UK BPM Motion Control Experimenter's Guide

Bino Maiheu, University College London
bino@hep.ucl.ac.uk

Introduction

This document gives a brief description of the mover system installed in ESA to support Alexey's new RF BPM prototype (3BPM7). It describes the setup into some detail as well as the LabVIEW VI's and MINT motion control tools needed to operate the stages. For people just interested in how to control the stages, please skip to section 4.

1 The hardware setup

For mechanical stability a stacked configuration was used, with a vertical elevation stage (NEAT BAZ Vertical Elevation Stage) and a horizontal translation stage mounted on top of that (NEAT TM-200 Precision Grade). The **BAZ stage has a 1/20 wedge** which translates horizontal motion of the wedge into vertical motion of the top plate. The **lead screw** for both stages has a **pitch of 2 mm**. Both stages are operated by Baldor DSM hybrid stepper motors. These have an integrated stepper drive equipped with a micro-stepper of up to 51200 steps per revolution. Without micro stepping the motors have nominally **200 steps per revolution**.

Onto the stages, optical limit switches are mounted, which return a TTL level. Low when open, high when the end limits are reached. On the horizontal stage a piece of metal, which switches these optical limit is attached to the moving table. This restricts the **horizontal travel**

range to +/- 7.5 mm. On **the vertical stage**, the optical end switches restrict the travel range to the stage's total range of **+/- 2.5 mm** from the stage's nominal position. These optical switches are fed back to the controller via 9 Way D-Type connectors on the edge of both motor mounts. Also the signals coming from two linear encoders are found on these connectors. On the **horizontal stage** we have a Renishaw RGH22 Y50D00 linear encoder with a **resolution of 100 nm**. On the **vertical stage** there is a Renishaw RGH24 Z15F00A, **500 nm resolution encoder**.

For extra safe machine protection, there are **mechanical limit switches** which restrict the horizontal travel range of the TM-200 stage to **+/- 8 mm** from nominal position. Once switched, these switches directly interrupt the power to the horizontal stepper motor, bypassing the controller. Also a "normally high" TTL level monitor signal is sent back to one of the digital inputs on the controller so the state of the relay that switches the power can be seen in the motion control program. Note that once these mechanical switches are reached, a **severe malfunction** has occurred since the optical switches did not prevent the stages from moving further. This can be either due to the controller which controls the response to one of the optical limits being hit, or some hardware/cabling failure. At this point you will **need an access** and manually drive back the stages using the manual knobs. As a last protection measure, mechanical hard stops are in place, which physically inhibit the horizontal stages from going beyond **+/- 9 mm** from the stage's center. This way during normal ESA T-474 chicane operation the beam can never hit the walls of the BPM[1]. The relay is housed in a little relay box which is mounted to the top of the BAZ stage. This box has a number of sockets for the control cables running from the controller in rack 61.

There are 3 cables per axis. One cable connects to the D-Type connectors on the stages themselves. The other ones contain on the one hand the DC power lines to the motors (+48 VDC, +5VDC and the GND). This cable ends in a 3 Way XLR connector. And on the other hand there is a 6 Way XLR cable which contains the actual control signals (STEP, DIR, ENABLE), as well as the monitor signal that gets sent back to the controller. Also a DC level coming from an LM35DZ temperature sensor is housed in one of the control cables. The cables run underneath the concrete girder and end up directly at the location of Rack 61, where the controller is housed.

[1] The BPM prototype has an aperture of 30 mm.

2 The motion controller, wiring and hardware

At the core of the motion control is a **Baldor NextMove e100** motion controller. This controller was, together with two TRACO power supply units (48 VDC, 24 VDC), built into a 4U 19” rack mountable box, which is 400 mm deep. At the back the controller takes 230/115 VAC power and has it’s main fused to 10 A. Also fuses (2A) have been put to the 24/48 VDC lines as well as to a separate 24 VDC line (500 mA) which serves to power the digital inputs on the NextMove e100 controller. A little electronic circuit built around optical couplers translate the TTL logic signals sent to the controller to a 0/24 VDC logic which the controller understands. Both Ethernet ports of the e100, as well as the RS-232 and USB port have been patched through to the front panel on the motion control box. And connectors for the DC power, motor signals and encoder/limit switch signals are present for each axis. A number of BNC connectors are present as well which contain two +5 VDC and +24 VDC monitor levels, the DC level given back by the LM35DZ temperature sensor in the relay box mounted to the top of the BAZ stage and two analogue inputs which are fed to two 12 bit, +/- 10V ADC channels present on the e100 with max sampling frequency of 2 kHz.

3 MINT programming on the controller

The stages are controlled by a MINT program which is running on the FPGA of the e100. MINT is a programming language based upon Visual Basic with an extended set of motion commands to perform moves and read e.g. encoder positions. You can find the repository for the mint programs in

C:\UKBPM\MotionControl\MintCode\ukbpm_vX.mnt

on the UK DAQ computer in ESA (see further on). These MINT programs perform various tasks. They first of all initialize the controller by setting e.g. scalefactors for the encoder positions and the stepper motors, the polarity of the motors and the direction of the encoders to

create a well defined axis system, which digital inputs on the controller to use for which exact limit switch function, motion speeds etc.
Secondly, the program defines a set of events as well, the most important one is the limit switch error event, in which the controller's reaction to hitting a limit is programmed. Note that even though the optical limits provide hardware end switches, we have **programmed software limits** on to the controller which restrict the motion **between +/- 7mm** from the horizontal stage's nominal position. For the **vertical stage**, software limits between **+/- 2.45 mm** have been programmed.

The move routine is programmed as well as a semi-closed loop operation. Once a move has been requested, the controller will perform a move to the requested location, and then iterate at lower speed until the requested position is reached within 5 microns, or the number of iterations has exceeded 5.

Some safety features are also programmed in, such as the speed, which is restricted to 1 mm/s for the horizontal stage and 0.1 mm/s for the vertical stage. The MINT code which resides in the mentioned location is foreseen from many comment, so should be pretty straightforward to understand.

Communication between the outside world (the PC) and the MINT program is done via the so-called COMMS array. This is an array of floating point values which are constantly monitored by the controller. So the main loop of the program running on the controller is an infinite while loop that constantly monitors the COMMS(1) element. This element defines a command number which is passed to the controller. This value is then matched in a large if – then – else statement which selects the action to do. Specific actions make use then of various other COMMS() array values to pass parameters to the routines e.g. position to move to, speed, axis etc. Here is a table of the commands (see also the MINT program header comments).

COMMS(1) iCommand	**Description**	**Parameters**
1	Enable current axis	iAxis = COMMS(2)
2	Disable current axis	iAxis = COMMS(2)
3	Get the encoder position on current axis into COMMS(8)	iAxis = COMMS(2) COMMS(8) = ENCODER(iAxis)
4	Set the encoder position on current axis to value in COMMS(8)	iAxis = COMMS(2) ENCODER(iAxis) = COMMS(8)
5	Perform a relative move	iAxis = COMMS(2) fDistance = COMMS(3) [mm] fSpeed = COMMS(4) [mm/s]
6	Perform an absolute move	iAxis = COMMS(2)

		fPosition = COMMS(3) [mm] fSpeed = COMMS(4) [mm/s]
7	Execute the homing routine, centers the current axis inbetween the optical switches, resets encoder positions to 0 in the center and resets the software limits	iAxis = COMMS(2)

Next to the COMMS() elements used in the command set above, a number of other special elements are defined. These are :

- **COMMS(5)** : when the value in COMMS(5) changes, the Comms5 Asynchronous Event is executed. We have programmed this in our setup as an abort requested by the user. Normally the value for COMMS(5) is kept to 0, however if the user changes this motion is stopped on all or the current axis, depending on the exact value of this element. This intended to be used to interrupt motion during a move.
- **COMMS(10)** : we have programmed as some soft of action indicator, mainly to let the outside world know when a complicated move using the semi-closed loop has been successfully completed. Normally this element is 0. When a move is in progress, COMMS(10) will turn to 1 and when the move is completed, it will be set to 2. **Note that it is up to the user controlling the values in the COMMS array to reset the COMMS(10) value back to 0** to let the running program know that the user has acknowledged that the action has been completed.
- **COMMS(20) = 1 :** indicates the software on the controller to be running. It is a sort of software status indicator
- **COMMS(21) :** is a hardcoded version number of the program running on the MINT controller. It can be used to check whether the program communicating with the COMMS() array is talking to the right version of the MINT program.

4 Operating the mover system

For people that are just interested in how to control the stages, this will probably be the most interesting section. There are 2 LabVIEW front-end VI's which control the stages. Both of them live on the UK DAQ computer which is located downstairs. To gain access to it, grab it's desktop from one of the windows machines located in ChA. Execute "**Remote Desktop Connection**" for this under windows. The name of the PC downstairs is

ucl-ilc.slac.stanford.edu, IP-address: **134.79.64.44**

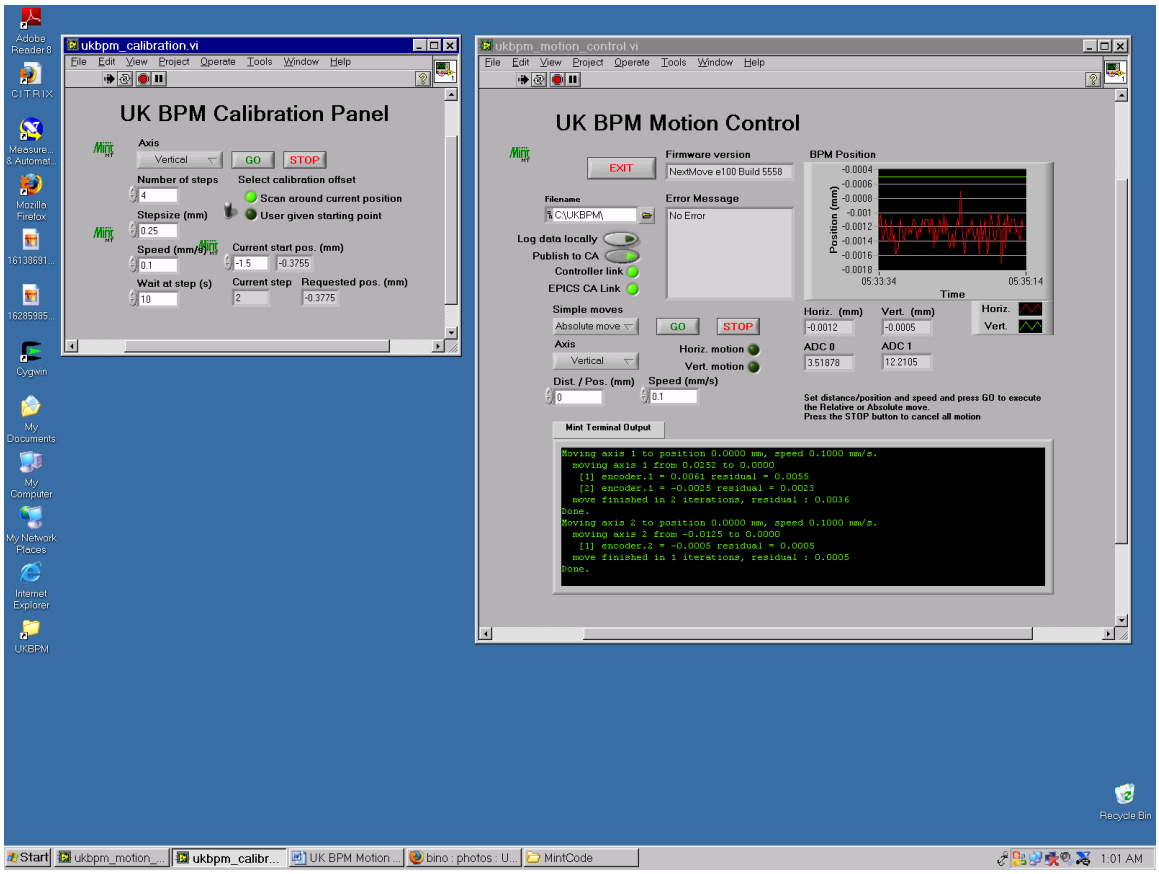
The motion control vi's are located in

C:\UKBPM\MotionControl

There are 2 shortcuts in this directory which fire up the controls :

UK BPM Motion Control.vi and **UK BPM Calibration.vi**

Start both of them. Let's have a look at what they do now. Below is a screenshot of how the vi's look like.



First of all let's have a closer look to the UK BPM Motion Control vi. This allows the user to select a number of simple moves, which correspond to the iCommand = 5,6,7 routines. So a relative move, an absolute move or homing of the selected axis. Select the axis from the list and enter the distance/position and the speed accordingly. Press "**GO**" to start the motion. Press "**STOP**" to cancel the motion. An LED will come on indicating horizontal or vertical motion. The graph on the right hand side constantly displays the encoder positions as do the numerical indicators below it. Also the values for the two ADC's present in the controller are shown.

The **EPICS CA Link** LED shows the status of the connection to Zen's main DAQ CA Server. The user can choose not to export the stages information back to the CA Server by clicking the "Publish to CA" control.
It is also possible to log the data locally to disk. By selecting a filename in the "**Filename**" dialog and toggling the "**Log data locally**" switch the encoder positions and the ADC values are written to disk locally.
The Motion Control vi also keeps a log file in

C:\UKBPM\MotionControl\Log\motion.log

For clean ending of the program, please use the "EXIT" button on the top left side of the vi. At the bottom there is the Mint Terminal Output window. This echoes the output from the mint program running on the controller.

The program constantly checks for a running program on the MINT controller with the correct version, when this is no longer the case, it will end informing the user of what went wrong.

The **UK BPM Calibration Panel** is nothing more than a front-end for some more complicated moves. This vi only has to be started when needed, but keep the UK BPM Motion Control vi running at all time, since this puts back the data into the main DAQ. To setup a calibration sequence, enter the necessary data in the controls on the vi. Set the **axis**, the **number of steps**, the **step size** for each calibration step, the speed and the **number of seconds to hold at each step** once a move has been completed to that particular step. You also need to specify a calibration offset. By toggling the switch, you can either choose to **scan *around* the stages current position.** Or, enter a **user defined starting position** for the scan and start from there. The numerical indicator next to the user input field for the starting position will always reflect the current starting position, whether set by the user, or computed from the calibration setup and the current position of the stages. Note that the starting position is really the lower starting postion for the scan and **not** a user defined value to scan **around**.
After each calibration the stages are set back to their original position.

Once you have setup the calibration, press "GO" to start it. Pressing "STOP" will have the calibration executing finish it's current step and afterwards return back to the original position.

A. APPENDIX - Changing the motor's holding current, micro steps etc...

As appendix, we’ll have a look at how to change the holding current for the motors and the number of micro steps.