A four position motorised optical path selector

1. Introduction

Microscopy systems often use a variety of detectors (e.g. monochrome and colour cameras) or indeed various imaging modalities (widefield, scanned beam etc.). In addition, illumination light sources may need to be selected and some form of beam switching turret is often desirable. We describe here a programmable device based on a direct microstepping motor drive system constructed around a Thorlabs (http://www.thorlabs.de) C4W cage system 'cube'. It is shown in Figure 1 and acts as a one pole, four way optical switch, capable of handling beam diameters in excess of 20 mm, limited by the Thorlabs SM1 tube system.

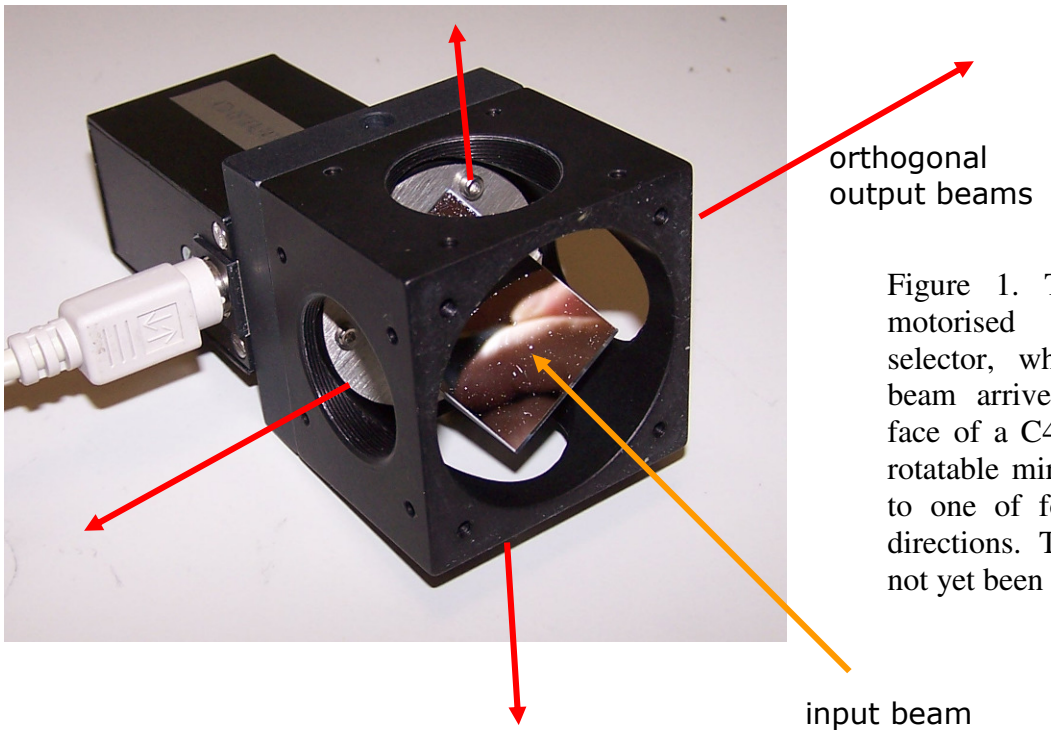


Figure 1. The 4-position motorised optical path selector, where the input beam arrives at the open face of a C4W cube and a rotatable mirror guides this to one of four orthogonal directions. The mirror has not yet been cleaned!

Although we chose to operate with four output directions, in principle the number of output ports can be increased, but a different mount would then be required. Depending on the mounting arrangements, all the ports may not be accessible; in fact in most of our applications only three ports are used as we found it convenient to mount the cube with an SM1 thread-topped mounting rod.

We use a rotating 45 degree 25 x 25 mm prism mirror: Thorlabs MRA25-G01, protected aluminum or MRA25-P01, protected silver units, though MRA25-E02 400-750 nm dielectric mirrors would be equally suitable; units available from Comar Instruments (http://www.comaroptics.com), 25RX03, have also been used and doubtless there are many suppliers of similar prism mirrors.

The main issue here is not the mirror, but the way it is mounted and the way it is turned. These processes are now explained.

2. Prism mirror mounting

It is clear that the mounting of the mirror is critical: neither tilt nor yaw is allowed during operation and the mirror must be at the correct height relative to the output ports of the C4W cube. However, since construction tolerances are inevitable, some form of adjustment is required. This need is met by using a 3-point kinematic mount of the round mirror baseplate, which can just be seen in Figure

1, close to the corner of the prism mirror. The prism mirror itself is glued onto this baseplate, against a machined edge which ensures lateral location. SolidWorks drawings are shown in Figure 2.

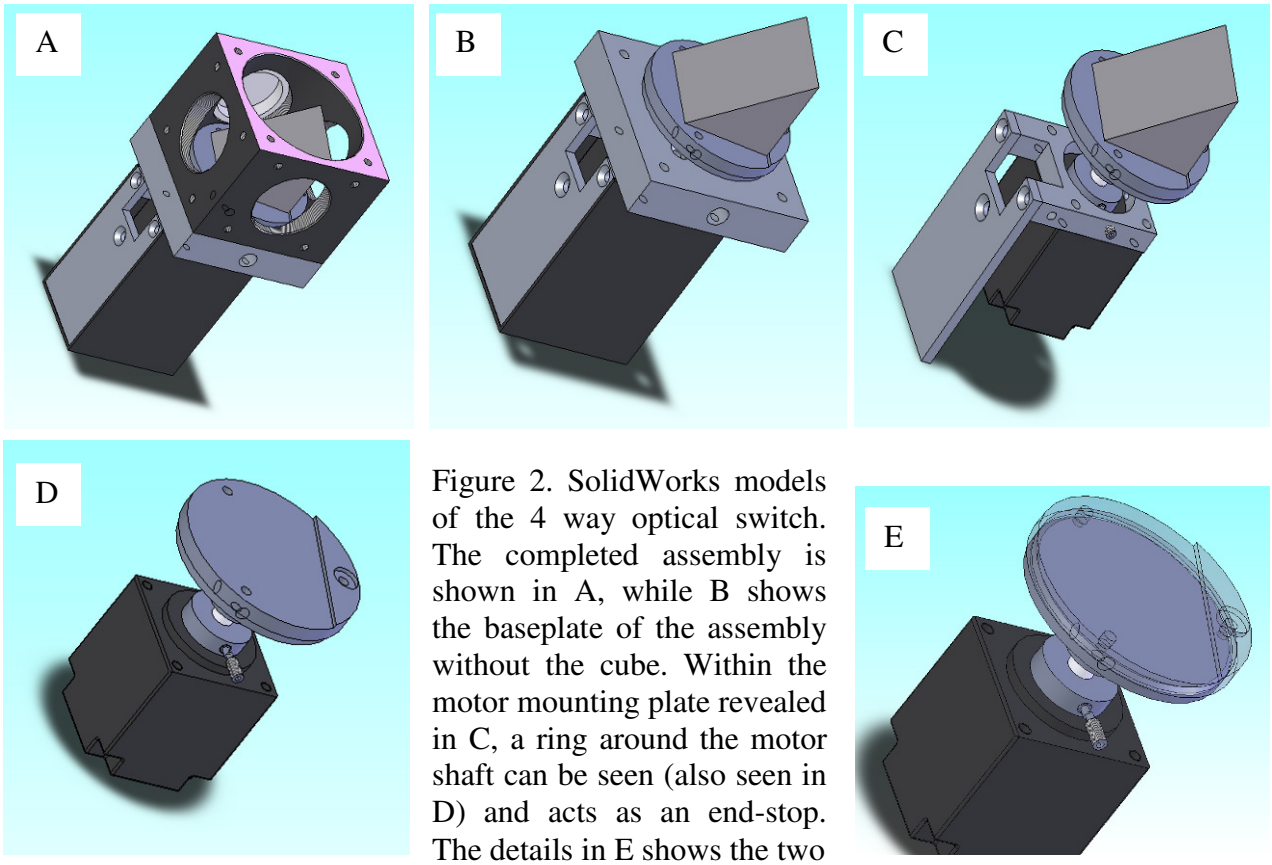


Figure 2. SolidWorks models of the 4 way optical switch. The completed assembly is shown in A, while B shows the baseplate of the assembly without the cube. Within the motor mounting plate revealed in C, a ring around the motor shaft can be seen (also seen in D) and acts as an end-stop. The details in E shows the two halves of the prism mirror mounting plates. The lower mounting plate fits snugly around the motor shaft, while the upper plate has a step machined in it to locate the prism mirror, The two plates are attached together with three screws and a domed spring plate in between. The spring plate is fashioned form a piece of spring steel or phosphor-bronze with small cutouts around the plate mounting holes. The upper plate is thus allowed to be set in a kinematic fashion so as to make the mirror prism height correct.

3. Prism mirror drive system

The drive to the assembly shown in Figure 2 is provided by a small stepper motor, arranged so that it may be driven in 1/16 microstep mode. The basic 200 step/revolution motor steps are enhanced to provide 3200 step/revolution, or just over 0.11 degrees per step. The microstep controller we used was produced by Lin Engineering (http://www.linengineering.com), type M2-211-13-01, ready mounted on a 42 mm stepper motor (Nema sixe 11). These however are no longer available, but if anyone is interested, we do keep a previously purchased stock of these items. A similar, though larger controller is currently available from Lin Engineering, type TMCM-110-42, suited to Nema frame size 17 motors.

In our case, we used a Nema frame size 11 stepper motor with an I^2C control board directly attached to the motor, sold as a MiniPak system by Lin Engineering. This system is based on a Trinamic TMC222-SI, SOIC20 chip (http://www.trinamic.com). A side plate is used to hold the connector carrying dc power and I^2C signals, as shown in Figure 3. A device for converting USB data to I2C data is described in one of our other application notes: “USB1 communications interface for controlling instruments”.

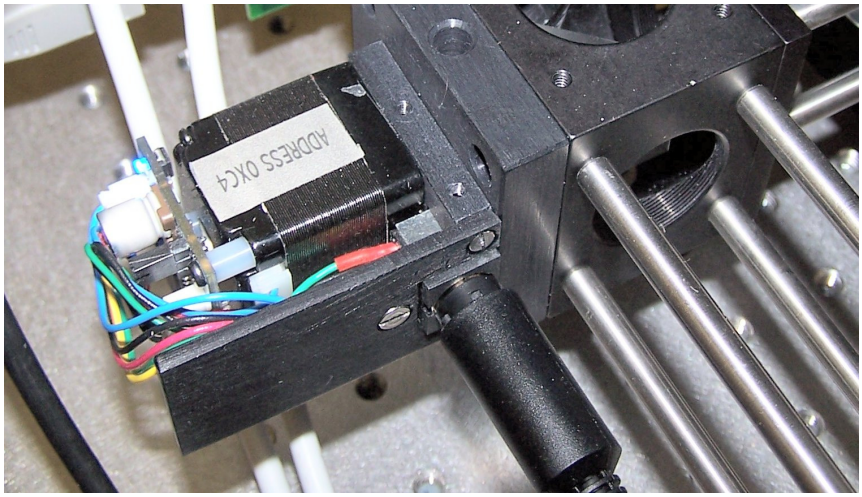
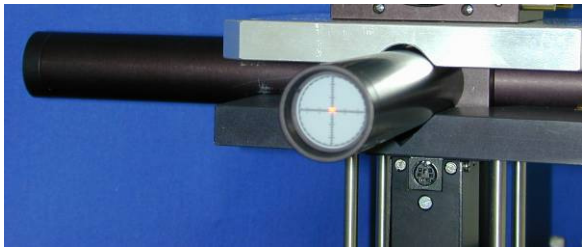


Figure 3. The drive system of the 4-position motorised optical path selector, showing the motor control board attached directly to the motor. A folded cover (not shown) encloses the assembly.

4. System alignment

There is no unique (nor easy!) way to align the system. Our approach is based on the fact that this system is most likely to be used with other Thorlabs components, tubes etc. and alignment is performed using the cube that will eventually be used in an instrument. Four long (150 mm) SM1 tubes are screwed into this cube and alignment screens are fitted to the ends of the tubes. The cube and these tubes are mounted between two plates with accurately machined 'V' grooves, so that the upper plate can be easily removed during alignment, and replaced in the same position. The motor in this arrangement is pointing upwards, with the 45 degree prism mirror and its mount also facing upwards such that a beam coming in from the top can be deflected through the centres of the tubes.



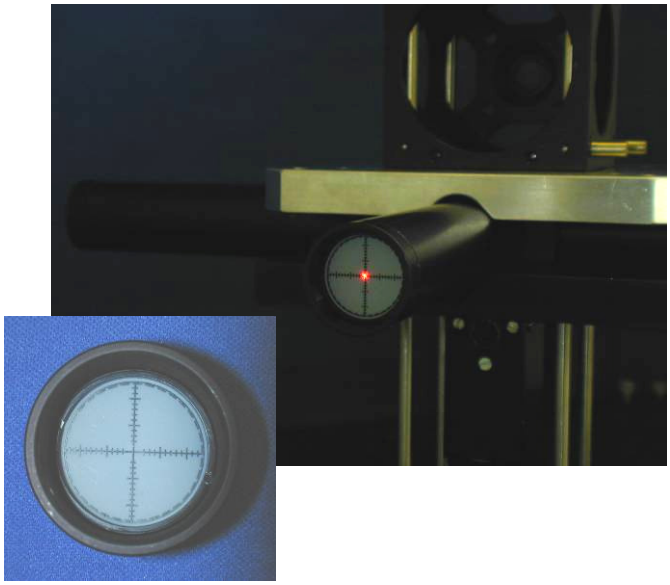


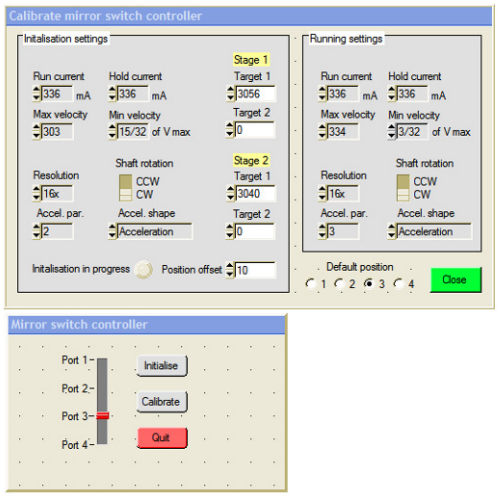
Figure 4. The alignment jig consists of a laser diode at the top of tower of Thorlabs components, including translation stages, apertures and cubes. These enable the laser beam to be centred onto the Thorlabs cube containing the mirror. The four output ports can be seen at the bottom. These may be extended for greater alignment accuracy. On the end of these tubes are placed cross-hair targets used to ensure centration of the beam into each of the four arms. The targets are housed in SM1L03 tubes, as shown on the inset.

A laser beam is then shone directly into the centre of the 45 degree prism mirror. This beam is produced with a small laser diode mounted on an X-Y translation stage, followed by a tip-tilt stage. This assembly is made up using a variety of opto-mechanical Thorlabs components as shown in Figure 4. The beam passes through a series of apertures (3 or more) so that it can be adjusted to be travelling parallel to the mounting rods and directly into the centre of the upper 'V' groove plate described earlier.

The adjustment procedures includes both mechanical and software calibration. The first essential requirement is to ensure that the height and the tilt angle of the 45 degree prism block is correct. This is accomplished by adjusting the three screws around the kinematic mount. Extension tubes are used to check beam 'height'. The next step ensures that the stepper motor rotation is appropriate for the port direction. We know that the number of motor steps between the four positions must be 800, since the motor drives at 3200 steps per revolution. The software makes use of this fact and allows us to set the offset away form the initialisation position. Ideally, the motor positions would be at 0, 800, 1600, 2400, i.e. the offsets would be at 0, 0, 0, 0 if the cube into which the assembly is installed were perfectly square and the motor construction were perfect. In practice, offsets of the order of a few counts are acceptable and indeed unavoidable. This adjustment procedure assumes that motor velocity, acceleration, running and holding currents have been previously set. These values were empirically set to provide fast switching with minimal overshoot of position.

5. Drive system software

We now list the high level code, developed under LabWindows CVI (http://www.ni.com/lwcvi/). This was used as the basic test code to allow flexibility in setting accelerations, speeds, target positions etc. and allows several parameters to be saved in the MiniPak module. The GUIs are shown in Figure 5 on the right.



The code should be 'read' in conjunction with the TMC222-SI chip datasheet and the MiniPak manual.

ROB-where is that? Can we have an e-copy or a scanned photocopy before it gets lost?

The code performs the following:

…INDICATE ROUTINE USED

Allows microstepping factor to be set SetMotorParam()

Allows direction of rotation to be set

Etc. etc. please carry on and describe with reference to end stop etc.

```
#include "cvixml.h"
#include <rs232.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "utility.h"
#include "formatio.h"
#include  <analysis.h>
#include "DeviceFinder.h"
#include <cvirte.h>
#include <userint.h>

#include "minipak mirror stepper_ui.h"
#include "minipak mirror stepper.h"
#include "password.h"
#include "IO_interface_v2.h"
#include "usbconverter_v2.h"
#include "DeviceManager.h"

#define Round  RoundRealToNearestInteger
#define address   0                                    //Programmable address of PIC
#define bus 2                                          //Set to 2 for usbconverter_v2.c
#define minipak     0xC0                               //Minipak I2C address
```

```
static int PORT;
static int panelHandle,calpanel,addresspanel;
static int init_holdcurrent,init_runcurrent,init_maxvelocity,init_minvelocity,init_acc,init_accshape;
static int init_shaft,init_resolution;
static int holdcurrent,runcurrent,maxvelocity,minvelocity,acc,accshape;
static int shaft,resolution,target_1,target_2,target_3,target_4,motion;
static int default_1,default_2,default_3,default_4,posoffset,initMirrorPos_fg=1;
static int answer=0,address1;
static char fname[MAX_PATHNAME_LEN];

static int initI2Cport(void);
static int getFTDIport(int *PORT);
static int RecallSettings(void);
int sendPosition(int);
int SetInitMotorParam(void);
int SetMotorParam(void);
int SetPosition(int position);
int GetParamVals(void);
int GetInitParamVals(void);
int runInit(int ,int );
int GetFullStatus(void);
int SetDefaultPosition(int position);
int DisableSwitches(int state);
int RestoreCalData(char *curfname);
int Reset(void);
int initAddress(void);

int main (int argc, char *argv[])
{
          if (InitCVIRTE (0, argv, 0) == 0)
                    return -1;          /* out of memory */
          if ((panelHandle = LoadPanel (0, "minipak stepper_ui.uir", PANEL)) < 0)
                    return -1;
          if ((calpanel = LoadPanel (0, "minipak stepper_ui.uir", CALPANEL)) < 0)
                    return -1;
          if ((addresspanel = LoadPanel (0, "minipak stepper_ui.uir", ADDRESSPNL)) < 0)
                    return -1;
          if(initI2Cport()==-1)return 0;                                                    //Set port number
          DisplayPanel (panelHandle);
          RecallSettings();
          GCI_EnableLowLevelErrorReporting(1);
          GCI_initMinipakStepper();
          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, "MirrorStepperID.txt");
          return selectPortForDevice(path, PORT, "Select Port for Minipak mirror stepper");


}
static int initI2Cport()
{
int  err,ans;
char port_string[10];

if (getFTDIport(&PORT) == 0)
                    sprintf(port_string, "COM%d",PORT);
          else {    //Device not found or not using FTDI. Use some other mechanism //such as Glenn's COM port allocation module.

                    while(getFTDIport(&PORT) != 0){

                              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);

          SetComTime (PORT, 1.0);                                                       //Set port timeout to 1 sec
          FlushInQ (PORT);
          FlushOutQ (PORT);
          return 0;

}
static int RecallSettings()
{
char *curfname1;

          curfname1="Minipak mirror1.txt";
          RestoreCalData(curfname1);

           return 0;
}
int GCI_initMinipakStepper()
{
char val1[20];
int x;
                              DisableSwitches(1);                                       //Disable switches
                              val1[0]=minipak;                                          //Address
```

```
                        val1[1]=0x81;                                           //GetFullStatus command

                        GCI_writeI2C_multiPort(PORT,2, val1, bus); //Needs this to activate motor
                        GetInitParamVals();                                     //Get initalisation parameter settings
                        SetInitMotorParam();

                        initMirrorPos_fg=0;                                     //Reset flag
                        if(default_1==1){                                       //Get mirror default position
                                SetCtrlVal(panelHandle, PANEL_MIRRORPOS ,1);
                        }
                        if(default_2==1){
                                SetCtrlVal(panelHandle, PANEL_MIRRORPOS ,2);
                        }
                        if(default_3==1){
                                SetCtrlVal(panelHandle, PANEL_MIRRORPOS ,3);
                        }
                        if(default_4==1){
                                SetCtrlVal(panelHandle, PANEL_MIRRORPOS ,4);
                        }

                        runInit(target_1, target_2);//Run init1
                        GetFullStatus();
                while(motion!=0){                                               //Allow time to move:wait until motion =0
                         GetFullStatus();
                         if (motion!=0)     {
                                SetCtrlVal(calpanel, CALPANEL_INIT,1);
                         }
                         else{
                         }
                         for(x=0;x<=20;x++){
                                   ProcessSystemEvents ();
                         }
                }
                 runInit(target_1, target_2); //Run init1
                 Delay(0.5);
                                GetFullStatus();
                while(motion!=0){                                               //Allow time to move: wait until motion = 0
                         GetFullStatus();
                         if (motion!=0)     {
                                SetCtrlVal(calpanel, CALPANEL_INIT,1);
                         }
                         else{

                                SetCtrlVal(calpanel, CALPANEL_INIT,0);
                         }
                         for(x=0;x<=10;x++){
                                   ProcessSystemEvents ();
                          }
                        }

                        GetParamVals();                                         //Get running parameter settings
                        SetMotorParam();
                        SetPosition(posoffset);                                 //Set target position 1
                        CallCtrlCallback (panelHandle, PANEL_MIRRORPOS, EVENT_COMMIT, 0, 0, NULL); //Set default position

                return 0;

}
int initAddress()
{

                GetCtrlVal(addresspanel, ADDRESSPNL_ADDRESS ,&address1);
                answer=ConfirmPopup ("Address", "Is this the correct I2C address");
                if(answer==1){
                        HidePanel (addresspanel);
                        DisplayPanel (panelHandle);
                        GCI_initMinipakStepper();
                        }

          return 0;
}
int runInit(int target_A,int target_B)
{
char val1[20];
int msb_target_A,lsb_target_A,msb_target_B,lsb_target_B;

                    msb_target_A=target_A>>8;
                    lsb_target_A=target_A & 0xff;
                    msb_target_B=target_B>>8;
                    lsb_target_B=target_B & 0xff;

                        val1[0]=minipak;                                                        //Address
                        val1[1]=0x88;                                                           //RunInit command
                        val1[2]=0xFF;
                        val1[3]=0xFF;
                        val1[4]=(init_maxvelocity<<4 |init_minvelocity);                        //Vmax/Vmin
                        val1[5]=msb_target_A;                                                   //Target position 1 (2 bytes)
                        val1[6]=lsb_target_A;
                        val1[7]=msb_target_B;                                                   //Target position 2 (2 bytes)
                        val1[8]=lsb_target_B;

                        GCI_writeI2C_multiPort(PORT,9, val1, bus);
return 0;
}
int GetInitParamVals()                                                                          //Initalisation settings
{
          GetCtrlVal(calpanel, CALPANEL_INITHOLDCURRENT ,&init_holdcurrent);
          GetCtrlVal(calpanel, CALPANEL_INITRUNCURRENT ,&init_runcurrent);
          GetCtrlVal(calpanel, CALPANEL_INITMAXVELOCITY ,&init_maxvelocity);
          GetCtrlVal(calpanel, CALPANEL_INITMINVELOCITY ,&init_minvelocity);
          GetCtrlVal(calpanel, CALPANEL_INITACC ,&init_acc);
          GetCtrlVal(calpanel, CALPANEL_INITSHAFT,&init_shaft);
          GetCtrlVal(calpanel, CALPANEL_INITRESOLUTION,&init_resolution);
          GetCtrlVal(calpanel, CALPANEL_INITACCSHAPE ,&init_accshape);
          GetCtrlVal(calpanel, CALPANEL_TARGET_1 ,&target_1);
```

```
          GetCtrlVal(calpanel, CALPANEL_TARGET_2 ,&target_2);
          GetCtrlVal(calpanel, CALPANEL_TARGET_3 ,&target_3);
          GetCtrlVal(calpanel, CALPANEL_TARGET_4 ,&target_4);
          GetCtrlVal(calpanel, CALPANEL_POSOFFSET ,&posoffset);
          GetCtrlVal(calpanel, CALPANEL_DEFAULT_1 ,&default_1);
          GetCtrlVal(calpanel, CALPANEL_DEFAULT_2 ,&default_2);
          GetCtrlVal(calpanel, CALPANEL_DEFAULT_3 ,&default_3);
          GetCtrlVal(calpanel, CALPANEL_DEFAULT_4 ,&default_4);
          GetCtrlVal(calpanel, CALPANEL_POSOFFSET ,&posoffset);

return 0;
}
int GetParamVals()               //Running settings
{
          GetCtrlVal(calpanel, CALPANEL_HOLDCURRENT ,&holdcurrent);
          GetCtrlVal(calpanel, CALPANEL_RUNCURRENT ,&runcurrent);
          GetCtrlVal(calpanel, CALPANEL_MAXVELOCITY ,&maxvelocity);
          GetCtrlVal(calpanel, CALPANEL_MINVELOCITY ,&minvelocity);
          GetCtrlVal(calpanel, CALPANEL_ACC ,&acc);
          GetCtrlVal(calpanel, CALPANEL_SHAFT,&shaft);
          GetCtrlVal(calpanel, CALPANEL_RESOLUTION,&resolution);
          GetCtrlVal(calpanel, CALPANEL_ACCSHAPE ,&accshape);
          GetCtrlVal(calpanel, CALPANEL_POSOFFSET ,&posoffset);

return 0;
}
int Set_Cal_Factors()
{
          SetCtrlVal(calpanel, CALPANEL_INITHOLDCURRENT ,init_holdcurrent);
          SetCtrlVal(calpanel, CALPANEL_INITRUNCURRENT ,init_runcurrent);
          SetCtrlVal(calpanel, CALPANEL_INITMAXVELOCITY ,init_maxvelocity);
          SetCtrlVal(calpanel, CALPANEL_INITMINVELOCITY ,init_minvelocity);
          SetCtrlVal(calpanel, CALPANEL_INITACC ,init_acc);
          SetCtrlVal(calpanel, CALPANEL_INITSHAFT,init_shaft);
          SetCtrlVal(calpanel, CALPANEL_INITRESOLUTION,init_resolution);
          SetCtrlVal(calpanel, CALPANEL_INITACCSHAPE ,init_accshape);
          SetCtrlVal(calpanel, CALPANEL_TARGET_1 ,target_1);
          SetCtrlVal(calpanel, CALPANEL_TARGET_2 ,target_2);
          SetCtrlVal(calpanel, CALPANEL_TARGET_3 ,target_3);
          SetCtrlVal(calpanel, CALPANEL_TARGET_4 ,target_4);
          SetCtrlVal(calpanel, CALPANEL_POSOFFSET ,posoffset);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_1 ,default_1);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_2 ,default_2);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_3 ,default_3);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_4 ,default_4);
          SetCtrlVal(calpanel, CALPANEL_HOLDCURRENT, holdcurrent);
          SetCtrlVal(calpanel, CALPANEL_RUNCURRENT, runcurrent);
          SetCtrlVal(calpanel, CALPANEL_MAXVELOCITY, maxvelocity);
          SetCtrlVal(calpanel, CALPANEL_MINVELOCITY, minvelocity);
          SetCtrlVal(calpanel, CALPANEL_ACC, acc);
          SetCtrlVal(calpanel, CALPANEL_SHAFT, shaft);
          SetCtrlVal(calpanel, CALPANEL_RESOLUTION, resolution);
          SetCtrlVal(calpanel, CALPANEL_ACCSHAPE, accshape);
 return 0;
}
int SetInitMotorParam()
{
char val1[10];


                              val1[0]=minipak;                                                    //Address
                              val1[1]=0x89;                                                       //Command for SetMotorParam
                              val1[2]=0xFF;
                              val1[3]=0xFF;
                              val1[4]=((init_runcurrent<<4) | init_holdcurrent);                  //Value of Irun and Ihold
                              val1[5]=((init_maxvelocity<<4) | init_minvelocity);                 //Value   of   Velocity_max   and
Velocity_min
                              val1[6]=(0<<5|(init_shaft<<4)|init_acc);                            //Acceleration shape/shaft rotation/secure
position top 3 bits
                              val1[7]=0;                                                          //Secure position bits 0 to 7
                              val1[8]=init_accshape<<4 | init_resolution<<2;                      //Acceleration shape /stepmode

                              GCI_writeI2C_multiPort(PORT,9, val1, bus);

return 0;

}
int SetMotorParam()
{
char val1[10];


                              val1[0]=minipak;                                                    //Address
                              val1[1]=0x89;                                                       //Command for SetMotorParam
                              val1[2]=0xFF;
                              val1[3]=0xFF;
                              val1[4]=((runcurrent<<4) | holdcurrent);                            //Value of Irun and Ihold
                              val1[5]=((maxvelocity<<4) | minvelocity);                           //Value   of   Velocity_max   and
Velocity_min
                              val1[6]=(0<<5|(shaft<<4)|acc);                                      //Acceleration    shape    /shaft
rotation/ secue position top 3 bits
                              val1[7]=0;                                                          //Secure position bits 0 to 7
                              val1[8]=accshape<<4 | resolution<<2;                                //Acceleration shape /stepmode

                              GCI_writeI2C_multiPort(PORT,9, val1, bus);

return 0;

}

int GetFullStatus()
{
char val1[10];
int slaveAddress;
int address1,ESW;
```

```
int IrunIhold,VmaxVmin,status1,status2,status3,na1,na2;

                              val1[0]=minipak;                                                  //Address
                              val1[1]=0x81;                                                     //GetFullStatus command
                              GCI_writeI2C_multiPort(PORT,2, val1, bus);
                              val1[0]=minipak| 0x01;

                    if (GCI_readI2C_multiPort(PORT,9, val1, bus)) return -1;                    //Read response
                              address1 = val1[0] & 0xff;
                              IrunIhold = val1[1] & 0xff;
                              VmaxVmin = val1[2] & 0xff;
                              status1= val1[3] & 0xff;
                              status2= val1[4] & 0xff;
                              status3= val1[5] & 0xff;
                              na1= val1[6] & 0xff;
                              na2= val1[7] & 0xff;

                              motion= status3 & 0xE0;                                           //Motion status
                              motion=motion>>5;

return 0;

}


int SetPosition(int position)
{
char val1[10];
int x;


                              if(position<0)
                              {
                                 position=(position | 0x80);  //To get +/- 32767
                              }
                              GetParamVals();                                                   //Get parameter settings
                              SetMotorParam();                                                  //Reset motor parameters

                              val1[0]=minipak;                                                  //Address
                              val1[1]=0x8B;                                                     //SetPosition command
                              val1[2]=0xFF;                                                     //
                              val1[3]=0xFF;                                                     //
                              val1[4]=position>>8;                                              //MSB position
                              val1[5]=position & 0xFF;                                          //LSBposition

                              GCI_writeI2C_multiPort(PORT,6, val1, bus);
                              while(motion!=0){                                         //Allow time to move:wait untill motion = 0
                               GetFullStatus();
                               if (motion!=0)     {
                                        SetCtrlVal(calpanel, CALPANEL_INIT,1);
                               }
                               else{
                                        SetCtrlVal(calpanel, CALPANEL_INIT,0);
                               }
                               for(x=0;x<=10;x++){
                                         ProcessSystemEvents ();
                                }
                              }
return 0;
}
int SetDefaultPosition(int position)
{

  switch(position)
  {
          case 1:
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_1 ,1);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_2 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_3 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_4 ,0);
          break;
          case 2:
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_1 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_2 ,1);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_3 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_4 ,0);
          break;
          case 3:
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_1 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_2 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_3 ,1);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_4 ,0);
          break;
          case 4:
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_1 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_2 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_3 ,0);
          SetCtrlVal(calpanel, CALPANEL_DEFAULT_4 ,1);
          break;

          }

return 0;
}
int DisableSwitches(int state)
{

          SetCtrlAttribute (panelHandle, PANEL_MIRRORPOS , ATTR_DIMMED, state);
          SetCtrlAttribute (panelHandle, PANEL_RUNINIT, ATTR_DIMMED, state);

return 0;

}
int Reset()
{
```

```
char val1[10];
                              val1[0]=minipak;                                          //Address
                              val1[1]=0x86;                                             //Reset position command

                              GCI_writeI2C_multiPort(PORT,2, val1, bus);

return 0;
}
void StoreCalData()                                                                     //Store calibration data
{
  char  shortname[MAX_FILENAME_LEN]="";
  char dataPath[MAX_PATHNAME_LEN] ;
  FILE *fp;
  static fpos_t ptr;

          GetInitParamVals();                                                           //Get calibration factors
          GetParamVals();
          fp = fopen (fname, "w");
          if (fp == NULL) return;
          SetWaitCursor (1);
          fgetpos (fp, &ptr);
                                                                                        //Note where start of file is

          fprintf(fp,
"%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\n"

          ,init_holdcurrent,init_runcurrent,init_maxvelocity,init_minvelocity,init_acc,init_accshape,init_shaft,init_resolution,holdc
urrent,runcurrent,maxvelocity,minvelocity,acc,accshape,shaft,resolution,target_1,target_2,target_3,target_4
          ,default_1,default_2,default_3,default_4,posoffset);
          fclose(fp);

          SetWaitCursor (0);

}
int RestoreCalData(char *curfname)
{
  char  dataPath[MAX_PATHNAME_LEN] ;

  FILE *fp;
          GetProjectDir (dataPath);
          strcat(dataPath,"\\Minipak mirror calibration\\");
          strcat(dataPath,curfname);
          strcat(fname,dataPath);
  fp = fopen (fname, "r");
  if (fp == NULL) return -1;

  SetWaitCursor (1);
  fscanf(fp,
"%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\t%i,\n"

          ,&init_holdcurrent,&init_runcurrent,&init_maxvelocity,&init_minvelocity,&init_acc,&init_accshape,&init_shaft,&init_resoluti
on,&holdcurrent,&runcurrent,&maxvelocity,&minvelocity,&acc,&accshape,&shaft,&resolution,&target_1,&target_2,&target_3,&target_4
          ,&default_1,&default_2,&default_3,&default_4,&posoffset);
  Set_Cal_Factors();                                                                    //Load calibration factors
  fclose(fp);
  SetWaitCursor (0);

  return (0);
}
//////////////////Callbacks/////////////////////////////////////////////
int CVICALLBACK cbquit (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              motion=0;
                              StoreCalData();                                           //Save to disc
                              QuitUserInterface (0);
                              break;
                    }
          return 0;
}

int CVICALLBACK cbresolution (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                           //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}


int CVICALLBACK cbaccshape (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                           //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}

int CVICALLBACK cbruncurrent (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
```

```
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                          //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}

int CVICALLBACK cbholdcurrent (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                          //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}

int CVICALLBACK cbmaxvelocity (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                          //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}

int CVICALLBACK cbminvelocity (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                          //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}

int CVICALLBACK cbshaft (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                          //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}

int CVICALLBACK cbacc (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              GetParamVals();                                          //Get parameter settings
                              SetMotorParam();
                              break;
                    }
          return 0;
}

int CVICALLBACK cbruninit (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];

          switch (event)
                    {
                    case EVENT_COMMIT:
                    GCI_initMinipakStepper();

                              break;
                    }
          return 0;
}


int CVICALLBACK cbmirrorpos (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
int mirrorpos,targetposition,x;

          switch (event)
                    {
                    case EVENT_COMMIT:
                              DisableSwitches(1);                    //Disable panel switches
                              GetCtrlVal(panelHandle, PANEL_MIRRORPOS ,&mirrorpos);
                              GetCtrlVal(calpanel, CALPANEL_POSOFFSET ,&posoffset);

                              GetInitParamVals();                                      //Get initalisation parameter settings
                              SetInitMotorParam();
                         runInit(target_3, target_4);        //Run init2
```

```
                    //Could put delay here for indicator
                    GetFullStatus();
                while(motion!=0){                                                   //Allow time to move:wait untill motion =0
                           GetFullStatus();
                           if (motion!=0)     {
                                   SetCtrlVal(calpanel, CALPANEL_INIT,1);
                           }
                           else{

                              //    SetCtrlVal(calpanel, CALPANEL_INIT,0);
                           }
                           for(x=0;x<=10;x++){
                                     ProcessSystemEvents ();
                           }

                }
                           targetposition=(posoffset+ ((mirrorpos-1)*(3200/4)));
                           GetParamVals();                                          //Get running parameter settings
                           SetMotorParam();
                           SetPosition(targetposition);                             //Set target position
                           DisableSwitches(0);                                      //Enable panel switches
                           SetCtrlVal(calpanel, CALPANEL_INIT,0);
                           break;
                }
        return 0;
}

int CVICALLBACK cbposoffset (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
int mirrorpos,posoffset,targetposition;

        switch (event)
                {
                case EVENT_COMMIT:
                           GetCtrlVal(panelHandle, PANEL_MIRRORPOS ,&mirrorpos);
                           GetCtrlVal(calpanel, CALPANEL_POSOFFSET ,&posoffset);
                           targetposition=(posoffset+ ((mirrorpos-1)*(3200/4)));
                           GetParamVals();                                          //Get running parameter settings
                           SetMotorParam();
                           SetPosition(targetposition);                             //Set target position
                           break;
                }
        return 0;
}

int CVICALLBACK cbcal (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                           //DisplayPanel (calpanel);
                            GCI_ShowPasswordProtectedPanel(calpanel);
                           break;
                }
        return 0;
}

int CVICALLBACK cbclose_cal (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                           HidePanel (calpanel);
                           break;
                }
        return 0;
}

int CVICALLBACK cbdefault_1 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                           SetDefaultPosition(1);
                           break;
                }
        return 0;
}

int CVICALLBACK cbdefault_2 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                           SetDefaultPosition(2);
                           break;
                }
        return 0;
}

int CVICALLBACK cbdefault_3 (int panel, int control, int event,
                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                           SetDefaultPosition(3);
                           break;
                }
        return 0;
```

```
}

int CVICALLBACK cbdefault_4 (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              SetDefaultPosition(4);
                              break;
                    }
          return 0;
}

int CVICALLBACK cbtarget_1 (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              break;
                    }
          return 0;
}

int CVICALLBACK cbtarget_2 (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              break;
                    }
          return 0;
}

int CVICALLBACK cbtarget_3 (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              break;
                    }
          return 0;
}

int CVICALLBACK cbtarget_4 (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              break;
                    }
          return 0;
}

int CVICALLBACK cbaddress (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:

                              break;
                    }
          return 0;
}

int CVICALLBACK cbaddressok (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                              initAddress();
                              break;
                    }
          return 0;
}
```

The above code allows stand-alone operation of the device and is used during testing and alignment. When this system is used as part of a more complex project (e.g. when incorporated into microscope system, the code and the user interfaces are modified somewhat and are shown in Figure 6. In this instance 3 positions are used for a laser scanning, monochrome and colour cameras. A small ‘main’ panel allows switching between these positions and settings and configuration panels are also accessible.

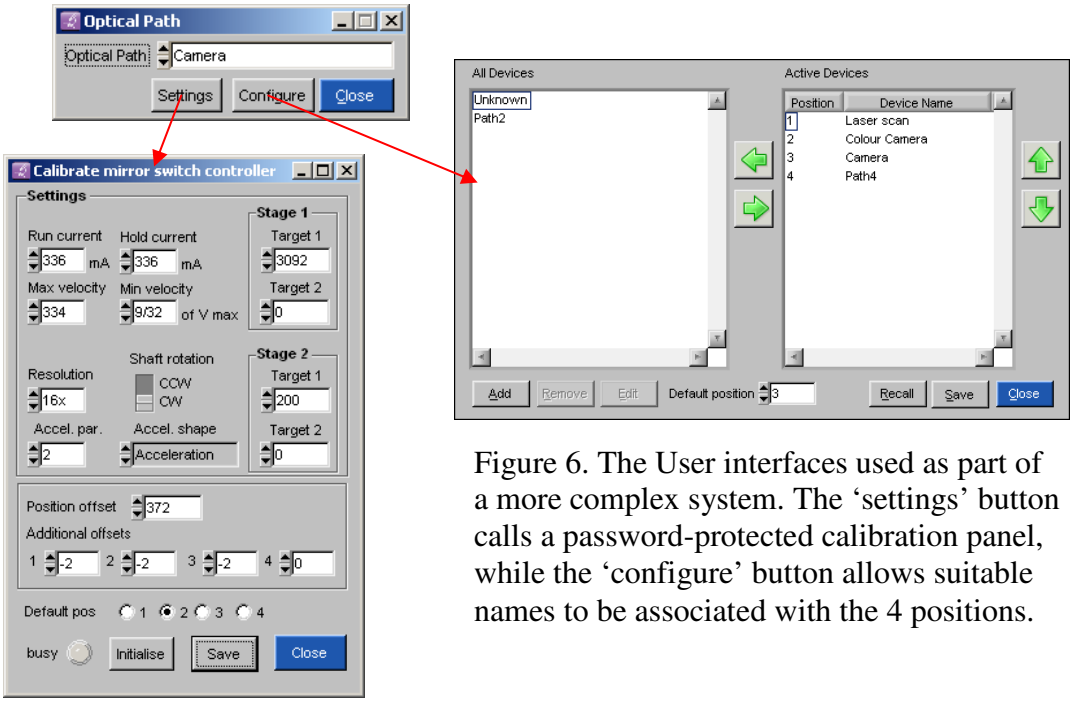


Figure 6. The User interfaces used as part of a more complex system. The 'settings' button calls a password-protected calibration panel, while the 'configure' button allows suitable names to be associated with the 4 positions.

6. Performance

Some error is always present and is inevitable with such a simple system. However the reproducibility of the positions is found to be near-perfect (less than 100 μradians) when approaching the target position from the same direction. Long-term stability has not really been determined accurately, but in practice we have not noticed any significant drifts when this system is used for imaging applications. The initialisation time takes 5 seconds and the time to move between positions is ~700 ms. At least ten units have been constructed to date and all perform reliably.

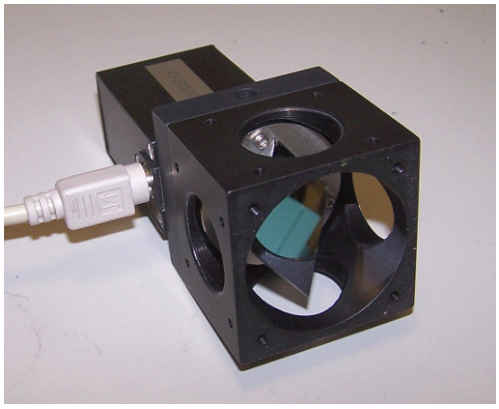


Figure 7. A view of the completed 4-position motorised optical path selector from the input side. Not too different from Figure 1, but at least the mirror is now clean!

This note was prepared by B. Vojnovic, PR Barber and RG Newman in August 2011. Thanks are due to J. Prentice and G. Shortland for machining the various components and to RG Newman and IDC Tullis for construction and testing. PR Barber contributed to software, which was developed by RG Newman and G. Pierce. Detailed SolidWorks drawings of the various components are available on request.

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