# 2-Axis, Stepper Motor Plotter

6.115 Final Project Proposal, Spring 2008

Bhaskar Mookerji

23 April 2008

## Overview and Motivation

My motivation for this project comes from my desire to build a laser cutter for cutting acrylic and other plastics using vector and raster graphics. This can be assembled by using a combination of the appropriate optics (mirrors), a laser, and an x-y plotter. However, for practical purposes, I'm pairing this down to a pen x-y plotter with the capability of independently drawing pre-programmed vector graphics shapes (using integer-based drawing algorithms) from a user prompt or drawing vectorized images (stored in external memory or read from a vector graphics file over serial).

## Mechanical Hardware

The mechanical design goal is to have a stable plotting tip with two degrees of freedom, each accurately controlled with a six-wire stepper motor as used in Lab 4. There are several solutions for mechanically improvising the gantry system of a plotter:

- For a larger plotter, separate axes can be driven reversibly, in one-dimension, by the scanning head of a flatbed scanner (See Figure 1). Cheap scanners typically have a single unipolar stepper motor driving a plastic scanning head mounted on two rails spanning the width of the scanner. A steel rod securely mounted on the scanning head gives us one-dimensional control of the head. Orthogonally-placed scanners with corresponding steel rods can be joined by a scissor joint. The scissor joint would have a block with two roller bearings forming the joint, and on this we can mount the plotting head. The plotting head will be actuated in the z-axis by a DC solenoid.[1]
- If the first solution fails, a smaller plotter can be made by cannibalizing an Etch-a-Sketch. An Etch-a-Sketch already contains two independent axes controlled by knobs. Unipolar stepper motors can be fastened to each of these knobs using an aluminum coupling.

## Electrical Hardware and Software Design

A block diagram of the electrical hardware and software design for the plotter is in Figure 2. Electrical design in this lab will largely be limited to parts from Lab 4: the 8255 CMOS peripheral,

[1]The original inspiration for this solution comes from Instructables `http://www.instructables.com/id/SV7E00GFAQDX75Q/`. Elbow joint from `http://instruct1.cit.cornell.edu/Courses/ee476/FinalProjects/s2001/apk4/main.html`.

Figure 1: Aluminum rod on scanner head and elbow joint.

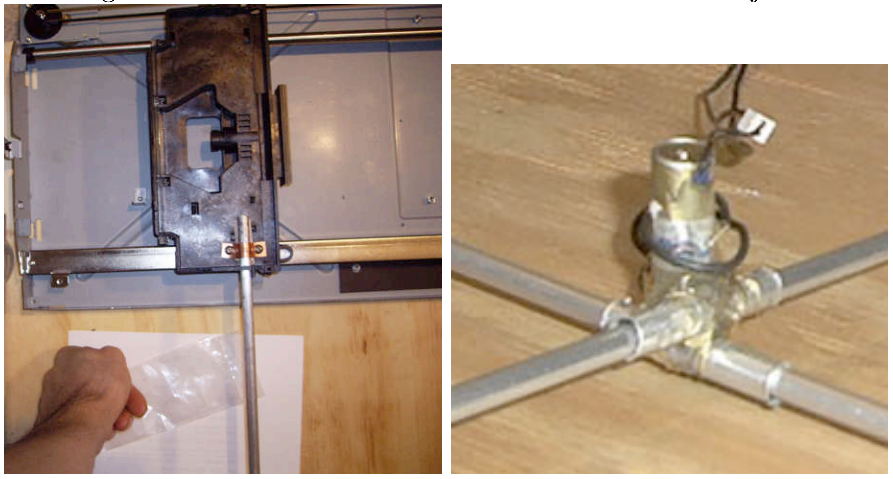
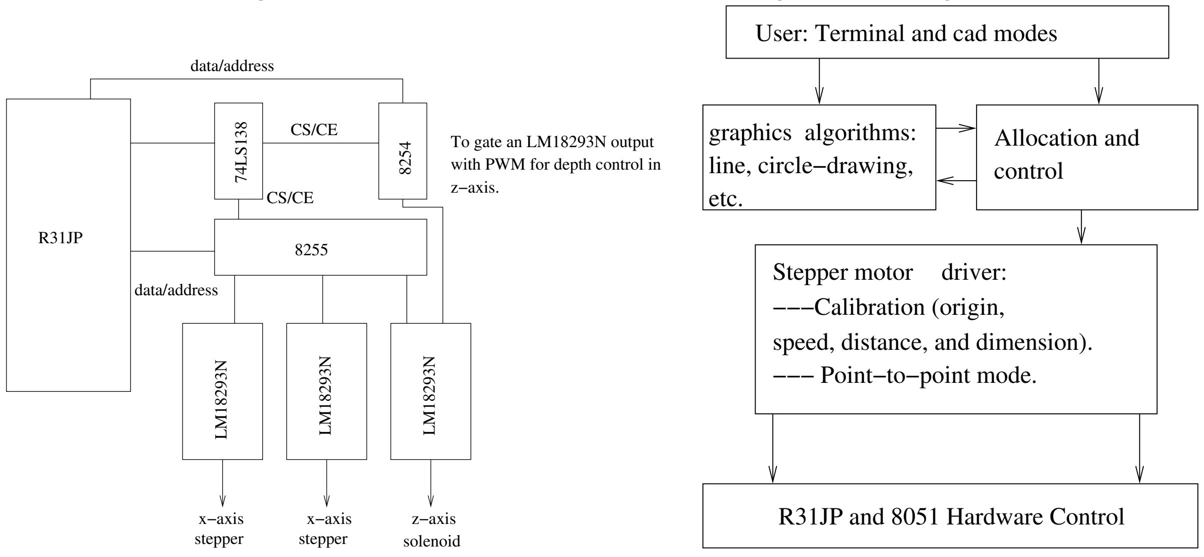


Figure 2: Hardware and software block diagrams of design.


data/address
R31JP
74LS138
CS/CE
8254
CS/CE
To gate an LM18293N output with PWM for depth control in z–axis.
8255
data/address
LM18293N
LM18293N
LM18293N
x–axis stepper
x–axis stepper
z–axis solenoid
User: Terminal and cad modes
graphics algorithms: line, circle–drawing, etc.
Allocation and control
Stepper motor driver:
–––Calibration (origin, speed, distance, and dimension).
––– Point–to–point mode.
R31JP and 8051 Hardware Control

LM18293N motor driver, and the 8254 timing chip. The 8254 will generate a PWM square wave to drive solenoid controlling the $z$-axis of a marker with varying applied forces. The 8255 will control the two stepper motors and the solenoid, as in Lab 4. Simple graphics processing will be done with the 8051 with the user selecting rendering options (terminal mode or run mode for graphics files) through Hyperterminal. The lowest software abstraction layer will be responsible for driving the LM18293N buffers between points (accurately) and calibrating the initial $(x, y)$-location of the plotter head. In the intermediate stage between user and hardware control, vector graphics algorithms will determine the current and future positions of the plotting head in the imaging plane.

The degree of sophistication here of electrical hardware and software will entail different amounts of risk. For a "wow" component, I'm contemplating using a serial/USB communication chip (not that horrible 16C450 Serial UART) to decode a USB drawing tablet, store the drawing in 8051 external memory, and then have the plotter render it. I could also write a computer script to send Postscript or Scalable Vector Graphics files (ultimately what I'd want to do for the laser cutter) to the 8051 and have them rendered with reasonable accuracy.

A regular completion of this project would probably entail a software implementation of a number of simple raster/vector graphics algorithms, and a scheme for having the 8051 read files for vectorized graphics on a PC and control the stepper motors. There are great examples of these algorithms online and in the literature, and remaining question concerns which algorithms are implemented. An example for a line between two points is Bresenham line algorithm, which uses only integer addition, subtraction and bit shifting for its implementation[2]:

```
function line(x0, x1, y0, y1)
    int deltax := x1 - x0
    int deltay := y1 - y0
    real error := 0
    real deltaerr := deltay / deltax
    int y := y0
    for x from x0 to x1
        plot(x,y)
        error := error + deltaerr
        if abs(error) >= 0.5 then
            y := y + 1
            error := error - 1.0
```

Note that such an algorithm is necessary due to the problem of synchronization between two stepper motors: it is not possible for the 8051 to drive two LM18293N buffers synchronously to generate a line of slope greater than or less than unity. This particularly algorithm composes a series of horizontal and vertical translations to draw an aliased line. Depending on the resolution of the stepper motors I use, this aliasing will not be a problem.

## Special Component Needs

As of this moment, I have one of the two needed scanners. If you guys have one laying around anywhere that you don't want, I'd love to have it. I haven't yet examined the specs on the scanner

[2] `http://en.wikipedia.org/wiki/Bresenham's_line_algorithm`

stepper motor that I currently have, however it seems USB powered, so it's probably 5V (I might replace it, if that's the case.). Swapfest is on April 20: I wouldn't mind getting money to buy an uber-cheap scanner bed.

**Timetable**

- April 14: Drive stepper motors with LM18293N, draw lines and circles of arbitrary dimensions.
- April 21: Start on vector/raster graphics drawing.
- April 28: Continue work on vector/raster graphics drawing.
- May 5: Work on USB pad and Postscript driver, if possible.
- May 12: Debug hardware for final presentation.

```
;; Started by Bhaskar Mookerji on spin-one-half.mit.edu
;; 6.115 Final Project
;; This program can start in either RUN mode.

;; Compiler directives for plotter parameters

;; Symbolic variables as memory locations.
state_x equ 30h ; current state of x motor
state_y equ 31h ; current state of y motor

x_pos_b equ 32h ; global x position variable (broad)
x_pos_f equ 33h ; global x position variable (fine)
y_pos_b equ 34h ; global y position variable (broad)
y_pos_f equ 35h ; global y position variable (fine)
init_pos equ 36h ; initial position

;; Constants
resolution equ #01h ; image scaling
steps_full equ #0C8h ; steps per revolution of motor
font_size equ #01h ; text scaling
height_b equ #01h
height_f equ #01h
width_b equ #01h
width_f equ #01h

;; Remember to add delays and repeats for scaling!

org 0000h
ljmp init

init:
lcall init_UART
lcall init_8255

mov state_x, #11h
mov state_y, #11h
mov init_pos, #00h
mov x_pos_b, init_pos
mov x_pos_f, init_pos
mov x_pos_b, init_pos
mov x_pos_f, init_pos


main:
lcall key_control
```

```
sjmp main

lcall getchr
mov dptr, #1400h
read_me:
movx a, @dptr
cjne a, #0AAh, continue
ljmp end_loop
continue:
mov R2, a
mov dptr, #dir_jumtab
lcall nway
inc dptr
ljmp read_me

;;  ljmp main
end_loop:
ljmp end_loop


key_control:
lcall getcmd
mov R2, a
mov dptr, #dir_jumtab   ;point dptr at beginning of jump table
lcall nway
ret

;================================================================
; subroutine nway
; this routine branches (jumps) to the appropriate monitor
; routine. the routine number is in r2
;=================================================================
nway:
mov   a, R2         ;load acc with monitor routine number
rl    a                   ;multiply by two.
inc   a                   ;load first vector onto stack
movc  a, @a+dptr          ;         "          "
push  acc                 ;         "          "
mov   a, R2               ;load acc with monitor routine number
rl    a                   ;multiply by two
movc  a, @a+dptr          ;load second vector onto stack
push  acc                 ;         "          "
ret                       ;jump to start of monitor routine

dir_jumtab:
```

```
dw main             ; command ’@’ 00
dw main             ; command ’a’ 01
dw main             ; command ’b’ 02
dw xy_se            ; command ’c’ 03
dw y_step_neg       ; command ’d’ 04 used
dw y_step_pos       ; command ’e’ 05 used
dw x_step_pos       ; command ’f’ 06 used
dw main             ; command ’g’ 07
dw main             ; command ’h’ 08
dw main             ; command ’i’ 09
dw main             ; command ’j’ 0a
dw main             ; command ’k’ 0b
dw main             ; command ’l’ 0c
dw main             ; command ’m’ 0d
dw main             ; command ’n’ 0e
dw main             ; command ’o’ 0f
dw main             ; command ’p’ 10
dw main             ; command ’q’ 11
dw xy_ne            ; command ’r’ 12
dw x_step_neg       ; command ’s’ 13 used
dw main       ; command ’t’ 14
dw main             ; command ’u’ 15
dw main             ; command ’v’ 16
dw xy_nw            ; command ’w’ 17
dw main             ; command ’x’ 18
dw main             ; command ’y’ 19
dw xy_sw            ; command ’z’ 1a

postscript_tab:
;;  dw main ; @
dw x_step_pos ; 0 East
dw xy_ne ; 1 Northeast
dw y_step_pos ; 2 North
dw xy_nw ; 3 Northwest
dw x_step_neg ; 4 West
dw xy_sw ; 5 Southwest
dw y_step_neg ; 6 South
dw xy_se ; 7 Southeast



;; subroutines x_step_pos, y_step_pos, x_step_neg, and x_step_neg
;; These control the Cartesian motion of the head, rotating the
;; accumulator left for positive motion and right for negative motion.
```

```
x_step_pos:
lcall crlf
mov dptr, #0FE1Ah
mov a, state_x
  movx @dptr, a
rr a
mov state_x, a
ret ; was lcall main (wtf?)

x_step_neg:
lcall crlf
mov dptr, #0FE1Ah
mov a, state_x
movx @dptr, a
rl a
mov state_x, a
ret

y_step_pos:
lcall crlf
mov dptr, #0FE18h
mov a, state_y
movx @dptr, a
rr a
mov state_y, a
ret

y_step_neg:
lcall crlf
mov dptr, #0FE18h
mov a, state_y
movx @dptr, a
rl a
mov state_y, a
ret

xy_ne:
lcall x_step_pos
lcall y_step_pos
ret

xy_nw:
lcall x_step_neg
lcall y_step_pos
ret
```

```
xy_sw:
lcall x_step_neg
lcall y_step_neg
ret

xy_se:
lcall x_step_pos
lcall y_step_neg
ret


;; subroutine init_8255
;; initializes the 8255 peripheral chip, connected to the motor drivers.
init_8255:
mov dptr, #0FE1Bh ; Set the control word of the 8255,
mov a, #80h ; for output on all ports.
movx @dptr, a
ret

;; sends an ASCII character in the accumulator over the serial
sndchr:
clr scon.1
mov sbuf, a
txloop:
jnb scon.1, txloop
ret

;==================================================================
; subroutine initUART
; this routine initializes the hardware
; set up serial port with a 11.0592 MHz crystal,
; use timer 1 for 9600 baud serial communications
;==================================================================
init_UART:
mov   tmod, #20h       ; set timer 1 for auto reload - mode 2
mov   tcon, #41h       ; run counter 1 and set edge trig ints
mov   th1,  #0FDh      ; set 9600 baud with xtal=11.059mhz
mov   scon, #50h       ; set serial control reg for 8 bit data
      ; and mode 1
ret

;================================================================
; subroutine getchr
; this routine reads in a chr from the serial port and saves it
; in the accumulator.
;================================================================
getchr:
jnb  ri, getchr        ; wait till character received
mov  a,  sbuf          ; get character
anl  a,  #7fh          ; mask off 8th bit
clr  ri                ; clear serial status bit
ret



;================================================================
; subroutine getcmd
; this routine gets the command line.  currently only a
; single-letter command is read - all command line parameters
; must be parsed by the individual routines.
;
;================================================================
getcmd:
lcall getchr           ; get the single-letter command
clr   acc.5            ; make upper case
lcall sndchr           ; echo command
clr   C                ; clear the carry flag
subb  a, #'@'          ; convert to command number
jnc   cmdok1           ; letter command must be above '@'
lcall main
cmdok1:
push  acc              ; save command number
subb  a, #1Bh          ; command number must be 1Ah or less
jc    cmdok2
lcall main           ; no need to pop acc since badpar
; initializes the system
cmdok2:
pop   acc              ; recall command number
ret


;================================================================
; subroutine crlf
; crlf sends a carriage return line feed out the serial port
;================================================================
crlf:
mov   a,  #0ah         ; print lf
lcall sndchr
cret:
```

```
mov   a,  #0dh         ; print cr
lcall sndchr
ret

;; ==============================================================
;;  subroutine print
;;  print takes the string immediately following the call and
;;  sends it out the serial port.  the string must be terminated
;;  with a null. this routine will ret to the instruction
;;  immediately following the string.
;; ==============================================================
print:
pop   dph ;  put return address in dptr
pop   dpl
prtstr:
clr  a ;  set offset = 0
movc a,  @a+dptr ;  get chr from code memory
cjne a,  #0h, mchrok ;  if termination chr, then return
sjmp prtdone
mchrok:
lcall sndchr ;  send character
inc   dptr ;  point at next character
sjmp  prtstr ;  loop till end of string
prtdone:
mov   a,  #01h ;  point to instruction after string
jmp   @a+dptr ;  return

;; Letter mappings, generic font.
letter_A:
db 00
letter_B:
db 00
letter_C:
db 00
letter_D:
db 00
letter_E:
db 00
letter_F:
db 00
letter_G:
db 00
letter_H:
db 00
letter_I:
```

```
db 00
letter_J:
db 00
letter_K:
db 00
letter_L:
db 00
letter_M:
db 00
letter_N:
db 00
letter_O:
db 00
letter_P:
db 00
letter_Q:
db 00
letter_R:
db 00
letter_S:
db 00
letter_T:
db 00
letter_U:
db 00
letter_V:
db 00
letter_W:
db 00
letter_X:
db 00
letter_Y:
db 00
letter_Z:
db 00
```