path. Since all the wheels that are spinning will be precisely in phase with each other, we only need one set of outputs, which appear on the bits 0-2 outputs for wheel 1, i.e. pins 21, 9 and 22. For synchronising other equipment to the wheel(s), a reference output is also provided on pin 23, which is present once per revolution, centred on filter position 1. (Note that outputs for all filter positions are available on the module interface connector). In all cases the output changes when the wheels are midway between filter positions. For PIC firmware versions 4.0 onwards, the module interface, described below, provides further outputs dedicated to each individual filter position. These are of programmable duration, and are potentially very useful for applications such as synchronising light sources to the individual fitlers in the wheel(s), as described in more detail there.

Finally, a low-power +5V output is provided on pin 25, along with a ground connection on pin 13. The +5V output is limited by a 100 ohm series resistor, as it is not intended to provide significant power to other equipment. However, it is sufficient to power our remote controller, described next, and it also provides a logic high level reference, to allow the functionality of the TTL inputs to be easily tested directly if need be.

REMOTE CONTROLLER

For manual control of the filter wheel(s) a remote controller is available. It communicates directly with the TTL interface, but still allows any other equipment to be connected in parallel with it, *provided* that the same "wired-or pullup" output format is used. What this means is that all the inputs to the TTL interface default to a logic low level if not connected, and all the outputs from the remote controller (and hopefully from other equipment too!) are effectively in a high-impedance state (actually just presenting a further, parallel, resistance to ground) when signalling that a logic low level is being sent. To signal a logic high level on any output, the remote controller does so by providing an appropriate "pull-up" current. Since the inputs can all be low for most of the time, this allows either the remote controller or any other equipment to take control when needed. If the other equipment has low-impedance outputs, it will take control away from the remote controller, but without causing any damage to it. If this potential conflict turns out to be a significant issue, we could quite easily design a buffer unit that would give the third-party equipment the same interface, although it would still be important for that equipment to keep all its own outputs at a logic low level when in the "inactive" state.

We have found the remote controller to be a particularly nice piece of equipment to use. To be able to have essentially full control over the Optospin without the usual bank of computers (and no, we don't have a mobile phone app either!) has proved to be a most refreshing experience. Detailed descriptions of the facilities provided by the remote controller are given in the appropriate sections elsewhere in this manual, but the following basic points can be noted here.

Although the remote controller is primarily intended for step mode control, it also supports spin mode, albeit mainly for demonstration purposes. It does so by incorporating a variable frequency generator, to which the filter wheel(s) can be synchronised. In step mode, full control of two filter wheels is provided via pushbuttons which illuminate to indicate the current filter position(s). In a four-wheel system, control is selectable between wheels 1 and 2, or 3 and 4, or with wheels 3 and 4 following the same commands as for wheels 1 and 2. For the currently selected pair(s) of wheels, they can be controlled individually, or as a combined pair to give (and select as if there are) effectively ten filter positions, or as a slaved pair so that both wheels are driven to the same position. These modes are all selected by switches on the remote controller, so in practice just about all feasible control combinations even for four filter wheels are supported.

Two indicator LEDs display the basic status information – a green one to show when the filter wheels are spinning at the set speed or have completed a step, and a red one to show when the filter wheels are all not spinning and are ready to accept a step command. This distinction is explained in more detail in the description of the stepping mode.

MODULE INTERFACE

The module interface is provided on a 25 way female D connector, and provides outputs only. Unlisted pins aren't connected to anything. This interface was originally intended to provide the necessary signals for using the Optospin in conjunction with our modular photometry system, and it retains that potential function. However, it includes eight digital outputs, which are actually general-purpose in design terms, and we have taken advantage of that in two other ways (so far – further suggestions are always welcome!) First, it also doubles as an interface to our diagnostic card, the uses of which are described in subsequent sections. Second, for PIC firmware versions 4.0 and above, it can provide programmable-duration signals linked to the individual filter positions, as described in the rest of this section.

But first, a description of the connector pinout. The pin numbering is even more confusing than for the TTL interface, because it is a female one, whereas the TTL interface is a male! This means that, by convention, whereas the male connector's outputs are numbered sequentially from left to right (top row, then bottom row), the female connector's outputs number from right to left! In a probably vain attempt to

make some sort of sense out of this situation, the conventional pin numbering is again shown here, but in the same physical sequence as for the TTL one, namely corresponding to top left, bottom left, second left, and so on, to again give the most logical sequence when making the physical connections via a ribbon cable.

CONNECTOR PIN	FUNCTION	
1	Ground	
14	+5V	
2	N/C	
15	N/C	
3	Ready	
16	Reset	
4	Read	
17	All Filters when spinning (always)	(“Wavelength 8”)
5	All Filters when spinning (if “shutter” open)	(“Wavelength 7”)
18	Filter wavelength 6	
6	Filter wavelength 5	
19	Filter wavelength 4	
7	Filter wavelength 3	
20	Filter wavelength 2	
8	Filter wavelength 1	
21	Stopped	
9	Ground	
22	N/C	
10	N/C	
23	N/C	
11	N/C	
24	N/C	
12	N/C	
25	N/C	
13	N/C	

Note that behaviour of the “wavelength 7” and “wavelength 8” outputs is somewhat different in step mode. See the section below describing this mode for further information.

From PIC firmware version 4.0 onwards, there are two principal operating modes for the module interface, which are “photometry” (the original one) and “imaging” (new for version 4.0) respectively. Version 4.0 also supports enhanced photometry operation, as described below, but first the “standard” photometry mode will be described.

In this mode, and in conjunction with our photometry modules, the photometric signal from each filter position is integrated in turn, for as long as the “Ready” output

signal on pin 3 is high, which by default is almost continuously so. Between filter positions, this signal temporarily goes low (for about 50 microseconds), during which time two other things happen. First, the "Read" signal on pin 4 goes high for about 20 microseconds. This causes the integrated output signal from the input amplifier to be copied to the appropriate sample-and-hold amplifier in the output module. The sample-and-hold amplifier is selected according to which of the six filter wavelength outputs is high at the time; each output is high for exactly one-sixth of a wheel revolution, so one of them will be high at any given time. After that, the reset signal goes high for about 20 microseconds. This causes the integrator in the input amplifier module to be discharged back to zero, in preparation for acquiring the next filter signal.

A possible issue is that when the wheel is between two filter positions, some light may be able to go through each one. Our default way of dealing with this, for both photometry and imaging, has been to ensure that the diameter of the light beam as it goes through the filter wheel isn't great enough for this to happen. However, firmware version 4.0 onwards supports a much better solution, which is to modulate the light source(s) so that illumination only occurs during a (programmable) part of each filter position. Furthermore, with multiple light sources, different illumination wavelengths can be synchronised to different filter positions, which can drastically reduce crosstalk in multifluorophore applications. This form of operation is best described with reference to the new "imaging" mode, the description of which follows.

In imaging mode, which is likely to be the more common application, it's assumed that the filter wavelength outputs aren't needed for control of our photometry modules, and this allows them to be taken over for other uses, especially for control of illumination as just mentioned. Correct operation of photometry mode requires the individual wavelength outputs to remain high throughout the corresponding filter position, so that one of these is high at any time. In imaging mode (no photometry modules present), that's no longer essential, so individual outputs can in principle be taken high or low for arbitrary fractions of a given filter position. Amongst other potential uses, this allows a (solid-state) light source to be switched on only during periods when a given filter is fully within the light path, in order to avoid crosstalk and possible image shadowing from "filter edge" effects. At least some light sources are now bright enough to to be correspondingly overdriven to make full use of the consequently shortened sampling windows, while still not being so bright as to risk the possible second-order bleaching effects that the much higher intensities and shorter duty cycles associated with confocal scanning (for example) may cause.

Under USB control in imaging mode, the duration of each filter position (defined as one-sixth of a wheel rotation) can be divided into sevenths, centred around the midpoint of each filter position. The associated wavelength output can be programmed to be anywhere between one and all sevenths of this overall period. The fraction is the same for all filter positions, so if programmed to be on for all sevenths,

the output for each wavelength is the same as in photometry mode.

In photometry mode, as previously noted, each wavelength output must be high for the entire duration of a filter position, so the light source gating possiblity isn't directly available here, but there is nevertheless an easy workaround. Our photometry module architecture supports up to eight wavelengths, but there are only six positions in our filterwheels, so the outputs for wavelengths 7 and 8 are potentially available for other purposes here. Therefore, and actually in imaging mode as well, the wavelength 8 output goes high for *all* filter positions during the fractional period (in sevenths) for which the individual wavelength outputs have been programmed to be high. Therefore an equivalent behaviour for light source modulation in photometry mode can be obtained just by logically ANDing the wavelength 8 output with each of the others. Of course, for many applications, both photometry and imaging, this "global" output may be all that is needed to control the light source. However, when multiple light sources are used for different excitation wavelengths in multifluorophore applications, the ability to control individual ones according to which filters are currently in the light path can greatly reduce the otherwise inevitable crosstalk between the individual excitation and emission wavelengths. This possibility is *very* well worth exploiting!

To assist in synchronisation with third-party imaging software (although it also works in photometry mode), a further output is available in the "wavelength 7" position. This is potentially the same as for "wavelength 8", but its presence depends on that of a "shutter" signal on pin 1 of the TTL interface. The idea is that an output signal that goes high for every filter position can be used by imaging software to acquire successive images, but the "wavelength 8" signal does not directly indicate *which* filter is present. Although this information is available on other outputs, it may not be so easy to communicate it to the imaging software, so this is where the "wavelength 7" output comes in. It is present only when the "shutter" input on TTL interface pin 1 is present, but the first time it appears it will *always* be for the filter 1 position, so the imaging software can keep track of things from then onwards. And just in case it's a bit slow on the uptake here, a USB command allows a delay of a variable number of wheel revolutions between the "shutter" going high and this signal first appearing. When the "shutter" goes low again, this output always switches off at the end of the current rotation.

New for firmware version 4.3 onwards, the "wavelength 7" output has been given some further functionality. Instead of being a potential copy of the "wavelength 8" one, it now goes high for just the first one-seventh of each filter position, but it remains gated by the "shutter" signal as described above. Its intended function is to trigger the acquisition of the next camera image, which can safely occur at the start of each filter position as the light sources are all likely to be switched off at this time (by the wavelength 1-6 outputs all being low). This potentially gives additional time for the camera to send the previous image to the acquisition software, which the "rolling shutter" type specifically requires.

Note that there is a potential hardware conflict in that the “shutter” input is also used as an input for direct digital control of the filter positions in step mode. Also, some users may not want to use this input for shuttering anyway, so in version 4.3 there is now a USB command (SET_SHUTTER) that can make the system function as if this input is permanently high, regardless of its actual status, thereby freeing it up exclusively for digital position control. If this hardware conflict is nevertheless a problem for anyone, various workarounds are possible here, so please contact us for further information.

In both photometry and imaging mode, firmware version 4.0 onwards supports some further potentially useful possibilities. One doesn't always need six wavelengths – perhaps only two or three will suffice. The facility to treat sequential pairs or triples of filters as if they were a single filter of the same wavelength has therefore now been supported, again under USB control. Control of the output duration for each wavelength is the same as before, but the second filter in a sequential pair, or the second and third filters in a sequential triple, put their control signals on the same wavelength output as the first. So in the paired situation, for example, wavelength output 1 goes high for the programmed fraction of filter position 1, and high again for the programmed fraction of filter position 2. The next pair would sequentially output on wavelength output 2, and so on. For version 4.3 onwards, the “wavelength 7” output goes high for the first one-seventh of only the first filter of a pair or triple when these modes are selected, so when used to trigger a camera it will ensure that each pair or triple is acquired as a single image.

For photometry applications, the control signals that are sent to the modules enforce filter positions paired and tripled in this way to be treated as single ones, so nothing else needs to be done here, but for imaging applications there are also some useful further control possibilities. In most imaging applications, some form of master timer is required to co-ordinate the wheel speed with the camera frame rate, and sometimes it might be more convenient for the camera and its software to set the frame rate, and then to synchronise the wheel(s) to this. For firmware versions prior to 4.0, the synchronisation rate was always to the rotation speed, but this can now also correspond to the number of filter positions in the wheel. Furthermore, the “filter” synchronisation rate corresponds to the effective number of filter positions in the wheel, so that the paired and tripled filter configurations are also directly supported. Therefore, the wheel(s) can now be synchronised to any of one, two, three or six camera frames per revolution. This makes support of the paired and tripled filter configurations in imaging mode just as straightforward as for the single ones. By the way, another advantage of treating multiple filters as single ones in this way averages out any optical differences between individual filters of the same wavelength, which can build up over time even if they were well matched to start with (we speak from experience here!).

MODULE INTERFACE IN STEP MODE

The module interface is likely to be of rather less direct use in step mode, which is signalled by the "Stopped" signal being high, as in that case different filter wheels can be in different positions, whereas only one set of positions can be sent to the modules. In a multiwheel system the positions sent via this interface will normally be those for wheel 1. The read/reset pulse combination is generated when any wheel steps to a new position, and the "Ready" signal will remain low until all wheels reach their new positions. When wheel 1 is at rest at a valid position, the appropriate wavelength output will be high, as will the "wavelength 8" one. The same as for continuous spinning, these outputs will only be high if the "shutter" input is high, unless this control has been disabled by the USB SET_SHUTTER command as described above.

When pairs of wheels are used in combined mode, giving up to 10 filter positions, the behaviour is slightly different. For filter positions 7 to 10, the "wavelength 7" rather than the "wavelength 8" output will be high, and these filter positions are output as wavelengths 1 to 4 respectively, which allows any third-party device to fully decode the effective filter position.

DIAGNOSTIC CARD

An optional diagnostic card can be connected to the module interface socket via the connecting cable supplied with it. The eight wavelength outputs illuminate eight LEDs, and there is also an 8-bit digital-to-analogue converter on board, which allows generation of an analogue output of 0 to 2.5V from these eight signals. Various alternative signals can be sent to these outputs, under control of corresponding USB commands. Connector pins are provided for access to the eight digital outputs and to the output of the digital-to-analogue converter that they drive.

When those diagnostic facilities that give an analogue output are in use, it will rise from zero to 50% of full scale at the start of a step, in order to provide a signal for triggering an oscilloscope used for observing the output. The output will remain at this level throughout the acceleration phase of a step, and it will then change with time during the deceleration phase, according to what is being measured, finally going to zero at the end of a step. The constant value during the acceleration phase of a step means that the most significant digital bit will be high throughout this period, so it may be more convenient to use this for oscilloscope triggering.

Firmware version 4.2 onwards supports a further diagnostic mode, where the output is a digital bit sequence for displaying the progress of the new braking code algorithm. In this case the most significant bit is high for the duration of the step, to provide the same oscilloscope triggering signal. Further information on all the diagnostic routines is given elsewhere in this manual, especially with reference to the USB commands that control them.

USB INTERFACE

This section is intended to give a general overview of what can be done via the USB. Since there are no front panel controls on the Optospin (but note the availability of the remote controller, which effectively takes the place of such controls), the equivalent functions are provided via our Optospin Control Program on the hosting PC. Or, since we would also like to encourage third party suppliers to support the Optospin, they may have incorporated USB interfacing within their own software. Since the Optospin's own internal software is likely to undergo continuing development (and will be updatable via the USB), we shan't attempt to give complete coverage here, and instead the currently implemented commands will be described fully in the Appendix.

The basic format is that each USB command sent by the PC has a unique two-byte identifier, which is then followed by however many additional data bytes (if any) needed for execution of that particular command. The Optospin software will return at least two bytes. The first is 0FF hexadecimal (255 decimal) if the command was successfully executed, in which case the second byte is the number of subsequent bytes (if any) that the command returns. If the command was not successfully executed, only two bytes are returned, the first of which is 0, and the second is a code number that gives basic information about the error (e.g. an attempt to step while a wheel is spinning).

An important point to note about USB communication is there is typically a time delay of a millisecond or possibly longer, so it's not appropriate for sending or receiving time-critical information; that is what the TTL interface is there for. However, this delay is of no consequence for many operations, and an interface of this type is pretty much essential to take full advantage of the flexibility of the Optospin anyway. So, much of the following sections is effectively a continuing description of the USB interface, albeit with the TTL interface also included where appropriate.

SOFTWARE INTERFACES

We have designed the Optospin IV to be equally at home as a standalone product or as a component of an integrated system supplied by ourselves or others. In many cases control will be via software packages such as Metamorph/Metafluor or MicroManager, using either the TTL or USB interfaces, or more generally a combination of the two. Therefore the software interface as provided to the user will be via these packages, so many of the details of what the Optospin can actually do, and how it does it, are likely to be hidden from the user. That is not necessarily a bad thing of course, but for people who do want to explore its operation in more detail, we have a couple of interfaces of our own.

Pictest

The first of these is a generic utility, initially developed just for in-house use, called PICTEST. This is a simple popup window that allows control data to be sent to and received from the USB. It is therefore capable of executing *any* of the USB commands listed in the Appendix. All one does is to give the identification number for the command, followed by however many further data bytes that command may require, and one then enters the number of bytes that are expected to be received (assuming the command was successfully executed). The program will then attempt to execute the command, and will report back with the received data bytes. If the program gets confused, usually because the wrong number of bytes has been received, then pressing the reset button should sort things out straight away. For convenience, some of the more "everyday" commands, with typical data parameters, are selectable from a preprogrammed list, to which the user can add their own if they wish. It is likely to be most useful to system installers, for general setup and testing, and for supporting the various functions of the diagnostic board for example. The crudeness of this interface is matched only by its versatility!

Optospin Control Program

The second utility is somewhat more advanced, although lacking the total flexibility of PICTEST, and is the OPTOSPIN CONTROL PROGRAM. This is a somewhat more polished interface, and is primarily designed to provide access to those specific facilities of the Optospin that third-party software may not provide, but which may nevertheless be appreciated by many users. Since this program is likely to be added to over time, it will probably be difficult to keep this description fully up to date, so for details it will probably be best to refer to the program itself to see what it can currently do – and to check our website, www.cairn-research.co.uk, for the latest updates.

Again this program is a popup window, so it can be used in conjunction with any other software package. One of its most useful functions is to be able to enter and display text information about the individual filters that have been installed in each wheel, so there should be no need to keep taking the system apart in order to have a look. This information is stored within the filter wheels themselves, so the system will *not* become confused if individual wheels are swapped around. It also allows the required speed (when spinning) and power (when stepping) to be entered and displayed, and again this is stored in filter wheel memory. There are also the usual Windows-style "controls" for manually stepping between filter positions, although yet again we must promote the far greater convenience of our remote controller for doing this sort of thing (why do people still think mice are such a great idea?). And finally, even though the Optospin's control software is of course perfect in every way, we have nevertheless decided to support the possibility of using the USB interface to install any updates that may become available.

SYSTEM OPERATION AND PERFORMANCE

SPIN MODE

The Optospin's main distinguishing feature over other filter wheels is its ability to spin continuously as well as to step. In design terms, this - especially the ability for multiple filter wheels to spin in precise frequency and phase with each other - was the more challenging requirement, as it required (for us at least) a new form of motor control. We're describing it here because it should make clear why the speed stability of the Optospin IV is so extraordinarily high.

It should also be noted here that the laws of physics mean that any rotating element is capable of continously spinning much more rapidly than it can step, so filter wheels are no exception. Therefore, if particularly fast “stepping” times are required, it's well worth giving serious thought to the possibility of continuously spinning the filter wheel instead, especially as the Optospin IV allows the choice to be made so easily.

The “simple” way of controlling the speed of an electric motor is just to vary the voltage fed to it. For greater stability a feedback system can be used. In this case the speed is somehow measured, and the difference between it and the target speed is used to adjust the voltage accordingly. Such systems can be made to work well, but by definition there has to be some sort of error to provide the feedback, and also the motor speed can't immediately respond to a voltage change (because of inertial effects), so “perfect” operation is difficult or impossible to achieve in practice. Even if the frequency control is excellent, there may still be small changes in the rotational phase of the motor compared with the reference. This makes it very difficult to guarantee that multiple wheels - each wheel necessarily with its own motor - will always remain in the same rotational phase.

So, although our previous designs had all used the feedback system described above, with the Optospin IV we decided to use a different approach, which is that of synchronous drive. Motors spin by virtue of rotation of the electromagnetic field that drives them. Normally the field is rotated by commutation, i.e. the rotation of the motor somehow causes the power to the motor coils to be switched so that the electromagnetic field always rotates in advance of it. Therefore the feedback control system doesn't need to get involved with any of this. However, an alternative is to power the motor coils in a precise sequence, so that the motor spins at exactly the same speed that the electromagnetic field is rotating. Now the only possible cause of error is that any change in the load on the motor will cause its rotational phase to shift somewhat relative to that of the field, but with a freely rotating filter wheel the loading will remain constant. Furthermore, it is quite usual for motors to be designed such that more than one cycle of coil energisations is required for each mechanical rotation, which reduces any errors still further. The motors we use actually require seven such cycles, each of which requires six sequential coil energisation states, per mechanical rotation, so the errors are insignificant in this application.

The downside of this approach is that - except at very low speeds - you can't just energise the coils to give the target speed and then expect the filter wheel to spin at that speed. Its inertia is simply too great. Instead, you have to start at a suitably low speed (we use 1Hz), and then steadily ramp up to the target one. Furthermore, at the highest speeds, in excess of 100Hz, the duration of each coil energisation state may be only a couple of hundred microseconds or so. Therefore the software that does all these calculations has to be pretty fast, so this was quite a challenging programming problem. For the record, we use an 18F series PIC microcontroller, programmed directly in assembler to achieve the necessary speed, and it does a great job.

For an internally generated target speed, the USB interface is used. The timings of the coil energisations are determined by an internal 1MHz reference clock, so from a programming point of view it is easiest to send the required speed as a rotation time, e.g. a speed of 50Hz can be specified by sending a time value of 20,000 microseconds. The PIC can then manipulate this value directly. Of course, such values generally can't be exactly divided up into the required number of coil energisation states, but as long as the division is done so as to distribute the rounding errors throughout the mechanical rotation cycle, this turns out not to matter at all. What is more important though is to get the overall total exactly right, as it also turns out that any errors here can build up surprisingly rapidly. The signals sent to the module interface are derived in the same way

However, it may be more convenient to provide the reference information as a speed instead, so this facility is also provided. In order to provide additional resolution, the speed values are multiplied by 100, to give a resolution of 0.01 Hz. So, to specify a speed of 33.75Hz for example, a value of 3375 is sent. Here though we have another version of that same division problem, as the speed won't necessarily convert into an integral number of microseconds. To get around this the PIC software keeps track of the rounding error, and corrects the rotation time accordingly whenever the error exceeds a microsecond. Even at 100Hz a microsecond adjustment corresponds to only 1/10,000 of a rotation, so the adjustment produces negligible jitter, but again one can see that the errors would build up significantly over time if not corrected in this way.

The other frequency control method is to use an external reference frequency. In fact, this works in a very similar way. The PIC software continually measures the reference frequency, converting it into a time value, and that value is then fed into the same control software as before. The wheels will then spin with the same frequency stability as that of the reference.

However, in this case we can usefully do something else, which is to get the phase right as well. When spinning under internal frequency control, we can use the wheel position itself to provide the reference for other equipment, so the issue doesn't arise there. But for an external frequency reference, the phase relationship between the spinning wheel(s) and the reference does become important, so it needs to be defined.

We have chosen this relationship such that the rising edge of the reference signal corresponds to midway between filter positions 6 and 1, to maintain a direct relationship with the various positional outputs on the TTL and Module Interface connectors.

Our remote controller unit has an external frequency generator built in, controlled by a slider potentiometer, so it can be used to control the spin speed if required. However, this facility is primarily intended for test and demonstration purposes, so it isn't accurately calibrated. Unless you want or need to synchronise to a reference frequency from some other source (as you may well do), then use of the USB commands to set the frequency is always recommended.

INTERACTION BETWEEN USB AND TTL SPIN CONTROL COMMANDS

The system starts up in step mode, and will remain in that mode until a valid spin command is received. When such a command is received, then by default all filter wheels will spin at the previously programmed speed (which will have been stored in the filter wheel memory). If one or more filter wheels are not required to spin, then they can be made to remain stationary by a USB command (SELECT_SPINNING_ROTORS). The spinning speed can also be set via the USB, using the SET_SPEEDx100 or SET_REV_INTERVAL commands. On receipt of the SPIN_ROTORS command, the currently selected filter wheels will spin at the currently selected speed, until the STOP_ROTORS command is received.

Spin control via the TTL interface is also possible, as long as the IGNORE_TTL command hasn't been issued via the USB (if it has, its effects can be reversed by tthe RESTORE_TTL command). In this case, the currently selected filter wheels will spin at the currently selected speed if the "spin" input is high (this input will default to low if it isn't connected). They will then stop if the TTL "spin" input goes low. However, the interaction between the USB and TTL interfaces for spinning is an ORed one, so the STOP_ROTORS command would also need to have been issued via the USB if there had been a previous SPIN_ROTORS one.

For both the TTL and the USB inputs, requests to spin the filter wheels while they are slowing down to stop, or to stop while they are spinning up to the requested speed, will be executed correctly, subject to the OR logic above. The situation with the IGNORE_TTL and RESTORE_TTL commands is as follows. If the TTL "spin" input is low, then these commands will have no effect on the spin state, but if it is high, then the following situation applies. If the filter wheels are currently spinning, there will again be no effect, but if they are currently slowing down or stopped (i.e. STOP_ROTORS has been executed, or a SPIN_ROTORS command has never been executed), then they will spin on execution of the RESTORE_TTL command.

Alternatively, the spinning speed can be controlled by a square-wave frequency applied to the "extref" TTL input, again unless the IGNORE_TTL command has been

issued via the USB. If such a frequency is detected, it will be measured and used as the speed reference. As long as this signal is present, its frequency will be measured on a continuing basis, so the filter wheels(s) will faithfully track any minor frequency changes. For PIC firmware versions prior to 4.0, significant frequency changes may cause temporary loss of synchronisation, but in that case the filter wheel(s) will automatically resynchronise. Subsequent versions should be able to take sudden frequency variations in their stride. If the reference signal disappears, the filter wheel(s) will remain spinning at the current speed, either until the reference is restored, or until the SET_SPEEDx100 or SET_REV_INTERVAL commands have been received via the USB.

STEP MODE

The design of a filter wheel system that can both spin continuously and step rapidly poses some interesting technical challenges, as the requirements for these two operating modes are potentially rather different. However, we have been able to achieve market-leading performance in both modes by using a relatively new type of motor. Of course, just about every possible motor configuration must have been developed over the years, but in practice one is limited to using whatever is commercially available at a reasonable cost, and for that there have to be one or more applications that generate a sufficiently large market. In this case the market is for electrically-driven model aircraft, which has been opened up by recent developments in battery technology. Itself driven by demands for lighter and more powerful batteries in laptop computers and mobile phones, for example, this technology has now advanced to the point where the performance of electric model aircraft can rival that of conventionally-fuelled ones, so the market for these motors has really taken off (sorry). Many types are now available, although they all tend to conform to the same basic format.

The motor requirements here are for a high power-to-weight ratio, and for a torqu-speed characteristic suitable for driving a propeller directly rather than through a gearbox. These requirements tend to be intermediate between those of conventional “high-revving” motors and stepper motors, so perhaps a little explanation is in order. In a high-speed motor, only one set of coil energisations - i.e electromagnetic field rotations - may be required for each mechanical rotation of the motor shaft. The other extreme is the stepper motor, where perhaps many tens of electromagnetic field rotations may be required for each mechanical one. In the stepper case, one is trading speed for torque, in that the effect is equivalent to that of a reduction gearbox. The increased torque means that inertial (or other) loads can be accelerated more quickly, but the electromagnetic field itself has to “spin” more rapidly in order to do this, and for efficiency reasons there is a limit to just how quickly this can be done. For relatively small angular movements, stepper motors are the devices of choice. However, the available torque for any electric motor tends to reduce as the speed increases, and the effectively greater gearing of a stepper motor makes this a relatively greater problem than for other types. It's just like driving a car in first gear

– you may be able to get off the mark quickly, but you can't go very fast. As discussed in the Appendix, this means that once one approaches a significant fraction of a mechanical rotation, a stepper motor can already be operating in a speed regime in which its torque - and hence overall performance - is already significantly reduced. The motors we use require “only” seven electromagmetic field rotations for every mechanical one, which still gives us sufficient torque when driven appropriately, but over a greater speed range.

The other problem with stepper motors is that their design tends to make them relatively bulky, since the motor coils are sited outside the part of the motor that rotates. This makes it difficult in practice to get them to drive a filter wheel directly, because they will tend to obstruct the light path when the motor and the filter wheel are on a common shaft. Therefore the filters have to be out on a greater radius (although one can at least take advantage of this by having more of them), so the inertia is significantly increased. Alternatively, they can drive the edge of the filter wheel via a reduction gear, but this will exacerbate the slew rate problem. And in either case, the physical size of the motors tends to give a dimension in the light path direction that is rather longer than may be desirable, complicating the connection of the filter wheel to other equipment.

By contrast, the low weight requirement for model aircraft motors has favoured a design that is also very compact for its rated power, as well as having the valuable intermediate drive characteristics. In this case the coils are *inside* the part of the motor that rotates, and the rotating part - which contains some small but very powerful permanent magnets - adds little to the overall diameter. This has allowed us to incorporate the motor directly in the wheel itself, while retaining a very modest overall size (100x100x35mm for a single wheel, or 170x100x35mm for a pair). The short 35mm path length along the optical axis, even when two wheels are in the light path, is particularly advantageous.

Mode of Operation

Although these aren't stepper motors, they can nevertheless be driven in a rather similar way, which allows them to be held in one of 42 basic positions as well as being made to spin. Of course, it is necessary for the drive electronics to have appropriate information about the wheel position at any time, and two systems are provided for this. The first is a fairly standard type of optical system, known as a quadrature encoder, which detects a series of grooves around the edge of the wheel. As the wheel rotates, each of two optical detectors goes alternately high and low, but they are 90 degrees out of phase with each other. This configuration allows the direction of rotation to be obtained as well, by feeding the optical pulses into an up/down counter, so that the absolute position can be tracked. However, some sort of reference also needs to be provided for these calculations, and in our system this is done by a series of magnetic sensors. The arrangement of these sensors and the small magnets in the wheel that they sense has been chosen so that when the filter wheel is

at rest at any given filter postion, a unique sensor combination is activated. Thus the drive system obtains direct confirmation that a requested step operation has been successfully executed, and if it hasn't for any reason, an appropriate correction can be applied automatically. The state of all these sensors is indicated by LEDs on the internal circuit boards, which is very useful if any troubleshooting is ever necessary - see the troubleshooting section for further information.

During a step operation, the PIC software looks after everything automatically of course. The filter wheel is accelerated for half the stepping distance, and decelerated for the other half, using the optical sensors to track the position. Its arrival at the target position is then verified from the magnetic sensors. Since the software runs much faster than the wheels can move, it can actually keep track of the instantaneous positions of up to four filter wheels simultaneously, even though they may be executing different stepping operations. The only limitation is that to step more than one wheel at the same time, this must be done by an appropriately formulated single command.

As already described, stepping can be initiated via either the TTL or USB interfaces. In both cases, the required new position of all (up to four) filter wheels is specified by the command, but only those wheels that are not already in that position will actually respond. Zero inputs in the commands are interpreted as “stay at current filter position”, so it isn't necessary to send explicit information for all wheels if only some are required to step.

For stepping, the existence of the combined and slave modes has already been noted. The main reason for their specific existence as modes, rather than just being different ways of using paired filter wheels, is primarily to do with the remote controller, as it has a specific switch that gives a choice between these two modes and independent operation. The remote controller has two banks of illuminated push buttons, six for each wheel. When the remote controller is switched to independent mode, the upper bank controls filter wheel 1 (or 3, or 1 and 3, according to the setting of another switch), and the lower bank controls filter wheel 2 (or 4, or 2 and 4). The positional information sent back from the TTL interface is used to illuminate the pushbutton for appropriate filter position for each wheel.

In slave mode the situation is almost the same, except that selecting a given filter position for either filter wheel drives both wheels to that position. This can be looked after entirely within the remote controller itself, so the TTL interface doesn't need to “know” that anything is different from independent operation. But for combined mode the situation is rather different. Here filter positions 1-6 are selected by the upper pushbutton bank, and filter positions 7-10 are selected by the first four of the lower pushbutton bank (the other two now do nothing). Now only one pushbutton is illuminated rather than two, and it doesn't necessarily correspond to the physical position of either wheel. Since the pushbutton illumination is controlled by the positional outputs from the TTL interface, then the interface now somehow does need

to know that a combined mode command has been sent, and to react accordingly.

As previously noted, positions 1-6 on the TTL interface (both as inputs and outputs) are encoded as bit patterns 001-110 respectively. A bit pattern of 000 as an input means “stay at current position”, but this leaves 111 spare. It is therefore used as follows. This pattern for either wheel sends it to (what should be!) its open filter position 6, and also informs the TTL interface that a combined mode command has been sent. The appropriate bit pattern for the other wheel encodes its actual required position directly, just as before. (In principle, sending 111 to both wheels would put them both in their open positions, but as that may not be desirable we have decided not to allow it in combined mode.) The consequence of sending 111 for either wheel is that the TTL interface responds with slightly different positional information. For the wheel in its open position 6, 111 is sent back, both as an alternative way to signify this position *and* to inform the remote controller that the pushbuttons should be illuminated appropriately for combined mode. If 111 is sent back for filter wheel 2, then the controller just has to prevent illumination of the pushbuttons in the lower bank, while illuminating the upper bank as before. If 111 is sent back for filter wheel 1, then filter position 1 for the second wheel needs to illuminate the position 6 pushbutton in the upper bank, and positions 2-5 need to illuminate pushbuttons 1-4 in the lower bank. This is all easily looked after by the logic circuitry in the remote controller. So, as far as third-party equipment is concerned, if combined mode is being used, then when reading from the TTL interface, that equipment just needs to treat 111 as well as 110 as signalling position 6 for either wheel.

Combined mode commands can also be executed via USB. The USB_GO command provides a direct equivalent of the TTL interface, and it takes two bytes in exactly the same format, so it includes combined mode support. However, the only reason for using a combined mode command here (although it may be a very valid one) is so that the TTL interface then provides the appropriate outputs for illuminating the pushbuttons on the remote controller correctly for this mode. For slave mode operation via USB one just sends the same positional information to both wheels in a pair of course.

One other facility worth noting here is that there are USB commands for storing and recalling the mode information in the filter wheels themselves, i.e. whether they are intended to be used independently or as combined or slaved pairs. However, they are for possible information purposes only, as the control software makes no explicit use of them.

There is of course no need for third-party developers to get involved in any of this if they don't want to, even though it's easy enough. However, for further assistance, we have also provided a completely straightforward USB stepping command, USB_INDEP_GO. This just takes four bytes, one for each (potential) wheel. Filter positions 1-6 are directly encoded by 1-6 for each byte, again with 0 meaning “stay at current position”.

The motor current during a step, which determines the stepping speed, is also under user control via the SET_STEP_POWER USB command, which is described in more detail in the following section. Higher currents give faster stepping, although there are diminishing returns on account of the square-root relation as described in the Appendix. However, they will also cause more vibration (but again note the advantages of slave mode here!) and will cause the filter wheels to become warmer in operation, although that shouldn't be enough to cause damage. In addition, each wheel contains a temperature sensor, allowing the actual temperature to be measured if required, via the MEASURE_TEMPERATURES and READ_TEMPERATURES USB commands.

Finally in this section, a more detailed description of the “ready” signals may be useful, as there are actually two in step mode. The first is on pin 12 of the TTL interface, and its status is also shown by the green LED on the remote controller. It goes low (LED off) at the start of a step, and goes high once all the filter wheels that are stepping have reached their target positions. This indicates that it is now safe for recording equipment to gather data. However, a variable interval can be programmed via the SET_RESTEP_DELAY USB command, during which further stepping commands will be refused. During both the step and this period, the signal on pin 24 of the TTL interface, which is otherwise high in step mode (it's low when one or more wheels are spinning) is low, and the associated red LED on the remote controller is off. The primary reason for providing this is to prevent other equipment from requesting unrealistically short “dwell” times at a given position, which may cause erratic operation under some circumstances, as well as placing more thermal stresses on the motor.

As previously noted, the coding enhancements in version 4.2 of the PIC firmware allow much shorter “dwell” times, so the motor can now work correspondingly harder, and hence get correspondingly hotter. The extent to which elevated temperature can reduce electric motor power generally (perhaps approaching 50%), as a result of both increased resistance and reduced magnet strength, doesn't seem to be widely realised, but fortunately such effects are fully reversible. For the Optospin, in practice they serve to protect the motor, because any such temperature-dependent power loss causes the stepping to malfunction before there can be any permanent damage. However, such conditions are likely to be encountered only for extended periods of stepping frequencies on the order of 10 per second or more, so we consider it unlikely for them to be encountered in practice anyway.

ASSEMBLY AND SETTING UP

For comprehensive information, please refer to the separate Setup Guide, so this section just provides some background information. The Optospin takes standard 25mm filters, of up to 6mm thickness, and they are secured in place by retaining rings with a 24.9x0.7mm thread, as used by Comar Instruments (Cambridge, UK) for the

"tubemount" optical hardware system. For continuous spinning, we recommend that the filters are reasonably well balanced in order to minimise vibration. The thickness, and hence weight, of filters used to vary somewhat according to their type, since they were based on a multilayered type of construction, but the currently preferred manufacturing process of "sputtering" needs just a single layer, so the filters from any given manufacturer now all tend to weigh the same. We can also supply blank (ie opaque) "filters" of the same weight, so this is now a pretty straightforward matter. However, if any application requires use of filters of differing weights, we do have a workaround in the form of using custom retaining rings of weights complementary to that of each filter.

The compact size of the Optospin means that it is normally possible to incorporate it directly in the space between a microscope sideport and a camera, as shown in the Setup Guide. The wheel assembly can be removed without affecting the integrity of the coupling, which makes the filters very readily accessible. The same applies to the alternative coupling that allows two wheels to be accommodated within the same physical and optical path length. In both cases, the slight convergence of the light path as it focusses towards the camera has negligible effect on the filter characteristics, and the only requirement is that the filters should be fully at right angles to the light path in order to avoid any astigmatism, but that is clearly going to be the preferred configuration anyway.

However, there are other occasions where one may wish to use a wheel in a collimated ("infinity") light path. Here it is important that any reflections from the filters aren't reflected *directly* back, where they might otherwise form ghost images as a result of being refocussed by the collimating lens. The solution here is to angle the wheel by a few (typically five) degrees, and for such applications we can provide the components for doing this. Unlike the situation for a focussing light path, angling a filter in an infinity space does *not* cause any astigmatism!

Each filter wheel connects to the controller box via a cable with a hybrid D connector at each end. Since each wheel can draw peak currents in excess of 10 amps, a rather substantial cable and connector pins are required for the power connections. In addition to this, ten signal connections are required, for which the requirmeents are far less stringent. The hybrid D connectors that we use here are particularly suitable for this combination of requirements.

In versions of the controller box that can can control one or two filter wheels (which is likely to be sufficient for the majority of requirements), the power supply is built in. For three- or four filter wheel versions, the power supply is an external unit, both to provide more space inside the controller for the extra electronics, and to allow provision of a necessarily more powerful unit.

The Optospin has been designed to allow the stepping performance to be optimised according to the total mass of the wheel, which depends to some extent on the mass

of the filters of course. To minimise the moving mass, the wheel is normally made of Delrin, but in applications where the wheel may be subjected to high temperatures (e.g. when used in close proximity to an arc lamp) an aluminium alternative is available, although its additional mass will inevitably reduce the stepping performance somewhat. Another reason for minimising the moving mass is that the acceleration and deceleration of the wheel during a step generates a countertorque that must be resisted by whatever other equipment to which it is attached, so this is a significant potential source of vibration. If that may be troublesome, and a more relaxed stepping performance can be tolerated, a facility (addressed by the SET_STEP_POWER USB command as previously noted) is provided for setting a wide range of stepping power levels. However, where vibration needs to be minimised, the advantages of using two wheels "slaved" together, so that their countertorques cancel out, cannot be overemphasised. Furthermore, since each wheel now only needs to have three filters installed (odd number positions in one wheel, even number positions in the other), the stepping performance will also be superior to that obtainable from a single wheel.

Whatever the configuration, optimum performance is obtained by a combination of three different control systems, which can broadly be described as feedforward, feedback and learning. All of these are under user control via USB commands, although in practice only the feedforward control needs to be set up with any care, and in any case we will have done this for you if we have supplied the system with filters installed (otherwise we'll have made a reasonable guess based on "typical" filters, which is likely to be close enough anyway).

The basis of the feedforward control is as follows. During the decleration phase of a step, the filter wheel is to some extent acting as a dynamo, so some of the current that was previously used to accelerate it is now effectively fed back into the power supply to assist the deceleration. This means that the applied decelerating current needs to be correspondingly less, according to the speed at which the wheel is moving at any time. The required corrections turn out to be quite large (and to depend on the total mass of the wheel), so in order to reduce the amount of work that the other control systems have to do, it makes good sense to apply a good estimate of them "in advance", hence this is feedforward control rather than feedback.

The relative amount of feedforward is set by the same USB command, "SET_STEP_POWER" that is used (as its name suggests!) to set the power for a given wheel during a step. A full list and description of the USB commands is given in the Appendix. Many users won't need to refer to these at all, as they may instead be accessed via third-party software or our own Optospin Control Program, but if required they can all be accessed via our simple "PICtest" utility, so this section is written as if that facility is being used here. First though, it's important that the filter wheel is secured to something relatively immoveable before *any* of the following adjustments are made. Otherwise the high stepping torques will tend to rotate the wheel housing in the opposite direction, which will give downright misleading sensor

information!

The SET_STEP_POWER command takes three bytes, one to specify the filter wheel, one to set the step power itself, and one to set the feedforward (“deceleration”) factor. By convention we give all numbers in hexadecimal format, i.e. 0-0FFh to represent decimal numbers 0-255. A typical step power is around 80h (but can be as high as 0F0h), and a typical deceleration factor is in the range 60-0A0h, with less massive wheels being at the higer end and more massive wheels at the lower end of this range. To set the optimum deceleration factor we just rely on trial and error, but it's important for the feedback and learning facilities to be turned off while this is being carried out. This is done by the DISABLE_FEEDBACK and DISABLE_LEARNING commands. For PIC firmware version 4.2 onwards, execution of the DISABLE-BRAKING command is also recommended, Please note that with all these switched off, the stepping performance is likely to be quite “wobbly”, to the extent that inappropriate deceleration factors may cause control of the wheel to be lost altogether (in which case an auto-correction routine will get it to its target position eventually!). The goal is, for a given stepping power, to find that value of deceleration factor that gives best overall stepping control between filters that are one, two or three psoitions away from each other. It should be possible to optimise the deceleration factor within a range of about 8-10h.

These values are stored in filter wheel memory, so they stay with that particular wheel and are read back whenever the sytem is restarted. Therefore, once set up (and this is usually done by us), no further attention should be needed. However, interested users (or their agents) may find the following further information helpful. First, we find our remote pushbutton unit to be a particularly convenient way of checking the stepping performance between various different filter positions, and we certainly prefer it to any software interface. Second, the digital-to-analogue converter in our diagnostic unit allows the feedforward correction to be displayed in real time on an oscilloscope, in conjunction with the DISPLAY_FEEDFORWARD command that sends this information to the module interface. As explained previously, the diagnostic card output will rise to 50% of full scale at the start of a step, but it will then be seen to suddenly reduce at the start of the deceleration phase, and then progressively increase (albeit in stepwise fashion) back to near its original level as the deceleration phase continues. The optimum amount of feedforward is likely to be the one that gives the most linear increase during the deceleration phase of a step. The amplitude relative to the original level is a direct reperesentation of the relative power (or strictly, current) being fed to the motor, so the effect of this adjustment can be seen to be quite large!

Once this has been done, we can revert the system to normal operation. That can be done via a restart, but for convenience we also provide the CLEAR_DIAGNOSTICS command.

The feedback and learning controls are much more straightforward as far as the user

is concerned, as they are both simply on or off. Since they both improve the stepping performance, we naturally recommend that they are both enabled in normal use. The operation of the feedback system is relatively easy to describe. During the deceleration phase of a step, optical sensors provide information on when the wheel reaches various positions. If these times differ from those expected, then the motor current is adjusted so as to correct for the error. These adjustments are in addition to those provided by the deceleration factor. They can be observed (if required) on the analogue output of the diagnostic card by executing the "DISPLAY_FEEDBACK" command, which displays the amount of feedback correction being applied at various times during the decelaration phase. Since the feedback is correcting for "unpredictable" factors such as varying motor efficiency between different coil phases, this waveform is likely to be relatively erratic. In fact, if it shows any significant upward or downward trend, that is probably an indication that the amount of feedforward has not been optimised.

Unlike the feedforward adjustment, the feedback can either increase or decrease the power fed to the motor. These adjustments are shown relative to the original 50% output level, and to make them more clearly visible (since they are going to be relatively smaller than the feedforward ones), their deviation from this level is multiplied by a factor of four. The operation of the feedback system is particularly noticeable if the filter wheel is not attached to anything else. As mentioned above, this situation must be avoided during setup, but it may nevertheless be instructive to see just how well the feedback corrects for the counter-rotation of the housing, compared with when this facility is switched off!

Next we come to the learning facility. This is based on the likelihood that although the feedback is correcting for "unpredictable" errors, these errors may well have a significant systematic element, i.e. they are going to be the same every time. It is basically an adjunct to the feedback system, and works as follows. Whenever a feedback adjustment is made at any point in the deceleration phase of a step, the system remembers any error at that point, and adjusts the value of an associated variable accordingly. The next time that particular step is repeated, that particular variable is used *in advance*, to adjust the motor current in a direction so as to reduce the expected error this time. The idea is to take out the systematic component of the error, so in order to prevent the system fruitlessly attempting to cancel the random component (which we can never know of course), the adjustment on any one occasion is relatively small. Thus the random errors average out over successive occasions, whereas the systematic ones build up in the variables and hence are progressively corrected.

The effect of these adjustments can be shown on the diagnostic card output by the DISPLAY_LEARNING command. The output is the same as for DISPLAY_FEEDBACK, i.e. starting at 50% of full scale, with the adjustments (either positive or negative) being amplified by a factor of four in order to make them more visible.

In principle we could store all this information in filter wheel memory, but it turns out that there there is more to store than there is available space there. Therefore the learning facility starts anew every time the system is switched on or reset, or when the DISABLE_LEARNING command is issued (in which case the ENABLE_LEARNING command must then be issued to switch it back on again of course). In practice it takes just a few steps between a given pair of filter positions for a useful amount of learning to take place for that particular step. Since it copes so well with systematic errors, it also corrects very well for non-optimal settings of the deceleration factor (in the SET_STEP_POWER command as described above). While we don't want to encourage users to rely on this, it does mean that if users install filters of significantly different mass from those for which the system had been set up, optimum performance will still be obtained. The only difference is that rather more learning will need to be done than otherwise, so the first few steps between any pair of filter positions may have rather more “wobble” than if the deceleration factor had been optimised.

And in order to see how this all goes together, we have the DISPLAY_POWER command, which shows (assuming they are all enabled!) the combined effects of feedforward, feedback and learning. The output format is the same as for DISPLAY_FEEDFORWARD, i.e. initally 50% of full scale, and then relative to this value as the deceleration phase progresses.

But now there is more, in terms of both hardware and software! First, on the hardware side, for the last year or two we've been adding what we call “registration magnets” to the wheel and its housing. The reason for this is that although the motor is energised between steps in order to hold it at the required filter position, the restoration torque for *small* displacements from the required central position is itself quite small. This meant that the wheel wouldn't necessarily come to rest in *exactly* the same position each time. Although the filter was always fully in the light path, this uncertainty was sufficient to affect the accuracy of the stepping algorithms. Adding the registration magnets provided an additional restoring force for these small displacements, thereby providing a local “detent” effect to keep the filter reproducibly central.

Second, we have been adding a mechanical damper that just gently presses against the motor housing. This doesn't seem to affect the continuous spinning, but for stepping it does cause any “wobbles” about the target position to die out more quickly. Whether we'll continue to do this is currently under review, as the third introduction, described next, has turned out to be particularly effective. However, we may well continue to incorporate an element of mechanical damping if it does provide any further help.

Third, PIC firmware version 4.2 onwards incorporates a “braking” algorithm at the end of a step, which actively damps any tendency for the wheel to wobble at the end

of a step. The problem here is to bring the wheel to a precise stop at the centre of the required filter position, after having been substantially accelerated and then decelerated. Any energy mismatch between the acceleration and deceleration phases will result in a nonzero final speed and/or a positional error, which all the above control systems are attempting to minimise, but it is difficult to avoid some residual. The effect of such a residual is potentially to cause a slight “wobble” of the filter around its central position. The filter remains fully in the light path, but if the wheel is asked to step again before its position has fully stabilised, the performance of that step is compromised, and the problem can build up during succeeding steps to the extent that control of the wheel is eventually lost.

Before version 4.2 the Optospin IV could still step up to several times per second, and for these rates and higher we felt it would make more sense to spin the wheel continuously instead, but it nevertheless it became clear that there was a significant market requirement for more frequent stepping, so we have now added some further control code to actively correct such residual speed and/or position errors, rather than waiting for them to die away. This now allows stepping rates up into double figures, with dwell times comparable to the stepping times (hint, you really might as well spin the wheel continuously instead now!). For those who care about such things (another hint, perhaps you should get out more, although it might be of genuine interest to some), the way we did this, while having relatively little sensor information available, is described in detail in the Appendix. But it's also given in outline form in the description of the DISPLAY_BRAKING command, which gives a digital display of the progress of the braking algorithm on the diagnostic board. Whether or not the braking code actually runs is controlled by the ENABLE_BRAKING and DISABLE_BRAKING commands, but it runs by default at switchon, and the only reasons for disabling it are for setting up the feedforward parameter in in SET_STEP_POWER, and for curiosity to show the difference that it makes.

To return the module interface to its normal mode of operation, either a system reset or the CLEAR_DIAGNOSTICS command can be used. Both of these will (re-)enable the feedback, learning and braking code for all filter wheels. Feedforward is *always* in operation, because it is so important, although it could effectively be disabled (why???) by setting a deceleration factor of 0.

MAINTENANCE AND TROUBLESHOOTING

Since each filter wheel is mounted directly onto its drive motor, there are no gear or belt linkages to worry about, so the only scope for mechanical failure apart is that of the motor bearings themselves - and they seem to be pretty reliable in our experience. Of greater risk of course is some electronic malfunction that causes the motor coils to overheat, so we have incorporated circuitry to protect the motors against all but catastrophic failure. This means that in the event of such a malfunction, (e.g. a massive interference pulse giving the PIC software a nervous breakdown), there's a

pretty good chance that all will be well again if you switch the system off for a few moments.

In the event of any drastic control problem, first check the connector cables, especially those to the filter wheels, to ensure they are all fully and correctly plugged in. If the problem persists, it may be because of trouble with one or more of the filter wheel sensors. To assist in diagnosis, the board(s) in the system box that drive the filter wheel(s) each have five LEDs that show the current state of each of the five sensors. Reading from left to right on the board, they are labelled M3, M2, M1, O2 and O1. The first three are for magnetic sensors, and they should all be illuminated for most of the time. At certain wheel positions, corresponding to one of the filters being fully in the light path, one of more of these LEDs will go out, indicating that a magnet in the filter wheel has been detected by the corresponding sensor. If any of these sensors are defective (or more likely, if there is a cabling or connector fault) then that LED will be permanently illuminated, and the filter wheel is unlikely to initialise or spin properly.

The other two LEDs are for the optical sensors around the edge of the wheel. Both should be on about half the time, and they should switch on and off in an overlapping sequence as the wheel rotates, *IF permanent sensor illumination has been selected by the jumper link on the filter wheel circuit board*, otherwise they will both be off except when the wheel is actually stepping. (As noted elsewhere, the infrared illumination of these sensors could cause problems in very low-light-level applications, which is why we provide the option to switch them off during the periods between steps, so that images can be acquired without any infrared contamination.) The jumper link is at the bottom left corner of the circuit board, visible behind the filter wheel, and for permanent illumination it should be in the CONT position. These sensors are interrogated only during stepping, so if a filter wheel initialises and spins normally but doesn't step properly, then a fault with these sensors (or again, the cabling and connectors) is to be suspected.

The following notes are primarily intended for system installers and servicers only! The sensitivity of the optical sensors can be varied by the preset next to the jumper link, and although it shouldn't need adjustment, it can be checked as follows. Select permanent sensor illumination with the jumper link if it isn't set already, and set the wheel to spin continuously at some convenient speed. The signals from the two optical sensors can be found on the filter wheel's driver board, at the points labelled on the silkscreen (near the bottom left corner) as O1 and 02 respectively. Each should be a square wave with close to 50% duty cycle, with the two being a quarter of a cycle out of phase with each other. When the wheel is stopped at any filter position, the O2 signal should be high, and the O1 signal should be near its low-high transition point, so may be in either state. If all these conditions are met but the wheel still isn't stepping properly, then the fault is likely to be on the main controller board. Even in this case all is not lost, as the active devices are are all socketed, and we can then advise as to which are the most likely to need replacing (and if you're

wondering why we haven't switched over to surface mount technology, perhaps you can now see why!).

APPENDIX

Moment of Inertia

Should you experience a sudden attack of the type of laziness suggested by the title of this section, and are interested in the design principles for a rapidly-stepping filter wheel, you could do worse than put your feet up with a cup of coffee and read at least some of what follows.

First of all, to accelerate an object, i.e. to change its speed over time, you have to put some energy into it. Since this will take a finite time, the object is said to have an inertia, since the energy transfer has the effect of providing a resistance to the force, and this resistance is directly related to (and effectively is) the mass m of the object. The amount of energy imparted to an object is the product of the force applied and the distance over which it acts. For a constant force, the acceleration is also constant, that is to say the speed of movement (termed the velocity V, as that takes the direction of movement into account as well) increases linearly with time T. But the catch is that as the object moves ever faster, the (constant) force exerted over a given time period is applied over an increasingly greater distance, so the rate at which energy is imparted to the object will also increase with time.

So for a constant force F, and a constant acceleration dV/dT, the instantaneous energy transfer is given by the instantaneous velocity V, multiplied by dV/dT, so to get the total energy transfer over a given time T, we just integrate VdV/dT, which is just $V^2/2$. This relation is for a unit mass, so once we've added the actual mass m to the relation, we come up with the "standard" equation for kinetic energy, i.e. $mV^2/2$.

That's for linear movement. Rotary motion is basically the same, except that for a given speed of rotation, the parts of the object that are further away from the axis of rotation will be moving faster. Therefore we have to consider the object as consisting of a large number of individual small masses, calculate the kinetic energy of each one, and then sum the result in order to get that of the whole object. For a wheel this is pretty straightforward, as all the points on a given radius will be moving at the same velocity, so we just need to take the velocity dependence with radius into account. The other slight wrinkle is the somewhat different velocity unit in this case, which is the radial velocity ω, measured in radians per second, where a full rotation is 2π radians.

The maths follows directly from two simple physical facts. First, as the radius R of the wheel increases, the total amount of material at any given radius r within it increases linearly with r, since it is proportional to the circumference, $2\pi r$, at that radius. So, the total number of those small masses in the wheel increases with R^2.

Second, although the angular velocity ω is the same for all points on the wheel, those small masses that are further away from the axis are moving correspondingly faster, so have correspondingly more kinetic energy. This increases with the square of r, so overall we're going to end up with a fourth-power relationship. Here's the full derivation.

The velocity V of a small mass *m* at a radius r is given by rω, so we just need to make that substitution into the linear motion equation, to get a kinetic energy of $m(r\omega)^2/2$. We can now lump together all the small masses at equal r to get the total mass of a thin ring of material of density D, radial width dr and thickness t, equal to $2\pi rdrtD$ (which is just volume x density). So, combining those two equations, we get a kinetic energy for the ring of $\pi rdrtD(r\omega)^2$, i.e. $\pi Dt\omega^2r^3dr$ after rearranging terms.

At this point we can introduce the concept of the "moment of inertia" for rotary movement. This is the same equation, but with the ω^2 term stripped out, so it's the equivalent of mass as far as rotary motion is concerned. To get the moment of inertia of the wheel as a whole, we just sum the values for all those individual rings, over a radius range going from 0 to the radius R of the wheel. That's just standard integration again, and since the integral of r^3 over this range is $R^4/4$ we get a moment of rotary inertia of $\pi DtR^4/4$ for the wheel as a whole.

As the equation shows, this is for a wheel of a given thickness. If you scale up all three dimensions, then the thickness would increase as well, giving a fifth-power relation overall. That compares with a third-power one for linear motion, since there the relation varies only with the mass. But for our application there's no need (within reasonable limits of course) to increase the thickness as well as the radius if you want a bigger wheel, so the fourth-power relation is what we need.

What this very clearly shows is that the penalties for using a bigger wheel are enormous. Clearly it helps to make the wheel out of a low-density material, but the benefits of this are as nothing compared with keeping the radius down. And near the edges, it helps disproportionately to reduce the amount of material there, which is why our wheels have arc-shaped regions of reduced thickness between adjacent filters.

So to summarise all this as “Rule 1”, we have:

1. The moment of inertia of a spinning wheel of given thickness increases with the fourth power of the radius.

But while we're on the subject, we can derive a few more useful relationships. Since they apply to both linear and rotary motion, we can keep things simpler by just considering the linear case. First, we can drive the distance/time relationship for a constant accelerating force. In this case the velocity V increases linearly with time,

being given by FT/m, where F is the accelerating force and m is the mass. The distance travelled in a small time dT is given by VdT, i.e. FTdT/m. Integrating that over the total time T gives a total distance of $FT^2/2m$. Therefore we have our second relation.

2. For a constant accelerating force applied to an object initally at rest, the distance travelled increases with the square of the time.

An alternative way of stating this, which is more applicable to our filter wheel discussions is:

3. For a constant accelerating force applied to an object initially at rest, the time required to travel a given distance increases with the square root of that distance.

So, if this situation applies to a filter wheel, the time taken to step a given number of filter positions increases only with the square root of that number. And from the same relation, we can also see that:

4. The time required to move a given distance varies inversely with the square root of the accelerating force.

5. The time required to move a given distance increases with the square root of the mass (or moment of inertia).

And since for rotary motion the moment of inertia increases with the fourth power of the radius, we directly have:

6. The time required for a wheel of given thickness, to move a given angular distance, when accelerated by a given force, increases with the square of the radius.

That doesn't sound quite so bad as the fourth power relation for the moment of inertia, but the amount of energy you have to put into the system to achieve a given time for the distance travelled is still going to increase with the fourth power though! And whatever the moment of inertia actually is, you'll still need four times the motor power in order to halve the time.

Of course, calculating the times in this simple way neglects the fact that the object is reaching its destination while moving, whereas for stepping we need it to both start and finish at rest. However, the relations given here are still directly applicable if we break the problem into two parts. To step a wheel from one angular position to another, so that it both starts and finishes at rest, we need to apply a constant rotary force (i.e. torque) in one angular direction until the wheel is halfway to its final position, and then reverse the direction for the other half of the distance. Thus the acceleration and deceleration phases are symmetrical about this midpoint, so the time required for a particular stepping operation is just given by twice the time required to

move half the total distance.

We also have to take into account that the filter wheel is neither of uniform mass (because of the motor and the filters) or thickness, but the basic inertia principles still apply. But when comparing performance specifications from different manufacturers, it's important to ask whether or not they are obtained with filters in the wheel. A set of six 25mm filters is likely to weigh around 25-30gm (see the section on wheel balancing), whereas our aluminium wheel weighs 40.6 gm and the delrin one only 20.6 gm, so the filters are a significant extra inertial load. For the record, the rotating portion of the motor weighs 31gm, but since its radius of 14mm is so much less than the 45mm radius of the wheel, its not going to contribute significantly to the total inertia.

To summarise, to achieve a short stepping time, the best thing you can do is to keep the wheel radius as small as possible. And while it clearly helps to use a more powerful motor, the inverse square-root relationship between torque and stepping time means that there are diminishing returns here. This analysis also assumes that the torque is of the same magnitude throughout the step, but is it in practice? It may well not be, so let's consider that now.

Torque-Speed Considerations

A direct current electric motor, in which the electromagnetic field is rotated by commutation (i.e. the coils are switched by the rotation of the motor shaft), will generally produce its maximum torque when at rest. As the speed increases, the torque tends to reduce, for two reasons. First, the rotation produces a “back emf”, which tends to reduce the current through the motor. Second, the motor windings have an inductance, which limits the rate at which the current through the coils will change as they are switched on and off by the commutation. Both effects can be countered by driving the motor from a current rather than a voltage source, which we therefore do, but there are limits to how far this can be taken. From our previous physics primer, we can see that high torque at high speed constitutes a lot of energy transfer (it goes up with the square of the speed if the torque stays constant), so the motor design is going to impose some fundamental limits here. Therefore, in practice there has to be some falloff in torque with increasing speed, so even an unloaded motor won't accelerate indefinitely.

Although this is a potentially complex subject, in practice the torque is likely to decline in a roughly linear manner with increasing speed, with zero being marked by the no-load speed. Therefore, in order to maintain high torque throughout a stepping operation, the maximum speed reached during the step needs to be low compared with the no-load speed. Consider a wheel that is required to have a worst-case stepping time of say 50msec (which ours can do). Worst-case means moving half way round, which means an average speed of 10Hz (1/20 sec for a 50% rotation). For a constant accelerating torque followed by a constant decelerating one, then the

speed will increase linearly with time for the first half of the step, and decrease similarly for the second half. Therefore the maximum speed, at the halfway point, will actually be double the average, i.e. 20Hz. For best performance, it is therefore important that the torque generated at this speed should not be significantly lower than when the motor is at rest. This implies that the no-load speed should be several times higher, say at least 100Hz. We therefore come to the apparently surprising but very nice conclusion that a filter wheel that can both step and spin rapidly is in no way a design compromise, since for continuous spinning we can get up towards the no-load speed if we want to.

This is not to dismiss stepper motors out of hand, as they are available in so many different formats. Indeed, some motors that are described as stepper motors actually have fewer steps per revolution than the 42 that "our" motors effectively do. But the whole point of a stepper motor is to provide a high torque at low speed, and to stay locked at the target position (i.e. to have a high detent torque in the stepper motor jargon). These requirements are best met by having many steps per revolution, but this is just equivalent to having a (greater) reduction gearbox; the torque goes up, but the speed goes down. A quick perusal of stepper motor data sheets shows that the torque may already be significantly reduced for rotational speeds as low as a few Hz. Also, the high detent torque can introduce a fair amount of vibration. On the other hand, there's no question that filter wheels based on stepper motors can be made to perform pretty well, and we certainly had to do a lot of work in order achieve the improvements that our choice of motor potentially offered.

The New Braking Algorithm in PIC Version 4.2 Onwards

INTRODUCTION

In addition to the feedforward, feedback and learning algorithms for the Optospin, we later added registration magnets to provide definite "detents" at each filter position. Without those, the wheel could stop at a slightly different position each time, because there tended to be a flat spot in the torque versus displacement profile when the coils were energised in "stepper motor" mode, to hold it at the required filter position. This meant that the length of the acceleration phase of the following step would also vary slightly, which reduced the accuracy of the control algorithms. That uncertainty has now been removed, but there is still a problem if the wheel still has some residual velocity when it reaches the new filter position. The registration magnets provide a locally strong restoring force to bring the wheel back if it overshoots or forward if it undershoots, but the system is somewhat underdamped, causing a decaying oscillation around the new position (but for which the mechanical damper provides at least a partial cure). If the wheel is going to stay there for a while, the oscillations seem too small to matter, but if a new step is initiated during this period, they will affect its accuracy, and the effects tend to be cumulative over successive steps, eventually causing loss of control.

The target position always corresponds to a transition point for the MCLOCK1 optical sensor. However, even for a “perfect” step we don't know which side of that transition point the wheel will end up, so we can't use that signal as a guarantee of having reached the target position. Therefore, the length of that final deceleration step to the target position has to be calculated and timed, rather than measured, and until now we've used that to determine when to switch the motor coils to “stepper motor” mode, as explained above. There is therefore necessarily some uncertainty as to whether the wheel has stopped at the target position at this calculated time. If not, any combination of speed and positional errors is going to cause its position to oscillate somewhat as described above

To put the problem in perspective, the combination of all the control systems and the registration magnets seems to allow minimum “dwell” times at a given filter position of as short as around 100msec in a “good” wheel, but this can't be guaranteed (there is inevitably some variation between individual wheels, which might be reduced by more careful setup). However, some people want to reduce this to perhaps only 30msec or so. This is way shorter than our original design target (of more like 250msec), but let's see if we can do it.

DAMPING THE SYSTEM

In principle the easiest way of dealing with this is to introduce some mechanical damping into the system. Experiments to date suggest that it doesn't need to be very much in comparison with the motor torques during stepping, so could be left in place all the time, which is what we have been currently doing. In any case, if we're looking for dwell times as short as 30msec, any mechanism for controlling a damper would have to operate quickly relative to that timescale – possible but not so easy. But on the other hand, a permanent damper could affect the continuous spinning, and in any case a damper may be subject to wear.

Whether we have a mechanical damper or not, the idea originally was that we might be able to introduce some electronic damping into the system. Although as previously explained we can't use an MCLOCK1 transition to signal arrival at the target position, then especially now that we now have the registration magnets, any speed or positional errors are likely to cause multiple transitions as the wheel wobbles about that position. I was therefore originally wondering about using those to damp the motion electronically somehow, but thinking along those lines resulted in an alternative possible approach to the problem.

CORRECTION RATHER THAN DAMPING

In fact, what has emerged from these thoughts doesn't directly involve the presence of the registration magnets at all, and it's not so much a damping as an electronic correction system, although in principle it could perhaps be run multiple times at ever-decreasing currents in order to provide a damping type of effect if needed

(although in practice that doesn't turn out to be necessary). The difference from the sort of thing I originally had in mind is that the algorithm described here theoretically sorts out everything in a single pass. Also theoretically, a mechanical damper should no longer be necessary, and might even be detrimental, but in practice a small amount may nevertheless help – this is currently under more detailed investigation.

The concepts developed here necessarily involve a number of assumptions. First, they assume that the motor torque just depends on the current, whereas in practice there is also some dependence on the wheel's angular position relative to the motor coil commutation points. However, if the angular changes in wheel position are relatively small and the torques for the two drive directions are symmetrical with respect to it, then we should be ok (in practice, the "forward" and "reverse" torques correspond to electrical phase changes of +/-60 degrees relative to the target position, equivalent to +/-8.6 mechanical degrees). Second, we have previously discovered and corrected for (via a feedforward algorithm) the fact that when the wheel is decelerating there seems to be an additional speed-dependent deceleration torque, which reduces the externally required current. It is assumed here that the speeds reached during the correction algorithm aren't high enough for that effect to be significant under these circumstances. Third, we neglect the effects of the registration magnets, but we may well be able to do so to at least a first approximation for reasons that will be discussed later. And finally, we neglect the effects of any damping in the system, but it's the lack of that which is causing the problems in the first place.

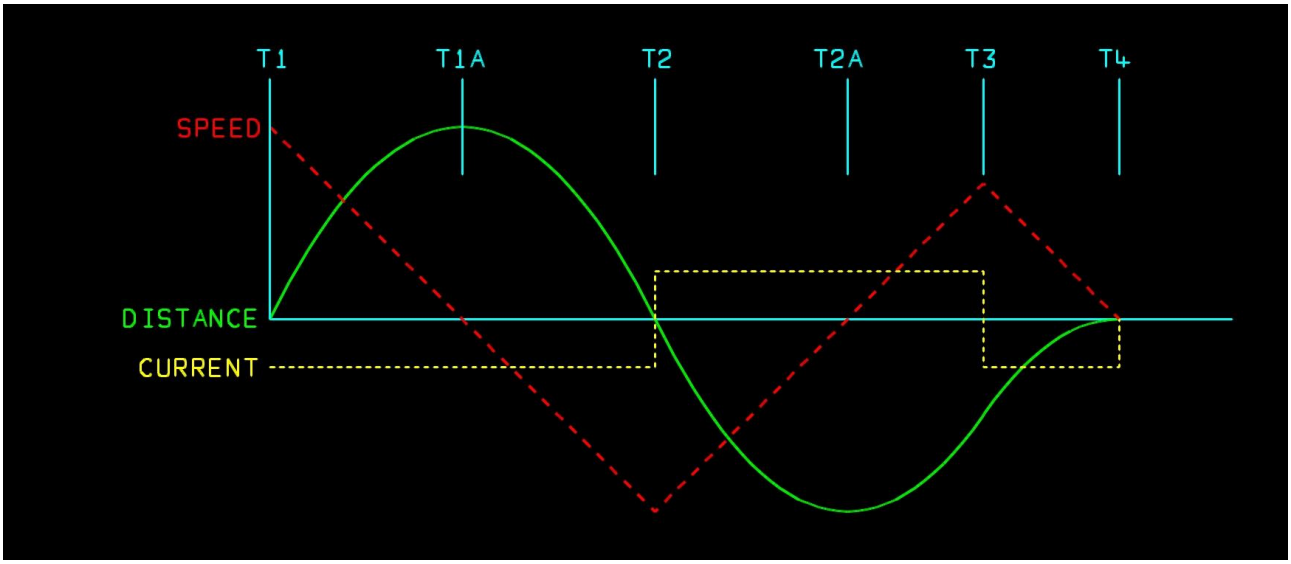
THE CORRECTION ALGORITHM

Considering that final deceleration step towards the target position, which we are timing because we can't guarantee an MCLOCK1 sensor transition there, there are two possible error situations. If the wheel is going too fast at the start of that final step for the calculated deceleration current to bring it to a halt at the target position, it will overshoot the final position, but more importantly it will cause that MCLOCK1 transition before the calculated time. As we shall see, this is the point at which the correction algorithm will kick in. Conversely, if the wheel was going more slowly or decelerating too rapidly, that transition won't have occurred by the end of the calculated time, and might not occur at all if the deceleration had been sufficient to reverse the wheel's direction or if we stop just short of the transition point. What we can do here is to reverse the motor current at the end of that calculated time, so that we *do* eventually get an overshoot as the wheel now accelerates towards the target position, at which point we restore the current to its original decelerating direction, and then run the same algorithm as for the original overshooting case.

Let's start at the time the wheel starts to overshoot the target position, which we have ensured will happen. To keep things simple we are going to switch to some constant current throughout the correction algorithm, and just mess around with its direction. First, we just let that overshoot happen, and because the direction of the motor

current is such as to bring the wheel back again, we'll get an opposite sensor transition as it goes back past the target position. At this point we reverse the motor current! We now have a symmetrical situation, as the wheel should now undershoot by as much as it originally overshot. The wheel is now at rest again, but it's now in the undershot position rather than the overshot one, so we'll need to have another acceleration-deceleration cycle to get it at rest again at the target position.

As to why we want to allow the wheel to overshoot first, the graph should start to make things clear. It shows the three key parameters against time. These are the motor current direction, which is just positive or negative (or effectively zero at the end), the rotational speed of the wheel, which linearly increases or decreases according to the current direction, and the slightly trickier case of the wheel's angular position, which is the integral of the speed versus the time. The current direction is effectively just plus or minus 1, but without further information we simply don't know the calibration of the speed and position with respect to the current value – but we're keeping that constant. Although that information is potentially available from other measurements, the nice point is that, thanks to allowing that initial overshoot, it turns out to be available *within this algorithm itself*.



Let's work this through. The wheel first passes the target position at time T1 (measured by the MCLOCK1 transition) at some unknown speed, but at a known and constant motor current that we have just switched to. The overshoot occurs at some unknown distance further on, at a time that we'll just call T1A, as we don't need to do anything there and we don't know what it is yet anyway. However, at that point the wheel comes to a halt, and then comes back towards the target position, following a symmetric profile with respect to both speed and distance as it accelerates back past the target position, causing an opposite MCLOCK1 transition at a measured time T2. The symmetry means that we now know what T1A was, as it must have been half way between T1 and T2. This is the key to the self-calibrating nature of the

algorithm as we'll see.

At T2 we reverse the motor current, to give speed and distance profiles opposite to those after T1. Therefore the wheel comes to a halt at an equal and opposite distance from the target as it did for the original overshoot. This happens at a time T2A, which we can now predict because the interval between T2 and T2A must be equal to the now known interval between T1 and T1A. Again as for T1A, we don't need to do anything at T2A.

The wheel was stationary at T2A, but now accelerates back towards its target position. To bring it to a halt at its target position (which is of course the object of the exercise), we need to reverse the motor current once more at T3, such that the wheel arrives there at a predicted time T4, when we switch the motor coils to "stepper motor" mode, so as to symmetrically hold the wheel around this position.

The symmetry of the graph shows that the time differences between T2A and T3, and between T3 and T4, must be equal, which is handy because we need to know both. Clearly we need to calculate T3 first, which we can do from knowing that for a constant accelerating force, the distance travelled is proportional to the square of the time, or to put it in the form we need here, the time we need to go a given distance is proportional to the square root of the distance.

Again we can see from the symmetry of the graph that the final motor current reversal must occur when the wheel is at a distance half way between its overshoot and target positions. This will therefore occur at a time beyond T2A that is not 1/2 but 1/sqrt2, or about 70% as long as the interval between T1A and T2. That gives us T3 and hence T4!

T3 = T2 + (T2-T1)/2 +1/sqrt2*(T2-T1)/2

Or simplifying, and making a reasonable numeric valuation for the square root term, since our PIC processor is brain-dead when it comes to maths, we have

T3 = T2 + 0.85(T2-T1)

And for T4 we have

T4 = T3 +0.35(T2-T1)

Can life really be this simple? I guess we'll find out! (We did. It was.)

RATIONALE FOR THE ALGORITHM

The fundamental control problem that we have here is that in order to bring an object to a stop at some specified position, we need to know both its current position and its

speed. If the position sensor gives a continuous signal, then we can measure the speed by differentiating it, but the Optospin's sensors only give discontinuous signals. In control terms the positional information is relatively coarse, but we can and do extract additional information by timing the transitions between the individual sensor positions. That turns out to be sufficient to control the basic motion of the wheel during a step, but inevitably there are going to be residual errors at the end of it. These can cause the wheel to wobble around its target position as already noted, and if the next step starts before it has settled, then the subsequent step may be affected, causing a larger error after that one, and so on.

The problem is that these wobbles may be below the resolution of the position sensors, and even if they are large enough to detect, the positional resolution is just too coarse for us to directly estimate the speed from it. However, we do know that the wobble will be about the target position, which is a sensor transition point, so we will get both positional and timing information there. The trick we employ with the algorithm is to drive the wheel in such a way that we can effectively work out the speed of the wheel *at each transition*. That's where the idea for this crazy algorithm came from. Having simultaneous speed and positional information allows the equations of motion to be solved exactly, which is effectively what is done by the rest of the algorithm. We don't need to know the actual speeds and detailed positions, because the algorithm cancels these out for us, just leaving us with the times at which we need to do things.

To repeat, a particularly nice feature of the algorithm is that it is completely independent of all the other control systems, so if the deceleration factor isn't optimal, or if the learning algorithm hasn't properly done its stuff yet, it should still work properly. But how well will that be in practice?

PRACTICAL ISSUES

My main practical concern is the registration magnets, as they are going to provide a position-dependent restoring force with respect to the target position. This will make the the wheel's speed versus time somewhat nonlinear. However, there are potentially two mitigating factors. First, the motor current we use during the algorithm is likely to be relatively high. The current we use during the accelerating phase of the step is likely to be a good choice, and this is likely to generate a motor torque that is significantly higher than that from the magnets, in which case their effects will be relatively small. Second, the symmetrical movement of the wheel between T1 and T2 will tend to make the effects of the magnets cancel out, so the speed measurement we effectively make here shouldn't be too much affected.

As to the possible effects of the registration magnets on the all-important T3 calculation, this is harder to assess, but there is also likely to be some built-in correction. In any case, there is a danger of over-analysing here, especially if the electromechanical resistive losses or other things we haven't thought about may be

big enough to take into account after all. In practice the way to handle all this will be to play around with the multiplication factors in the T3 and T4 calculations, to see what works best. If need be we could make them user-programmable, but my provisional feeling was that it would be better not to unless they really do make a big difference, and in practice they don't seem to.

And as for the T4 estimate. the time at which we switch the motor to “stepper motor” mode, there is at least in principle another possibility here. Since this is a timed step, just as for the final deceleration one, we could run the whole algorithm again, according to whether we arrive significantly early or late here. In that case we would probably use a lower motor current to allow the algorithm to run more slowly, in the expectation that any wobble this time would be less. Other things being equal, a lower current would cause a relatively bigger overshoot for a given speed error at the start of the algorithm, but since the first pass of the algorithm should have reduced that error, the wobble should still be less overall the second (or subsequent?) time around. However, in practice there has turned out to be no need to do this. The most noteworthy observation is that any residual errors no longer accumulate during a fast step sequence, so just a single pass seems to work just fine.

From a user point of view, once this algorithm begins to run, we will already be “at” the target position, as the target filter will be and remain fully in the light path in spite of any initial wobble. We therefore will potentially have the entire minimum dwell time period in which to run it, which in practice is going to be plenty, as we need only a few milliseconds..

Now we “just” have to code all this....

CODING ISSUES

For coding, we have the same issues as for the rest of the stepping code, in that more than one wheel may be stepping at the same time, so we can't just have linear code that waits in sequence for each of the sensor transitions or calculated times to be reached. Instead we need to run in that same multitasking environment, where the PIC goes around a loop that checks a whole series of potential tasks in turn, and carries out each individual one only when it is ready. We therefore need a fairly detailed set of flags to mark off each task as it has been done. Although this makes the code harder to write, it does give a nice way of following its execution, by using the diagnostic board to display the digital status of each of these as the braking code executes. This is selected by the DISPLAY_BRAKING USB command, for which the individual flag bits are as follows:

0 Mainly for internal housekeeping, remains set throughout the code if the wheel initially arrived at the target position early.
1 A control flag, primarily for internal use, which briefly goes high at several points during the braking code. However it remains high from the end of the braking

code proper until the restep delay timer finishes, so providing a visual indication of that interval.

2 If the wheel hadn't reached the target position when the braking code starts, this bit will be high until it has.

3 Once at the target position, the wheel will go *slightly* beyond it and then return. This bit will be high while that is happening. Referring to the Figure, this bit will be high between T1 and T2

4 The wheel will now go *slightly* to the other side of the target position, and then reverse back towards it. This bit will now go high until the wheel reverses, corresponding to the interval between T2 and T3.

5 The wheel will now return fully to the target position. This bit will be high while that is occurring, corresponding to the interval between T3 and T4.

6 This bit will be high while the braking code is running. It is intended to be a synchronisation pulse for viewing all this on an oscilloscope.

7 Same as for the other diagnostic routines, this bit will be high from the start of the entire step onwards.

While this sort of detail may be more than a user would need, it is indespensible for writing and then debugging the code, especially because of the multitasking environment. Even so, the situation seemed pretty hopeless until suddenly everything started working. Realtime code is like that.... The main and welcome discovery was that the algorithm is pretty robust. In particular, although the coefficients for calculating T3 and T4 do seem to be "about right", they don't seem to be particularly critical, so we decided not to make them user-adjustable. Even our decision to make the braking current programmable (by SET_BRAKE_POWER) was a marginal one, but at least that is consistent with the other current parameters being programmable too.

A BRIEF HISTORY OF OUR FILTER WHEELS

When Martin, our company founder, and also the author of this little lot, first got a filter wheel up and running, he little suspected that the darned things would become such a significant part of his professional life. Indeed, there have been times when he wished he could never see one again, but since Cairn got its business start by commercialising one, such sentiments are perhaps not entirely fair.

So, how did it all start? Well, back in the days when he was a young and innocent postdoc, he needed to make some differential absorbance measurements, and to do that he put together a system that could switch between wavelengths very quickly. To do that, a spinning filter wheel seemed the obvious answer. For both speed and simplicity, the drive method he used was a jet of compressed air directed against the edge of the wheel. This worked really well, the only problem being a strange slow cyclical speed variation, eventually traced to the pressure of the lab's air line not being as well regulated as it could have been. A customised smaller version, using 9mm filters, achieved speeds of around 650 revolutons per second (40,000 rpm!) in

tests, so this really is the method of choice if you want to go that fast.

But those speeds weren't actually necessary, so for greater convenience when using the system in other labs, the next version was a mains-powered fan with a motor in the hub, with the blades cut off and a belt to couple the remnants to the filter wheel. This also worked well, but it was a bit crude for a commercial system, and could only spin at one fixed speed. However, our first commercial design was quite similar in many ways, as the type of motor we used also had a rotating case, so it had the same belt drive system for the filter wheel. The main difference was that the motor was DC-powered, and incorporated a feedback-stabilised speed control system. The speed was sensed optically, by detecting a white stripe on the edge of the wheel. This worked very well as long as the stripe remained in good condition, which it tended not to on account of its exposed location, so these early users did need to get the painters in from time to time. Apart from that and the occasional stretched or broken drive belt, the system tended not to need much if any mainternance, and a few are still in use even now.

Our next version was somewhat more sophisticated though, in that it could properly step as well as spin - the original just had an escapement-type mechanism to allow it to be stopped and held at a given filter position. This version was conceptually very similar to the current one, in that the motor was in the hub, and yet again had a rotating case. However, the motor wasn't anywhere near as powerful as the type we're now using, primarily because the magnet strength was a lot less, so we had to drive it pretty hard to get good performance. This made it more vulnerable to burning out in the event of a malfunction elsewhere (the favourite one being someone blocking the wheel rotation by spearing it with a light guide!). The case was also unnecessarily big and heavy, so we had to perform the additional step of turning it down to a more reasonable size.

All our filter wheels up to this point had used 12.5mm diameter filters, as this had enabled them to be smaller and hence faster than ones based on the 25mm size. That was fine for us at the time, as all these designs were intended for changing the fluorescence excitation wavelength, where it's relatively straightforward to squeeze all the excitation light through apertures of this size. However, once we introduced our Optoscan monochromator in the late 1990s, it pretty much took over from filter wheels for this application, and then the motor we were using was discontinued, effectively killing the product.

We then decided to scale up to a 25mm filter design, as this is much more appropriate for using in imaging pathways, and hence as an emission wavelength filter changer, but we never made the initial product in any quantity. Since we were rather dsiappointed by the relatively low torque (compared with our expectations) of the motor we'd been using before, we opted for a rather more ambitious design this time, in which the filter wheel was part of the motor itself. We did this by incorporating a series of magnets around the rim of the wheel, which ran between coils and pole

pieces arranged like the calliper brakes around the rim of a bicycle wheel. We were also using the more powerful type of magnets (neodymium iron boron) that were then becoming more readily available, and these factors combined to produce vastly higher torques. However, the design proved difficult to build, and the coils around the edge took up a lot of extra space, although the dimension in the light path direction was our shortest ever.

After that experience, further work on filter wheels was delayed for a while because of pressure for development of other products, but we didn't want to let things rest there. So, once we'd acquainted ourselves with those model aircraft motors, the work resumed. It was now clear to us that a motor in the hub was the best approach after all. However, again there was a delay, as once we'd got a prototype running as-proof-of-concept, we started getting further ideas, most notably the one of being able to get two wheels into the light path with no additional path length requirement. That required a mechanical redesign. Also, the PIC control software needed a fair amount of development, as it was all written in assembler to get the necessary speed, not to mention the further complications of being able to control up to four wheels simultaneously. But we got there in the end, and we hope you like the result.