A motorised fluorescence cube linear positioner

1. Introduction

Fluorescence microscopes require fluorescence 'cubes', i.e. assemblies of excitation, dichromatic and emission filters and automated microscopes require a system to move one of a selection of such cubes into the optical path. Commercial microscopes often employ a rotating, motorised turret mechanism holding six or more cubes. However, very often only a few of thee are actually used. A circular geometry has the advantage of compactness, utilising available space rather efficiently. On the other hand, a linear geometry is more suitable when dealing with a restricted number of cubes. Such a linear geometry is much simpler to design and we present here such an arrangement. In common with most of our optical mechanics, we make our design compatible with the Thorlabs (http://www.thorlabs.de) range of optomechanical components, i.e. we make use of their 6 mm diameter 'cage' rod system and SM1 lens tube fittings.

Our laboratory uses several Nikon microscopes and we have thus made this system compatible with their current (!) range of cubes (http://www.nikoninstruments.eu/Products/Microscope-Systems) which have 96xxx part numbers and use 25 mm diameter filters. The filters are held in a plastic housing, as shown in Figure 1, and are used in the TE2000 and 90i series of microscopes.

Figure 1. The Nikon fluorescence cube suitable for the design presented in this note.

2. Servo electronics

We use a very simple position servo system, using a geared DC motor coupled to a rack and pinion system which converts circular motion into linear motion to drive the 'cubes' to their required position. Feedback is provided by Hall effect digital magnetic sensors which 'read' the cube slider position using encoded position, defined by a 2 bit code. The code is determined by small permanent magnets inserted in the cube slider. A Microchip PIC microprocessor reads the magnet code and drives the motor in the appropriate direction until the desired position is reached. In addition, we provide two microswitches to sense an open side lid on the assemblies and drive the slider to the appropriate extreme so as to facilitate cube interchange.

The full circuit of this servo is shown in Figure 2. The set-point is determined either through an external I^2C interface or a local three-position toggle switch. The PIC (a Microchip 16F876, 28 pin)

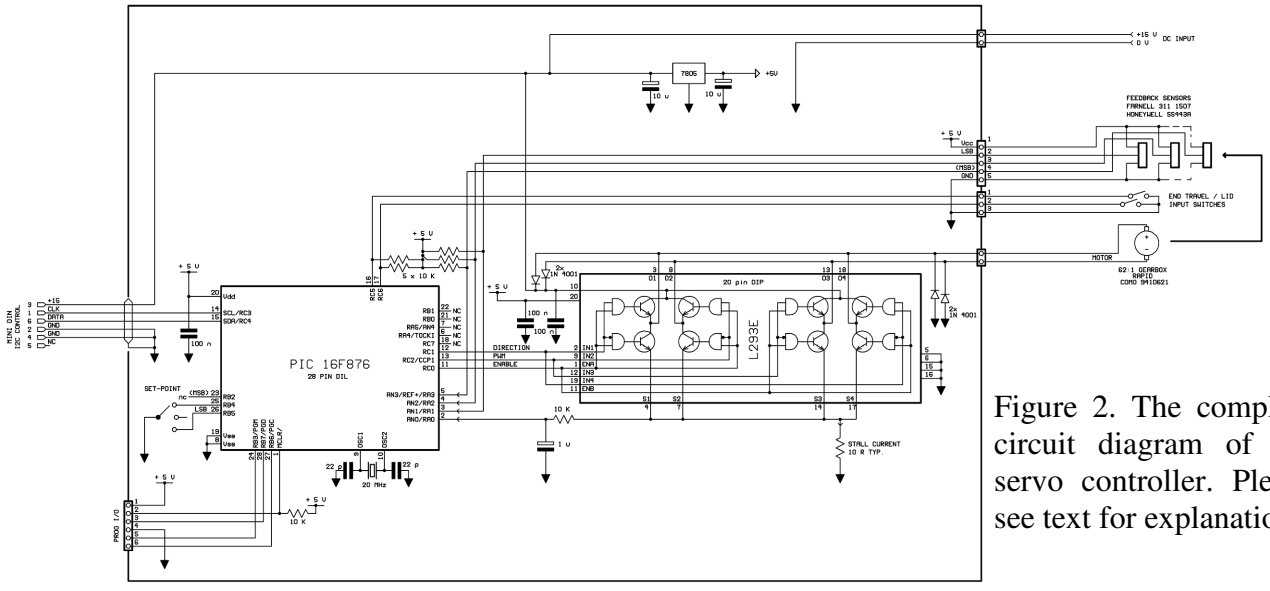


Figure 2. The complete circuit diagram of the servo controller. Please see text for explanation.

is able to read up to three Hall effect switches (though only two are used here), making the same basic design suitable for setting up to 7 positions and drives the DC motor through a bipolar H-bridge driver, made form two paralleled halves of a L293E 20 pin chip (ST Microlectronics, Farnell 146-7711). A device for converting USB data to I2C data is described in one of our other application notes: "USB1 communications interface for controlling instruments".

The position sensors used are Honeywell S&C type SS443A (Farnell 3111507) and they are employed in conjunction with small neodymium permanent magnets, obtained from Rapid Electronics (http://www.rapidonline.com), type M1219-2, order code 78-1066. These sensors are mounted on a small, fixed printed circuit board, while pairs of magnet locations are allocated on the slide mechanism. The position codes are defined by whether or not the magnets are inserted in them, along the lines indicted in Figure 3.

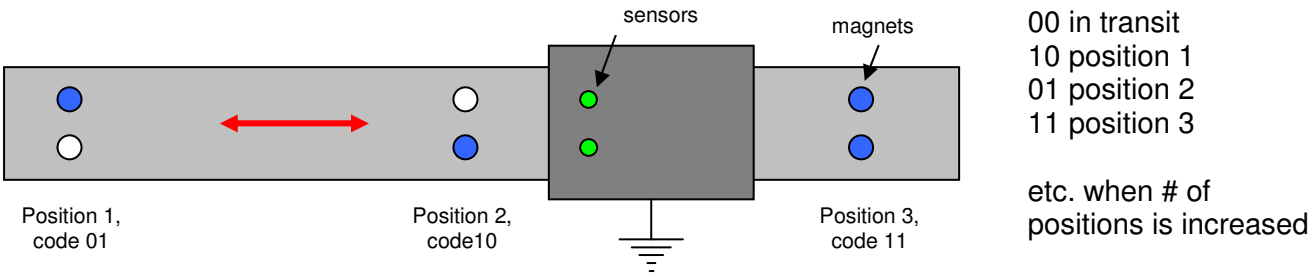


Figure 3: Arrangement of magnets (their presence indicated by the blue dots) placed in holes along the moving part of the assembly, and the sensors (green) on the printed circuit board.

The microcontroller continuously reads the outputs from the two sensors: code 00 indicates that the stage is travelling and when the required code is reached the motor either stops or travels past the required position for a short time, reverses and approaches the target position from the same direction. This ensures not only a high degree of positional accuracy but also removes any potential issues with hysteresis in the gearbox or coupling. With hindsight, we should have used a Gray code (which only allows a single bit change at adjacent positions (i.e. codes 01, 11, 10, rather than 01, 10, 11) but nevertheless the firmware code in the PIC copes well with finding the required position. There is always some potential for misreading the code, so any change in either of the bits is sensed and a timer activated for a few milliseconds. After this time (adjusted for the travel speed required), the stage is stopped. More details can be found in the section dealing with software and firmware.

We used a Microchip 16F876 controller; this requires a crystal in order to operate at the appropriate clock frequency (20 MHz in our case). More recently PICs with an internal clock oscillator have become available and these could equally well be used in this application. The PIC provides three signals to the motor driver: an enable signal, a signal to determine motor direction and a pulse width modulated output to determine motor speed. Using these three signals allows all motor conditions to be taken care of.

3. Servo printed circuit boards

The servo system is constructed on two double-sided boards: one contains the sensors and the other the rest of the circuit. All interconnections are made with 0.1" pitch board-to-wire Molex connectors. The main circuit board is itself housed in a small (73 x 51 x 25 mm) plastic case (Multicomp HBT3, Farnell part# 645680) which also contains the set-point 3-way toggle switch. The sensor board layout is shown in Figure 4, while that of the main board is shown in Figure 5.

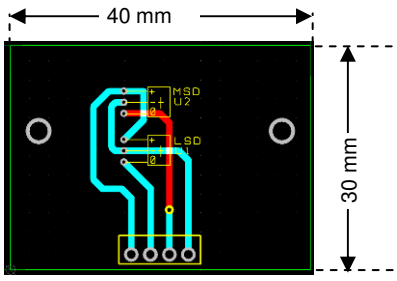
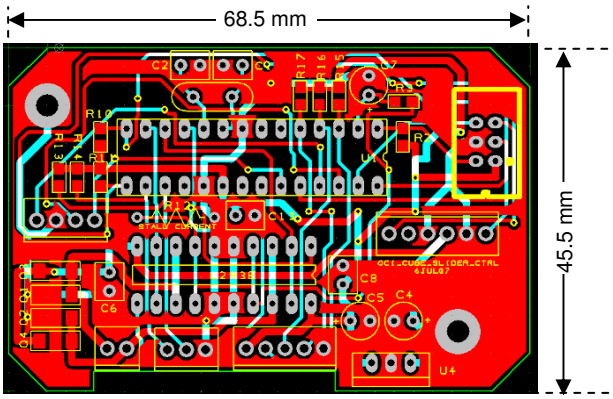


Figure 4. Printed circuit board layout of the sensor board. Not much to be described here!



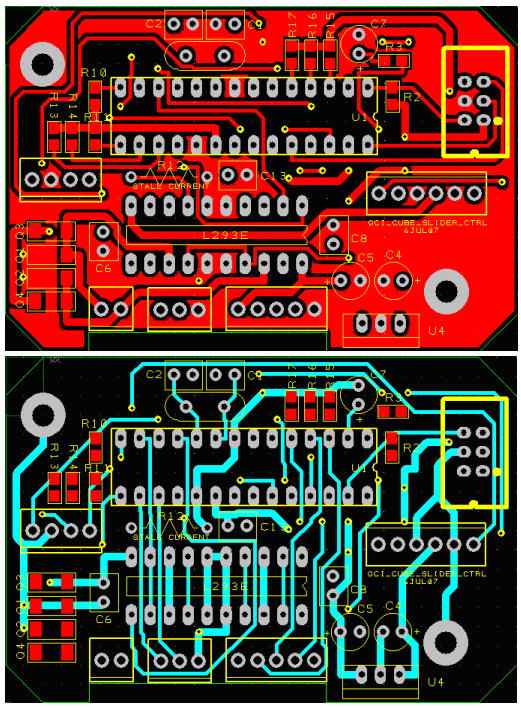
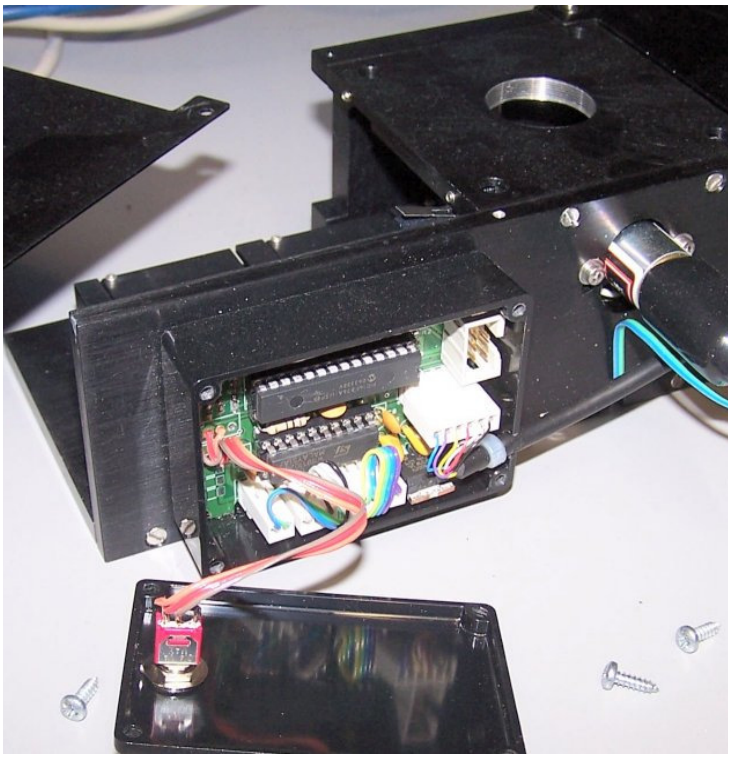
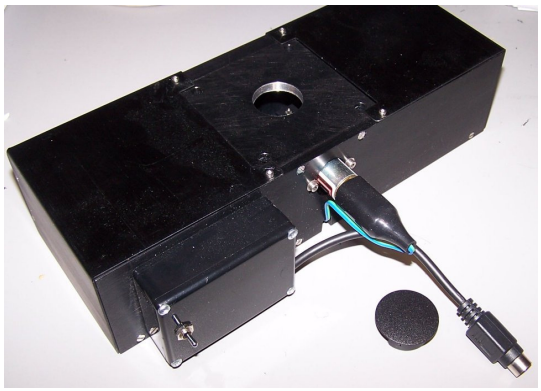


Figure 5. Printed circuit board layout of the servo system electronics.

We use PCB pool for board manufacture (http://www.pcb-pool.com/ppuk/info.html) and board assembly is very quick as there are so few components! The assembled printed circuit board, mounted in the box and the rest of the assembly is shown in Figure 6.

Figure 6. Assembled electronics in plastic case on the side of the cube drive unit. The motor, sensor and microswitch connections are at the bottom of the board, having been routed through apertures in the assembly. The i/o connections are made through a multi-way cable exiting the side of the box and terminating in a 6 way mini-DIN connector, as seen in the lower image.





4. Mechanical system

Since mechanical loads on the motor are fairly low and stage travel speeds are similarly low, a low cost geared motor has been used to provide mechanical drive. With this combination of gearing, cubes can be changed in ~2 seconds going from one end to the other, i.e. in just over 1 second when travelling between adjacent positions. The motor is a MFA/Como 941D series, coupled to a 62:1 ratio epicyclic gearbox (http://mfacomodrills.com/). It can be obtained from Rapid Electronics (http://www.rapidonline.com/) as part number 37-1198. The gearbox is coupled to a small 9 mm diameter 0.5 module plastic pinion, also available form Rapid Electronics as part number 37-0200,

which in turn drives linear 0.5 module Delrin rack. The rack can be obtained from Davall Stock Gears / Stock drive Products (www.sdp-si.com) as part number A1M12MYZ0525A. An alternative supplier for the rack is Huco (http://www.huco.com), part# 104291601.

The construction of the unit can be seen in Figure 7. A long baseplate holds a pair of 100 mm travel, 8 mm tall linear ballslides (Deltron DA-5, http://www.deltron.com/) Similar ballslides are also available from Automotion (http://www.automotioncomponents.co.uk/), part # L1029.014-100 and doubtless from other suppliers. This plate is made wide enough so as to be compatible with the Thorlabs (http://www.thorlabs.de) 60 mm cage system (e.g. LC6W) and allows mounting rods from this system to pass trough the assembly without interference form any other components. A small plate is attached to the front of this baseplate, and arranged to be compatible with the 30 mm cage system (e.g. C4W). An SM! Hole is tapped into this plate to allow a lens to be mounted ahead of the fluorescence cubes which are dovetail-mounted on a plate attached to the moving parts of the ballslides. The front plate width is restricted to allow cubes to be removed or interchanged. In contrast, a full length rear plate is attached to the baseplate and the drive motor is mounted on it. A top plate attached to the rear and front plates is attached, completing the assembly. Finally two folded aluminium sheets are fitted around the sides to complete the assembly. Small microswitches are used to sense when the cover plates are removed, driving the ballslide so as to expose the cubes.

The feedback system described earlier is constructed by fitting a thin plate to the back of the dovetailed cube plate and mounting the magnets within this. The sensor printed circuit board is fitted to the bottom plate with a 'U' shaped block (shown in blue on the SolidWorks model.

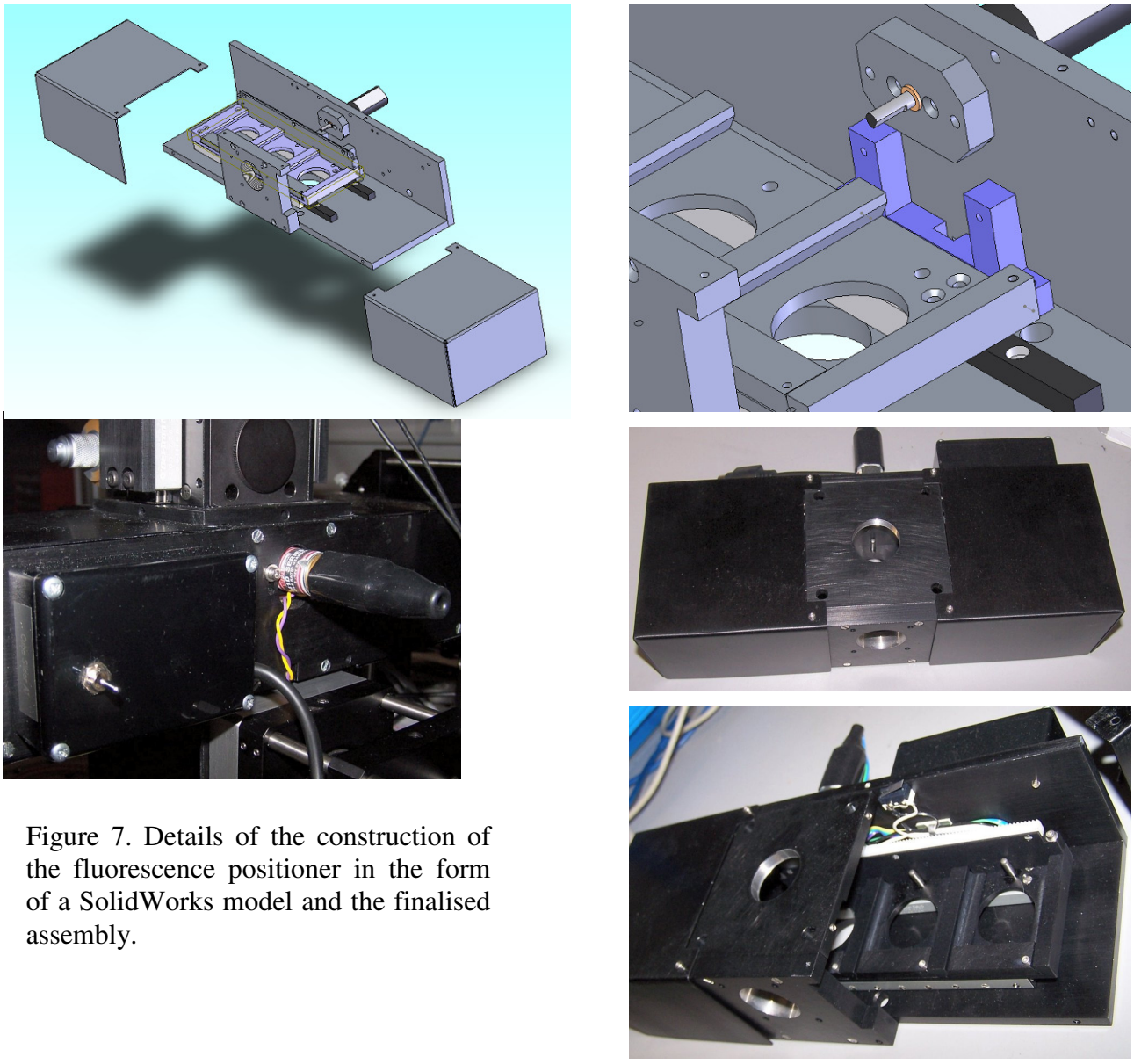


Figure 7. Details of the construction of the fluorescence positioner in the form of a SolidWorks model and the finalised assembly.

5. PIC firmware

The firmware was written using a CCS C-code compiler (http://www.ccsinfo.com/) which makes generating the code considerably easier than using an assembler code language such as Microchip MPLAB. The sample code below may be found useful should future modifications be required. There are probably neater or more succinct ways of writing code, but nevertheless, the sample code below satisfies our needs.

```
//Compiler used CCS PCWH 4.101
//Program to control cube slider with 3 cubes using 2 Hall effect sensors
//Inputs are:
//A1,A2 hall effect position inputs
//B4,B5 set position controls
//C5,C6 cover micro-switches
//Modified 8 Jan 08 - changed write listing in the I2C read interrupt so I2C clock is released later (2us) after data than in the
original CCS compiler
//25 May 10 added longer delay on overcurrent stop

#include "C:\Program Files\PICC\Programs\Cube position\cube_position.h"
#define RX_BUF_LEN  10                                          //This must be 10 or less or it takes to long to clear
#define NODE_ADDR   0x64                                        // Primary I2C address of the slave node
#define currentLimit 240                                        //Motor current limit across 10R when moving to end positions
#define intergratedCurrentLimit  750                            //Three readings added over 60 ms from start of movement during
any movement(was 650)
#use i2c(Slave,Slow,sda=PIN_C4,scl=PIN_C3,address=NODE_ADDR)
BYTE slave_buffer[RX_BUF_LEN];

void ReadInput(void);
void i2c_interrupt_handler(void);
void MoveCtrl(void);
void PositionCTRL(void);
void ExtSetCtrl(void);
void InitPosition(void);
void ExtSetCtrlfg(void);

int1 ReadPosition_fg=0,stop_fg=0,end_fg=0,move_fg=0,readinputs_fg=0;  //Initalise flags
int1 RHlimit_fg,LHlimit_fg,ExtSetCtrl_fg=0,FirstRead_fg=0;
int1 direction,duty_cycle_fg=0;
int buffer_index,mode,SetPosition=1,position=2,read_position;
int16 OverCurrentDelay=10000;                                   //1000 equals 40ms/read (ie if set to 10,000, 5 reads in 2
seconds)
BYTE state;
unsigned int16 motorCurrent;
signed int Cn;

#int_RB
void RB_isr()                                                   //Read external switch position to set slider
{
  ExtSetCtrl();
  ExtSetCtrl_fg=1;                                              //Set flag
  FirstRead_fg=1;                                               //Delays the first position read to move off sensors
}
#INT_SSP
void ssp_interupt ()
{
int16 OverCurrentDelay_lsb;

i2c_interrupt_handler();

   if(state==5){                                                //Six bytes received address discarded
      mode=slave_buffer[0];
    switch(mode){
      case 0:                                                   //Set position
      SetPosition = (slave_buffer[1]);
      ReadPosition_fg=1;                                        //Set flags
      FirstRead_fg=1;
      stop_fg=0;
      break;
      case 1:
      OverCurrentDelay = (slave_buffer[1]);
      OverCurrentDelay_lsb = (slave_buffer[2]);
      OverCurrentDelay = (OverCurrentDelay<<8 |  OverCurrentDelay_lsb);
      break;
      }
   }
}
//-----------------------------------------------------------------------------
void ReadInput()
{
   switch(mode)
   {
      case 255:
      slave_buffer[buffer_index] = read_position;
      buffer_index++;
      break;
   }
}
//-----------------------------------------------------------------------------
void i2c_interrupt_handler(void)
{
BYTE incoming;
unsigned char tx_byte;

   state = i2c_isr_state();

   if(state < 0x80)                                             //Master is sending data
   {
      incoming = i2c_read();                                    //State 0 is address
```

```
   switch(state)
      case 1:                                                  //First data byte
         slave_buffer[(state-1)] = incoming;
         break;
      case  2:                                                 //Second data
         slave_buffer[(state-1)] = incoming;
           break;
      case  3:                                                 //Third data
         slave_buffer[(state-1)] = incoming;
           break;
      case  4:                                                 //Fourth data
         slave_buffer[(state-1)] = incoming;
           break;
      case  5:                                                 //Fifth data
         slave_buffer[(state-1)] = incoming;
           break;
      case  6:                                                 //Sixth data
         slave_buffer[(state-1)] = incoming;
           break;
      }
   }
   if(state == 0x80 )                                          //Master is requesting data
   {
      buffer_index = 0;                                        //Reset the buffer index
      ReadInput();                                             //Read bytes into buffer
      buffer_index = 0;                                        //Reset the buffer index
      tx_byte = slave_buffer[buffer_index];                    //Get byte from the buffer
                                                               //i2c_write(tx_byte);
#asm                                                           //Assembler for write command
         MOVF   tx_byte,W
         MOVWF  0x66
         MOVF   0x13, W
         MOVF   0x66, W
         MOVWF  0x13                                           //Set data into buffer
         nop                                                   //Delay before releasing clock
         nop                                                   //10 nop i.e. 2 us
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         BSF    0x14.4                                         //Release clock
         BCF    0x0C.3                                         //Clear SSP interrupt flag
TEST_BF:
         BSF    0x03.5                                         //Change to bank 1
         BTFSS  0x14.0                                         //Test BF bit
         GOTO   BF_OK
         BCF    0x03.5                                         //Change to bank 0
         GOTO   TEST_BF
BF_OK:
         CLRF   0x78
         BCF    0x03.5                                         //Change to bank 0
#endasm
      buffer_index++;                                          // increment the buffer index
      break;
   }
      if(state > 0x80 ){
         tx_byte = slave_buffer[buffer_index];                 // Get byte from the buffer
    //     i2c_write(tx_byte);                                          //Write next byte
#asm                                                           //Assembler for write command
         MOVF   tx_byte,W
         MOVWF  0x66
         MOVF   0x13,W
         MOVF   0x66,W
         MOVWF  0x13                                           //Set data into buffer
         nop                                                   //Delay before releasing clock
         nop                                                   //10 nop i.e. 2us
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         BSF    0x14.4                                         //Release clock
         BCF    0x0C.3                                         //Clear SSP interrupt flag
TEST_BF_1:
         BSF    0x03.5                                         //Change to bank 1
         BTFSS  0x14.0                                         //Test BF bit
         GOTO   BF_OK_1
         BCF    0x03.5                                         //Change to bank 0
         GOTO   TEST_BF_1
BF_OK_1:
         CLRF   0x78
         BCF    0x03.5                                         //Change to bank 0
  #endasm
         buffer_index++;                                       //increment the buffer index
         break;
      }
}
//*********************************************************************
void ExtSetCtrl()                                              //Set position to external switch setting
{
int B4,B5,tempSetPosition;

tempSetPosition=SetPosition;                                   //Save copy of previous value

   B4 = input (pin_B4);
   B5 = input (pin_B5)<<1;
   SetPosition = (B4 | B5 ) ;                                  //Inputs have internal pull-ups
```

```
   switch(SetPosition){
      case 3:
      SetPosition=2;
      break;
      case 2:
      SetPosition=3;
      break;
   }
      if(SetPosition >3 || SetPosition <1){                          //if outside permitted values
      SetPosition = tempSetPosition;                                 //Reset to previous value
      }
      else{
         ReadPosition_fg=1;                                          //Set flag to read
         stop_fg=0;
      }
}
//*********************************************************************
 void MoveCtrl()                                                     // Move control
{
int16 x;
int duty_cycle;

   Cn=SetPosition - position ;                                       //Wanted position - actual position

if(Cn==0 && stop_fg==1){
         output_bit(PIN_C1 ,0);                                      //Pin C1 low direction pin
         set_pwm1_duty (0);                                          //Brake motor
          x=64000;                                                   //Delay to allow to stop
         while(x>0){
            x--;
         }
         output_bit (PIN_C0 , 0);                                    //Disable motors
         ReadPosition_fg=0;                                          //Clear flags
         end_fg=0;
         duty_cycle_fg=0;
  }

if((Cn==0 ) && (stop_fg==0 ) && (direction==1)){                     //Position reached so do reposition
                                                                     //Drive further in -ve direction for short distance

         position=0;                                                 //To make slider overshoot position
         duty_cycle_fg=1;
         x=50000;                                                    //Drive a bit further in same direction
         while(x>0){
            x--;
         }
         stop_fg=1;                                                  //Set stop flag so will only do this once
      ////////////////////////////////// added code
         set_pwm1_duty (200);                                        //Brake motor
          x=32000;
         while(x>0){                                                 //Stop delay
            x--;
         }
  }
if((Cn==0) && (stop_fg==0) && (direction==0)){    //Position reached

         stop_fg=1;                                                  //Set stop flag
  }
if(Cn >0){
         direction=0;
         output_bit(PIN_C1 ,0);                                      //Pin C1 low direction pin
         duty_cycle=200;
         set_pwm1_duty (duty_cycle);
         output_bit (PIN_C0 , 1);                                    //Enable motors
  }
if(Cn <0){
         direction=1;
         output_bit(PIN_C1 ,1);                                      //Pin C1 high direction pin
         set_pwm1_duty (0);
         output_bit (PIN_C0 , 1);                                    //Enable motors
 }
}
//*********************************************************************
void checkuSwitch()
{
int16 x;
int C5,C6;

motorCurrent=0;

C5 = input (pin_C5);
C6 = input (pin_C6);

   if(end_fg==0){                                                    //Check so only do once
      if(C5==0 && C6 ==1){
         while(motorCurrent<currentLimit){
            direction=0;
            output_bit(PIN_C1 ,direction);                           //Pin C1 low direction pin
            set_pwm1_duty (200);
            output_bit (PIN_C0 , 1);                                 //Enable motors
            motorCurrent=read_adc();                                 //Read current on motor
            end_fg=1;
         }
         output_bit (PIN_C0 , 0);                                    //Disable motors
         position=3;
         LHlimit_fg=1;
      }
      if (C6 ==0 && C5==1){
         while(motorCurrent<currentLimit){
            direction=1;
            output_bit(PIN_C1 ,direction);                           //Pin C1 high direction pin
            set_pwm1_duty (0);
            output_bit (PIN_C0 , 1);                                 //Enable motors
            motorCurrent=read_adc();                                 //Read current on motor
            end_fg=1;
```

```
        }
        output_bit (PIN_C0 , 0);                                    //Disable motors
        position=1;
        RHlimit_fg=1;
      }
   }
    if(C6 ==1 && C5==1 && (RHlimit_fg==1 || LHlimit_fg==1)){
      if(LHlimit_fg==1){
           while(motorCurrent<currentLimit){
           direction=1;
           output_bit(PIN_C1 ,direction);                           //Pin C1 high direction pin
           set_pwm1_duty (0);
           output_bit (PIN_C0 , 1);                                 //Enable motors
           motorCurrent=read_adc();                                 //Read current on motor
           RHlimit_fg=1;                                            //Now at RH limit
        }
        output_bit (PIN_C0 , 0);                                    //Disable motors
      }
      if(RHlimit_fg==1){
           position=0;
        }
         x=64000;
        while(x>0){
              x--;
           }
        output_bit (PIN_C0 , 1);                                    //Enable motors
        ReadPosition_fg=1;                                          //Set flags to reposition
        stop_fg=0;
        RHlimit_fg=0;                                               //Clear flags
        LHlimit_fg=0;
    }
}
//*************************************************************************
 void PositionCtrl()    //Read 2 (3)bits to determine position RA1, RA2(and RA3)
 {
 int16 x;
 int A1,A2,temp_position;

   temp_position=position;                                          //Save previous value
if(FirstRead_fg==1){                                                //Delay first read
       x=64000;
        while(x>0){
              x--;
           }
      FirstRead_fg=0;                                               //Clear flag
}
   A2 = input (pin_A2)<<1;
   A1 = input (pin_A1);

   position = (A1 | A2 )^ 0x03;                                     //Inputs have pull-ups so invert
   read_position = position;

   if(position == 0){
      position = temp_position;
   }
}
 //*************************************************************************
 void InitPosition()                                                //Read 2 (3) bits to determine position RA1, RA2 (and RA3)
 {
int A1,A2;

   A2 = input (pin_A2)<<1;
   A1 = input (pin_A1);
   position=(A1 | A2 )^0x03;

   if(position==0){                                                 //If not at any position
      position=2;
      ReadPosition_fg=1;                                            //Set flag to read
      stop_fg=0;
   }
 }
//*************************************************************************
//Init() - routine
//*************************************************************************
void Init()
{
   setup_wdt (WDT_72MS);  //Setup WDT
   setup_ccp1(CCP_PWM);   //Set for PWM output
   setup_timer_2(T2_DIV_BY_1,49,1);                                 //Set period to 100kHz resolution 0-200
   port_b_pullups(TRUE);
   setup_adc_ports( AN0 );                                          //Select channel
   setup_adc(ADC_CLOCK_DIV_16);
   set_adc_channel(0);                                              //Set to read ADC channel 0
   input(PIN_C3);                                                   //Set I2C clock and data lines as inputs
   input(PIN_C4);
   output_bit (PIN_C0 , 0);                                         //Disable motors
   delay_ms( 1000 );                                                //Delay to let lines settle
   delay_ms( 1000 );
   InitPosition();                                                  //Get initial position
   enable_interrupts(INT_SSP);
   enable_interrupts(INT_RB);
}
//*************************************************************************
void checkFlags()
{
     if(ReadPosition_fg==1){
           MoveCtrl();                                              //Move slider
           PositionCtrl();                                          //Read current position
         if(move_fg==0){
             move_fg=1;
         }
     }
    if(ReadPosition_fg==0){
           move_fg=0;
    }
```

```
}
//*************************************************************************
void main() {
int16 tot_motorCurrent=0,tot_motorCurrent1=0,tot_motorCurrent2=0,tot_motorCurrent3=0,tot_motorCurrent4=0,y,x=0;

      Init();                                                    //Initialize 16F876A Microcontroller
      enable_interrupts(GLOBAL);
      SetPosition = 1
      ReadPosition_fg=1;                                         //Set flags
      FirstRead_fg=1;
      stop_fg=0;
while(1)                                                         //Loop Forever
   {
      restart_wdt();                                             //Restart watchdog timer

      if(ExtSetCtrl_fg==1){
            y=16000;                                             //Delay
         while(y>0){
               y--;
            }
             ExtSetCtrl();                                       //Get switch setting
             ExtSetCtrl_fg=0;                                    //Reset flag
         }
         checkFlags();
         checkuSwitch();                                         //Check u-switch state

   if(x==OverCurrentDelay){                                      //Do OverCurrentDelay cycles then read
          output_bit (PIN_C7 , 1);
         motorCurrent=read_adc();                                //Read current on motor
         tot_motorCurrent=motorCurrent;
         tot_motorCurrent4=tot_motorCurrent3;
         tot_motorCurrent3=tot_motorCurrent2;                    //Shift values
         tot_motorCurrent2=tot_motorCurrent1;
         tot_motorCurrent1=tot_motorCurrent;
         tot_motorCurrent=0;                                     //Zero total
         x=0;                                                    //Reset count
          output_bit (PIN_C7 , 0);
   }
    x++;
         if((tot_motorCurrent4+tot_motorCurrent3+tot_motorCurrent2)> intergratedCurrentLimit)
          {                                                      //Check if motor current to large
           position = SetPosition;                               //Make position and setpoint equal
            output_bit (PIN_C0 , 0);                             //Disable motors if motor current too large
           }
         checkuSwitch();                                         //Check u-switch state
         readinputs_fg=0;                                        //Clear flag
   }
}
```

6. Software

Although the unit may be used as a stand-alone device through the 3 position set-point switch on the control box (and a DC power supply), it is usually controlled through higher level software. We routinely use a National Instruments LabWindows CVI package, using C-code to write simple test programs which may also be integrated into larger programs. The test user interface is shown in Figure 8. The interface between computer and device is by means of USB devices from FTDI Ltd. Using a combination of FTDI-USB drivers and a separate interface board described elsewhere, I^2C communications can be made to the linear positioner.

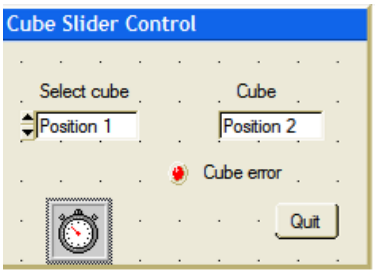


Figure 8. The user interface panel used for testing the cube positioner. A cube position is first selected; during the time that the motor is driving, the 'cube error' indicator is on, indicating a mismatch between required position and actual position. When the correct position is reached, the error light goes off and the cube position is indicated.

The test code shown below is used when testing the system through a host controller. It continually reads the position of the cubes and different cubes may be selected using RS232 commands which the FTDI-USB drivers convert to USB communications. The GCI_writeI2C_multiPort( ) and GCI_readI2C_multiPort( ) functions format the writing and reading commands so that the USB interface board can decode to I^2C commands.

```
//Program to test 3 cube positioner

#include "cvixml.h"                                              //CVI header files
#include <rs232.h>
#include <cvirte.h>
```

```
#include <userint.h>
#include "DeviceFinder.h"
#include <utility.h>
#include "cube slider_ui.h"
#include "usbconverter_v2.h"

#define CUBEPIC 0x64                                                    //Programmed I2C address of PIC
#define Round  RoundRealToNearestInteger                                //Redefine CVI function
#define bus 2                                                           //Set req'd bus (MPTR system) else set to 2 as default

static int PORT;
static int panelHandle;
static int mode;
static int setPosition=1;
static int timer;

static int initI2Cport(void);
static int sendPosition(int);
static int cubeSlider_initI2Cport(void);
static int cubeSlider_getFTDIport(int *port);
static int getFTDIport(int *PORT);

int main (int argc, char *argv[])
{
          if (InitCVIRTE (0, argv, 0) == 0)
                    return -1;          /* out of memory */
          if ((panelHandle = LoadPanel (0, "cube slider_ui.uir", PANEL)) < 0)
                    return -1;
          DisplayPanel (panelHandle);                                   //Display panel
          if(initI2Cport()==-1)return 0;                                //Initialise port  and set port number
          GCI_EnableLowLevelErrorReporting(1);
          SetCtrlAttribute (panelHandle, PANEL_TIMER, ATTR_ENABLED, 1);         //Enable timer
          RunUserInterface ();
          GCI_closeI2C_multiPort(PORT);
          DiscardPanel (panelHandle);
          return 0;
}
static int getFTDIport(int *PORT)
{
char path[MAX_PATHNAME_LEN],ID[20];
int id_panel,pnl,ctrl;

//If we are using an FTDI gizmo Device Finder will give us the port number
          GetProjectDir (path);
          strcat(path,"\\");
          strcat(path, "CubesliderID.txt");
          return selectPortForDevice(path, PORT, "Select Port for cube slider");
}
static int initI2Cport()      //Initialise port  and set port number
{
int  err,ans;
char port_string[10];

if (getFTDIport(&PORT) == 0)
          sprintf(port_string, "COM%d",PORT);
          else {                                                        //If device not found or not using FTDI or port error.

           while(getFTDIport(&PORT) != 0){                              //Keep testing until exit
          ans=ConfirmPopup ("Comms error", "Try plugging USB cable in or do you want to quit?");
          if(ans==1){   //quit
          return -1;
          }
            }
}
          sprintf(port_string, "COM%d",PORT);
          err = OpenComConfig (PORT, port_string, 9600, 0, 8, 1, 512, 512);     //Open port
          SetComTime (PORT, 1.0);                                       //Set port time-out to 1 sec
          FlushInQ (PORT);
          FlushOutQ (PORT);
          return 0;

}
static int sendPosition(int position)                                   //Send required position function
{
char val1[20];

          mode=0;                                                       //Set to position mode
          val1[0]=CUBEPIC;
          val1[1]=mode;
          val1[2]=position;
          GCI_writeI2C_multiPort(PORT,6, val1, bus);                    //Set new position
   return 0;
}
int CVICALLBACK cbsel_cube (int panel, int control, int event,          //Callback for position move
void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                    GetCtrlVal(panelHandle, PANEL_SEL_CUBE ,&setPosition);      //Get required position
                    sendPosition(setPosition);                          //Send cube position
                    break;
                    }
          return 0;
}
int CVICALLBACK cbquit (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                     QuitUserInterface (0);                             //Exit program
                    break;
                    }
          return 0;
```

```
}
int CVICALLBACK cbtimer (int panel, int control, int event,                    //Timer callback to read back position
                         void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int position;
          switch (event)
                    {
                    case EVENT_TIMER_TICK:
                    mode=255;                                                  //Set mode for reading
                    val1[0]=CUBEPIC;
                    val1[1]=mode;
                    GCI_writeI2C_multiPort(PORT,6, val1, bus);
                    val1[0]=(CUBEPIC | 0x01);                                  //Add 1 to address for I2C read
                    GCI_readI2C_multiPort(PORT,1, val1, bus);                  //Read cube position

                    position = val1[0] & 0xff;    //Returned position value

                    if(setPosition !=position){                                //If  mismatch  between  set  position  and  read  back
position
                              SetCtrlVal(panelHandle, PANEL_LED ,1);           //Turn on position error LED
                    }
                    else{
                             SetCtrlVal(panelHandle, PANEL_LED ,0);}           //If correct position turn off position error LED
                             SetCtrlVal(panelHandle, PANEL_CUBE_SET ,position);//Set cube position
                    break;
                    }
          return 0;
}
```

However, this self-contained code is of limited use when the system is integrated within a larger application, for example when we use the positioner as part of a microscope system. The approach used, shown in Figure 9, is to first define the optical characteristics of numerous cubes which may be used and select from the list of available cubes which will be the ones used. A rather simple GUI is then used during normal operation.

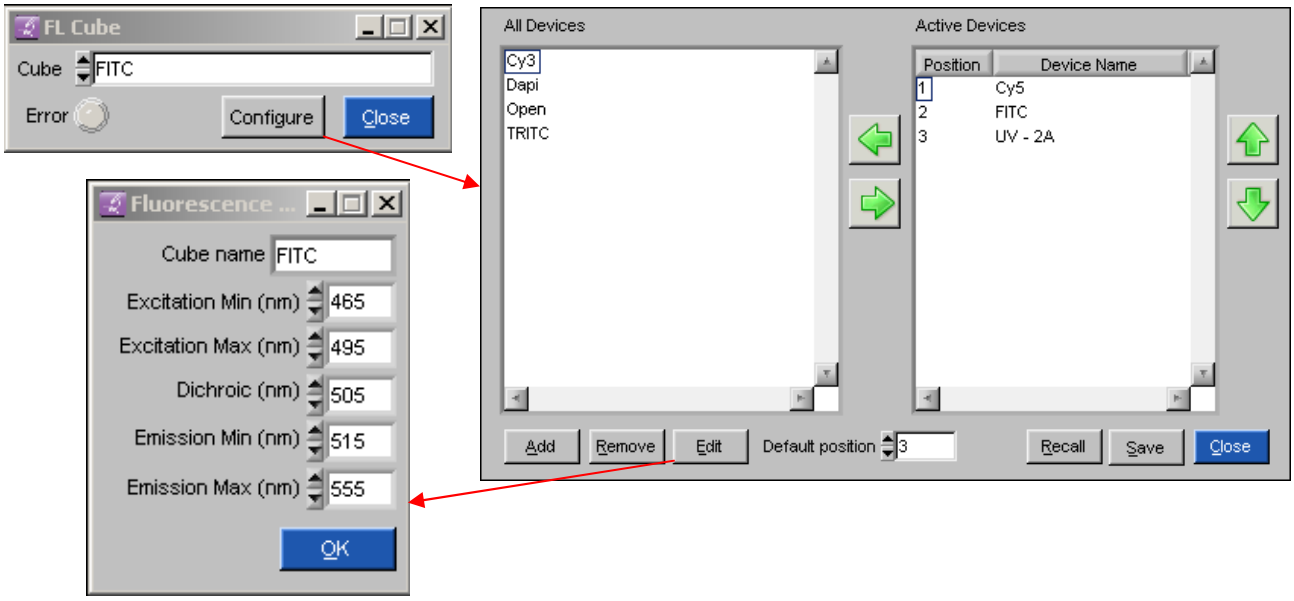


Figure 9. The main panel (top left) contains a fluorophore selector, an error indicator, and a 'Configure' button that opens up the device lists; in this instance, these allow the user to configure which cubes are present. Selecting one cube and clicking Edit allows the parameters stored (excitation, emission and dichroic reflector wavelengths) for the cube to be viewed and edited.

These wavelength details entered in the user panel shown in Figure 9 are used by the microscope to false-colour the images and in the metadata of each image. This 'mid' level code provides an application programming interface (API) so that higher level code, such a time lapse sequence with multiple cubes, can change the cube automatically without user intervention. Please refer to the Microscopy Software Architecture document for details of how the high level code is arranged in the microscopes.

This note was prepared by B. Vojnovic, PR Barber, IDC Tullis and RG Newman in July 2007 and updated in August 2011. Thanks are due to J. Prentice and G. Shortland for machining the various

components and to RG Newman for construction and testing. Detailed drawings of the various components are available on request (SolidWorks format), as are printed circuit board layouts (Number One Systems EasyPC (version 14 or below, http://www.numberone.com/).

We acknowledge the financial support of Cancer Research UK, the MRC and EPSRC.