Maccon Controller Encoder Values through Polling

Richard J Mathar, MPIA

December 4, 2000

Abstract

Some runs of motor motions monitored by polling of the `/mcon0` VxWorks channel encoder values of the Maccon board are shown. The polling frequency is restricted to only 10 Hz.

1 Motivation

The aim of this investigation is to establish means to look at motor motions as close as possibly supported by the encoder readings of the Maccon controller, hopefully bypassing some restrictions that might cause the Motor Engineering Interface to report slowlier.

2 Method

The following results were obtained by

1. Compiling the short C program (listed in the final pages) for VxWorks on the MIDI delay line LCU `lmidl`, which just loops as close as possible to read the encoder values of one Maccon channel (here: `/mcon0`) and store them in an array, maintaining also time stamps for each individual read. The reading is done through the VxWorks ESO device driver for this device.

2. Software initializing of a motor attached to the MIDI Translation Stage as currently available on the MIDI warm bench (with `motei`, the Motor Engineering Interface).

3. Detaching the motor from the Motor Engineering Interface and moving the motor with the ESO Handset attched to the front panel of the Maccon VMEbus board. This seems to be necessary to succeed with the `ioctl` call on the device, probably because `motei` opens the device in exclusive mode. This means monitoring the motor motion with this auxiliary program *and* moving the motors with `motei` at the same time does not (yet) work.

4. Running the program under the VxWorks shell with variable priority while moving the motor in different directions at different velocities as supported by the handset.

5. Looking at the two column ASCII table generated in the file with `gnuplot` for visualization *after* completion of the program, offline.

3 Results

As the figures of five runs show, the polling period is close to 0.1 seconds — rather slow, but faster than the period of 0.5 seconds selectable by the `moteiStatus` panel —, although

1. the VxWorks task was spawn up to rather high priorities (of 75)

2. and although we know that one can reach down with a very similar loop (used for the piezo control) to about 40 μs

3. the additional MIDI specific `miidevServer` and `miidevMonitor` processes were removed from the VxWorks task list

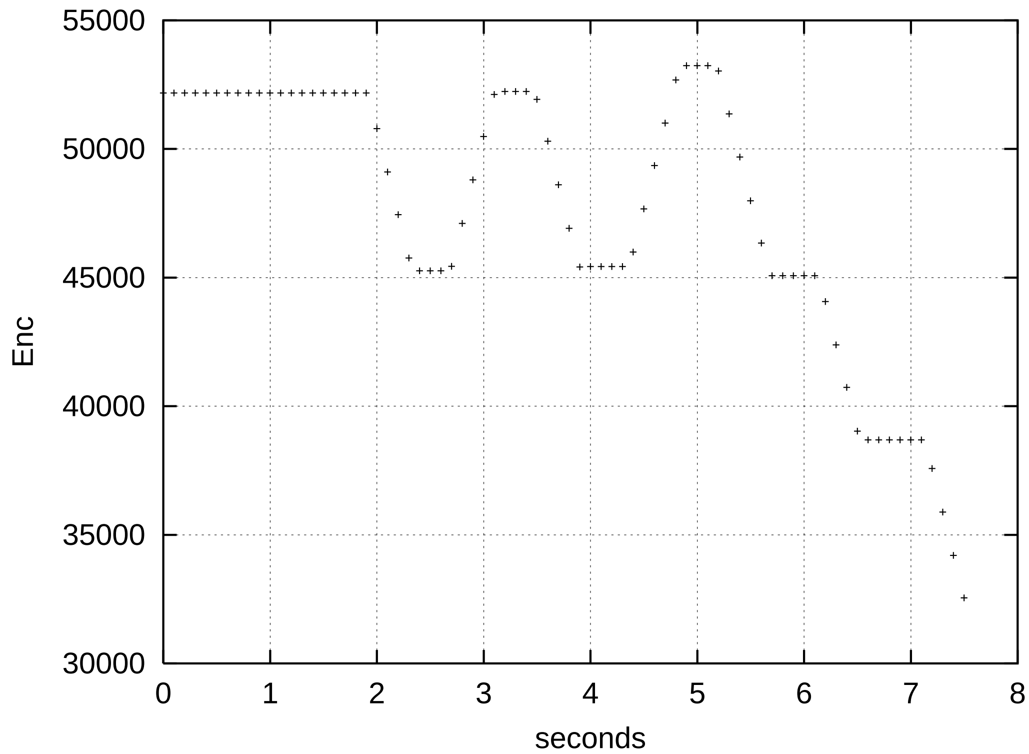

Figure 1:

4. artificial slowing down by looking at OLDB points (maybe scanned from the LCU environment to the workstation environment) does not exist

This means that this method is not valuable to detect high frequency jiggles of the motor movement or encoder glitches. A look at the actual device driver implementation in the ESO software does not reveal artificial timeouts or modifications of the VxWorks priority are responsible for the low update frequency achieved here; this might be a restriction specific to the . command queue of the controller itself.

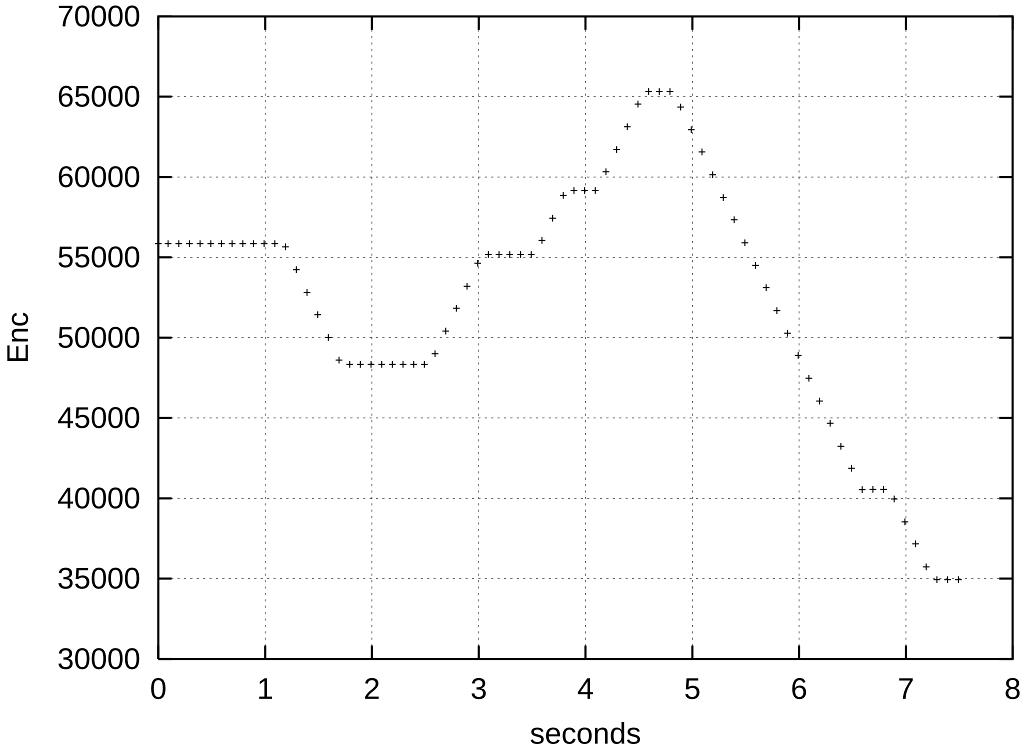

Figure 2:

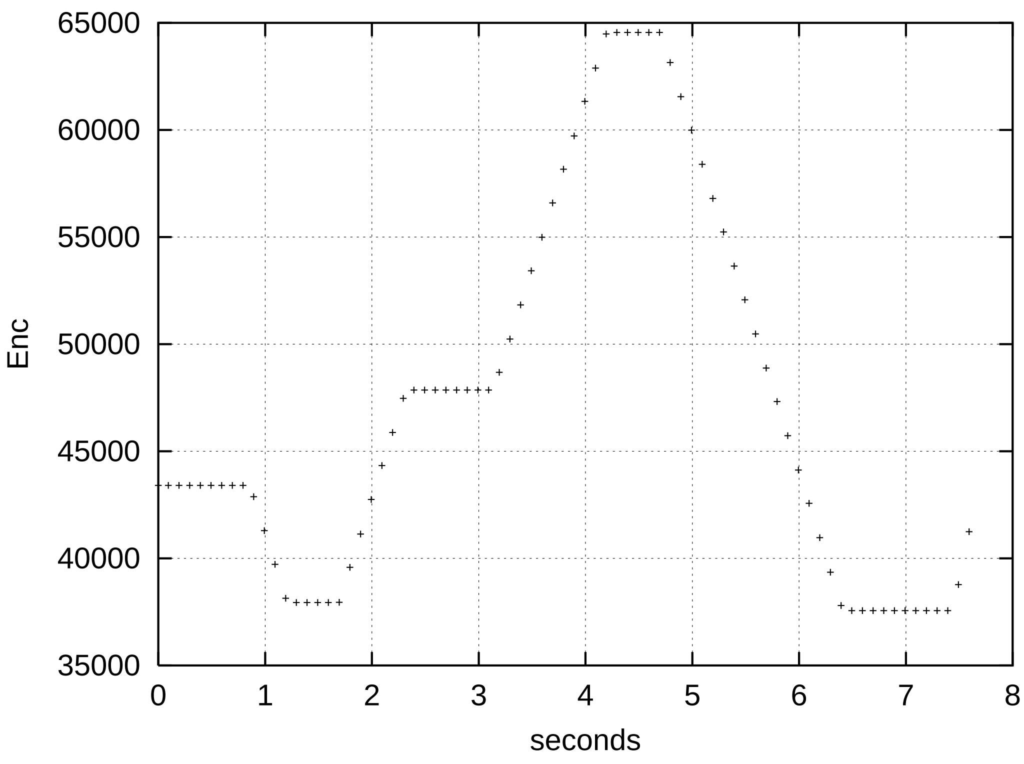

Figure 3:

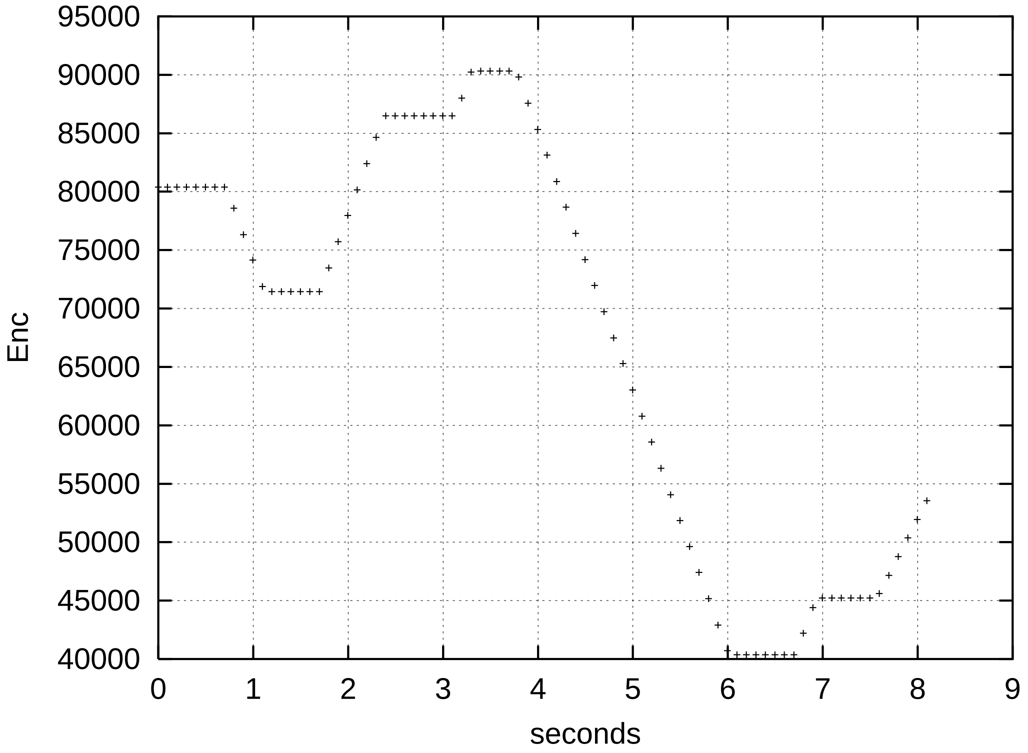

Figure 4:

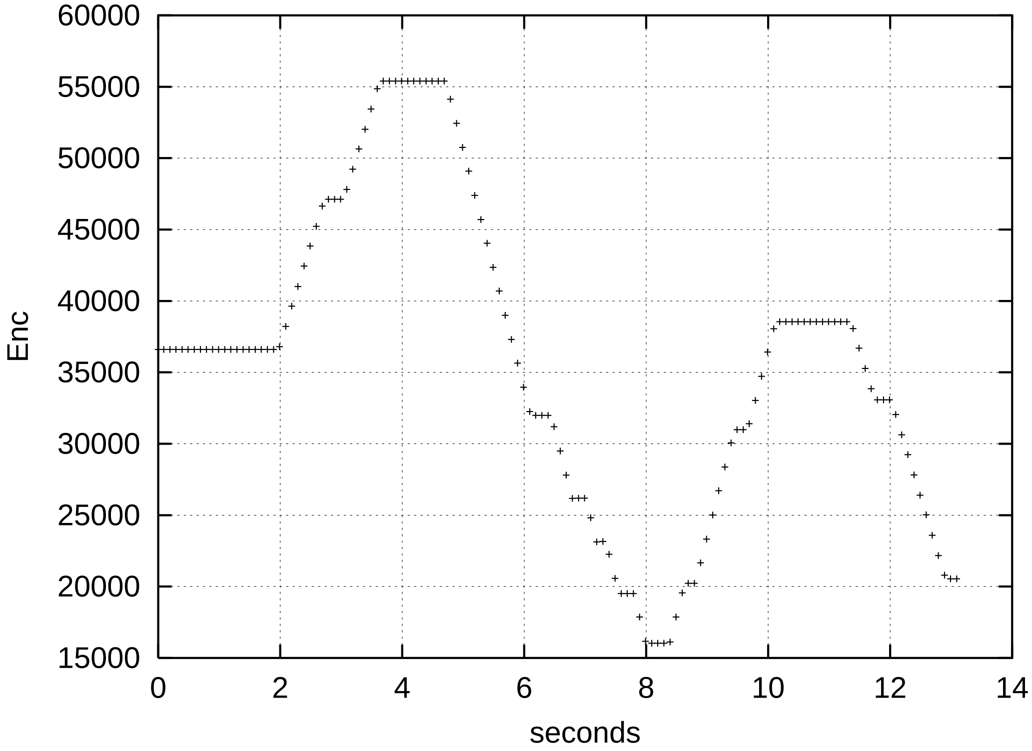

Figure 5: For all figures shown, 10000 Enc mean 2.44 mm translation of the piezo.

4 VxWorks Source Code

The following server process has been run on `lmidl`.

```
/*****************************************************
* MPIA - MIDI project
*
* "@(#) $Id: $"
*
* who       when      what
* --------  --------  ----------------------------------------------
* mathar    2000-11-05 created
*
* NAME
*
*  lmimcon.c - MIDI VxWorks task to monitor actual speed reported by mcon
*
* SYNOPSIS
*
*  int lmimcon(char * devName, char *resulfile, int duration)
*
*  parameters: <resulfile> UNIX file name of the optional voltage time
*                           report produced at task termination.
*              <duration> maximum run time of the program (seconds).
*                          Negative values mean that it only stops on receipt
*                          of the standard VxWorks signals.
*
* DESCRIPTION
*
* FILES
*
*  If resulfile is not "" (ie if it has string length larger than zero), the
*  ASCII file /vltdata/tmp/<resulfile> is created when the task finishes,
*  which reports the last RINGBUFF actual speeds of the /mcon-board and their time
*  stamps. RINGBUFF is defined at compile time.
*
* ENVIRONMENT
*
* RETURN VALUES
*
*  EXIT_FAILURE or EXIT_SUCCESS
*
* EXAMPLES
*
*  lmiics2-> taskspawn("lmimcon",100,0x8,30000,lmimcon,"/mcon0","lmimcon.1",20)
*      Monitors the motor speed for 20 seconds and stores the data
*      of the final interval of this time into the file /vltdata/tmp/lmimcon.1
*
*
* SEE ALSO
*
*  mconTestProc(1)
*
* CAUTIONS
*
*  This is an engineering interface. The task does not have a command
*  interface, and is not associated with an ICS or an OLDB.
*
*******************************************************/
#include "vxWorks.h"
#include "err.h"
```

```
                        /* see /disk-c/vlt/vw5.3/target/h/signal.h */
#include "signal.h"
#include "tim.h"

                        /* taskLib.h to please the compiler (else complaints implicit declaration of taskDelete, taskIdSelf */
#include "taskLib.h"
                        /* sysLib.h to avoid complaints about redefined sysClkGet(), because CLOCKS_PER_SEC is a macro */
#include "sysLib.h"

#include <stdio.h>
#include <stdlib.h>
#include <fioLib.h>
#include <ioLib.h>
#include <time.h>
#include <ccs.h>
#include "tims.h"
#include <string.h>
#include <types.h>
#include <math.h>
#include <mcon.h>

#include "miidev.h"

                        /* maximum number of bytes in the name of the result file */
#define LL 132
                        /* number of values in the ring buffer; occupies (1+ADCS+DACS)*RINGBUFF*8 bytes in the LCU memory,
                        and produces an ascii output file of about 15*RINGBUFF*(1+ADCS+DACS) bytes, but both only in the case it is
                        actually used, ie, in case of a 'resulfile' argument with a nonzero length of the char string. */
#define RINGBUFF 30000

                        /* variables that ought to be available in cleanup(), too, and therefore must be global */
static int mconopen;
static char lmiResulfile[LL] ;                  /* a global placeholder (the same as resulfile) */
                        /* the data to be stored; here:a ring buffer; and an index on the next position to be written */
double *ringbuff=NULL ;
int buff=0 ;
static int timopen ;
int lmimcon(char * devName, char *resulfile, int duration)
{
        int mcstatus ,timstatus;
        timsMODE timode ;
        ccsERROR ccserr ;
        double laps ;           /* time elapsed since start of the period */

        struct timUTC now,then ;
        static void cleanup() ;                 /* function called when the SIGTERM, SIGUSR1 or SIGUSR2 signal is received */
        sigset_t sigterm, oldsigs ;

                                        /* corresponding ccsExit() in cleanup() */
        if( ccsInit(taskName(0),0,NULL,NULL,&ccserr) != SUCCESS)
        {
                miidevLogData(miidevLOG_ID,"lmimcon(): Couldn't ccsInit\n") ;
                printf("lmimcon(): Couldn't ccsInit\n") ;
                errPrint(&ccserr) ;
                return EXIT_FAILURE ;
        }


                                        /* create a RAM buffer if 'resulfile' is set, else do not create output */
        strncpy(lmiResulfile,resulfile,LL-1) ;
        if( strlen(resulfile) )
```

```
{
        if( (ringbuff=calloc(RINGBUFF*2,sizeof(double))) == NULL)
                                /* sizeof() is type long on m68k, type int on ppc, so cast to int for compatibility */
                miidevLogData(miidevLOG_ID,"Couldn't allocate ring buffer, %d bytes\n",
                                RINGBUFF*2*(int)sizeof(double)) ;
}
else
        ringbuff= NULL ;

                                                /* could be lcudrvOPEN_EXCLUSIVE alternatively */
mconopen=open(devName,lcudrvOPEN_SHARED,(int)&mcstatus) ;
if ( mconopen <= 0 )
{
                        /* see VLTROOT/ERRORS/ */
        miidevLogData(miidevLOG_ID,"open on %s failed, status = %d\n",devName,mcstatus) ;
        return EXIT_FAILURE ;
}

                                /* if SIGTERM is send to the task (e.g. with kill(<TASKID>,15) , do some clean-up;
                                Stay POSIX compliant as demanded by the ESO Release Notes (as long as only signal() is
                                used, this is achieved anyway). All ioctl-commands on the aio driver but
                                aioCMD_GET_TIMEOUT are semaphore protected (see
                                ~vltmgr/VLTSW-REL1-FEB99/Drivers/aio/src/aioInitCmdTable.c), which means aioCMD_WRITE_VALUE
                                and aioCMD_READ_VALUE, too.  The corresponding lines in Drivers/aio/src/aioIoctl.c ensure
                                that the semaphore operations are included (so the corresponding lines in aioWRiteValue.c
                                and aioReadValue.c are not relevant and it's misleading to argue that they don't
                                set/take the semaphores. */
timopen=open("/tim0",lcudrvOPEN_READONLY,(int)&timstatus) ;


sigemptyset(&sigterm) ;
sigaddset(&sigterm,SIGTERM) ;
sigaddset(&sigterm,SIGUSR1) ;
sigaddset(&sigterm,SIGUSR2) ;
sigprocmask(SIG_UNBLOCK,NULL,&oldsigs) ;
signal(SIGTERM,cleanup) ;
                                /* ESO message system will send SIGUSR1 and SIGUSR2 signals instead of  kill and break */
signal(SIGUSR1,cleanup) ;
signal(SIGUSR2,cleanup) ;

timstatus=ioctl(timopen,timCMD_GET_TIME,(int)&then) ;
do {
        int av ;

                                        /* get time */
        timstatus=ioctl(timopen,timCMD_GET_TIME,(int)&now) ;
                                        /* and convert to seconds */
        laps=now.seconds-then.seconds+(now.microsec-then.microsec)/1.e6 ;

        if( ringbuff )
                ringbuff[buff++] = laps ;

        /*mcstatus=ioctl(mconopen,mconCMD_READ_VELOCITY,(int)&av) ;*/
        mcstatus=ioctl(mconopen,mconCMD_READ_CURRENT_POSITION,(int)&av) ;
        switch ( mcstatus )
        {
        case lcudrvOK :
                break ;
        default:
                mconErrorPrint(mcstatus) ;
        }
```

```
		/*printf("%d\n",av) ;*/
						/* read values of ADC channels if 'resulfile' is set; skip otherwise */
		if( ringbuff )
			ringbuff[buff++]=av ;
		buff %= RINGBUFF*2 ;
	} while (laps < duration || duration < 0 ) ;
	cleanup() ;
	return EXIT_SUCCESS ;
}
/************************************************************************
 To be called if either the main loop finishes or if a SIGTERM is received by the task;
 Move the values of the ring buffer to a file in /vltdata/tmp.
************************************************************************/
#define TIMSTRING 40
void cleanup()
{
	int mcstatus ;
	ccsERROR ccserr ;
				/* disconnect from VMEbus boards */
	mcstatus=close(mconopen) ;
	if( ringbuff )
	{
		FILE *asciifile ;		/* the formatted 'resulfile' */
				/* transform values from the ring buffer to the formatted resulfile */
		chdir("/vltdata/tmp") ;		/* returns OK or ERROR .. don't check for success ... */
		asciifile=fopen(lmiResulfile,"a") ;
		if( asciifile==NULL)
			miidevLogData(miidevLOG_ID,"Couldn't open %s for writing\n",lmiResulfile)
		else
		{
			int buffindx ;
/*#ifdef DEBUG*/
					/* to let time() work: execute -> tooSetLCUTime
					on the LCU at least once: see log0699 of 06/11/99 */
			time_t today;
			size_t timstring=TIMSTRING ;
			char todaybuf[TIMSTRING] ;

			time(&today) ;
			fprintf(asciifile,"# %s",ctime_r(&today,todaybuf,&timstring)) ;
/*#endif*/

			fprintf(asciifile,"# seconds      RAV value\n") ;
				/* do not assume that 'buff' points to the beginning of a block of 2 values, as the task
				may have received the signal SIGTERM at any time. So skip the current (maybe incomplete) block
				and set 'buff' to a value that is a multiple of 2 */
			buff += 2 - buff % 2 ;
			for( buffindx=0; buffindx < RINGBUFF-1 ; buffindx ++)
			{
				const start= (2*buffindx+buff)%(RINGBUFF*2) ;
				fprintf(asciifile,"%.7e %9.5f\n",ringbuff[start],ringbuff[start+1]) ;
			}
			fclose(asciifile) ;
		}
		free(ringbuff) ;
	}
	ccsExit(&ccserr) ;
	taskDelete(taskIdSelf()) ;
}
#undef TIMSTRING
#undef BUFFBYTES
```