MotionCode™

Step-by-step solutions for motion applications

Galil MotionCode Solution:

Computer-Controlled Microarrayer

Contents **Page**

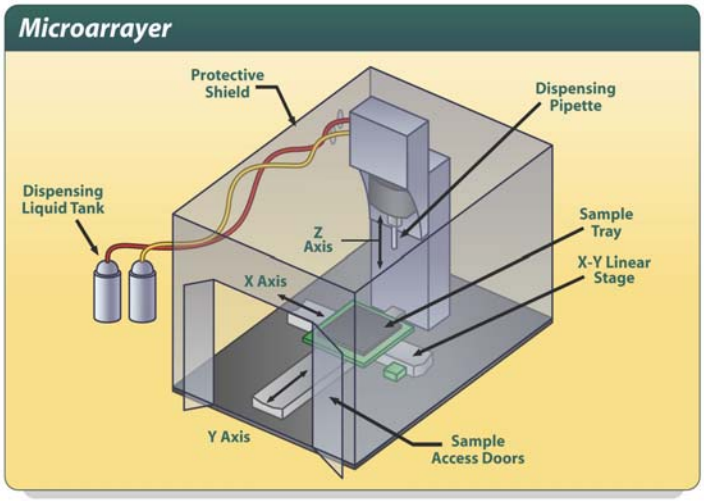

Figure 1.

1. Machine Description

The field of biotechnology has developed a need for machines that are designed to dispense small, accurate, contamination-free samples onto a media tray for further analysis. These machines are commonly called micro-arrayers *(figure 1)*. In a microarraying application, point-to-point motion is repeated over a grid to dispense liquid onto an array of locations. When the tray is complete, the machine waits for the tray to be removed and a new tray to be inserted and the arraying process repeats.

2. Requirements

This section summarizes the requirements for the machine described above:

(1) Two servo axes for XY tray positioning
(2) One servo axis for pipette extension
(3) Index pulse, forward and reverse limit switches for every axis
(4) One digital input for a door open sensor
(5) One digital output for the dispense valve
(6) Machine must dispense an 11 x 11 sample pattern of fixed size
(7) No user interface is required

3. Components Selected

This section describes the Galil hardware and software products chosen to implement the machine's control system. Below is a complete bill of materials followed by a description of major components.

Table 1. Bill of Materials for Microarrayer Control System

Part Number	Description	Unit Price (U.S.) Qty 1 / qty 100
DMC-2133	3-axis Ethernet, RS232 controller card with 96-pin DIN connector	$1045/$725
-DIN	DIN-rail mount option	$100/$50
-DC24	DC-to-DC converter option for 18V to 36V	$100/$70
AMP-20540	Amplifier Board with four 500 W PWM drives for brush or brushless servos	$795/ $495
BLM-N23-50-1000 or equivalent	NEMA 23 Brushless Servo Motor, 1000 ppr encoder with Hall Sensors	Consult mfg. x3
CPS-12-24 or equivalent	24 V, 200 Watt Power Supply	Consult mfg.
WSDK Servo Tuning Software	Servo Tuning and Analysis Software	$195 (one time)

Controller: DMC-2133

Since the microarrayer does not require a host PC, we choose the DMC-2133 stand-alone motion controller. A laptop may be connected via Ethernet or RS-232 to perform servo tuning with the WSDK and to load the application program.

Figure 2. DMC-2133 Motion Controller

Motor: BLM-N23-50-1000

For maintenance-free operation, we choose brushless motors. Galil's NEMA 23 #BLM-N23-50-1000 brushless motors, or equivalent, are appropriate because all axes require less than 0.3 Nm of continuous torque. Incremental encoders with 1000 cycles per revolution are installed on the motors resulting in 4000 quadrature counts per revolution. Hall sensors are not required on the motors as the incremental encoders provide commutation tracks for input to the amplifiers.

Amplifier: AMP-540

To drive the three motors, we choose the very compact AMP-20540, which is a four-axis brushless amplifier (500 W per axis) that directly mounts to the top of the controller.

Figure 3. DMC-2133-DC24-DIN with attached AMP-20540

4. Implementation

This section details how the components selected above were used to implement the control system.

Program Flowchart

The flowchart in *figure 4* shows how the motion program works. The three axes are homed immediately upon power up and are not homed again until the next power up. The #AUTO label is used to designate code that runs upon controller power up. After homing is complete, the program waits until the door sensor indicates that the operator has opened the door, removed the old sample tray, inserted a new sample tray, and closed the door. After the door is closed, the dispense cycle begins. The XY stage moves the sample tray to the dispense location, the dispensing pipette is extended, the sample is dispensed, and finally the pipette is retracted. This process is repeated until samples have been placed on all XY locations. Once all XY locations have been populated, the program loops back to wait for the operator to open the door and the entire process is repeated for the next sample tray.

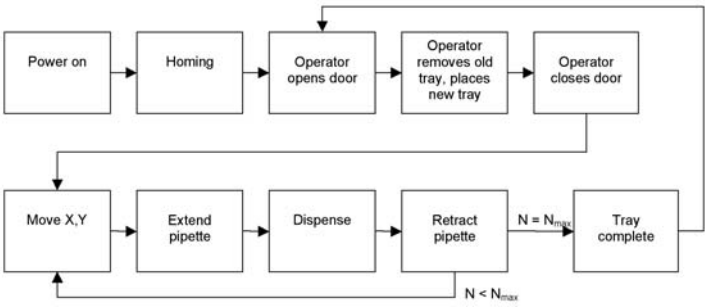

Figure 4. Microarrayer Program Flowchart

Homing

This section details the homing routine used for all three axes. Each axis has a forward limit switch, a reverse limit switch, and an index channel on the encoder. A separate home switch is not necessary to home the axes. The homing routine is split into three sections *(figure 5)*:

(1) A search for the reverse limit switch
(2) A slow search for the index pulse in the positive direction
(3) An offset command (DP) to define the 0 position

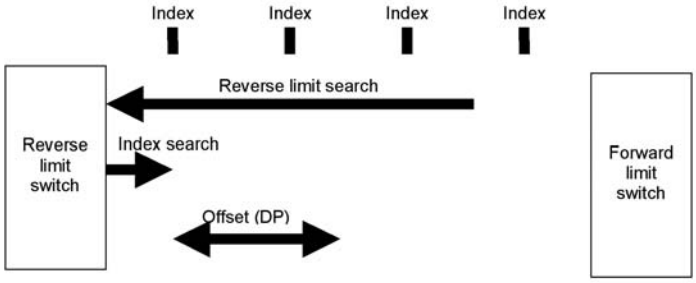

Figure 5. Homing routine for all three axes

The reverse limit switch search may be performed at a high speed because the accuracy of it's position is not critical. The axis is sent off indefinitely in the negative direction until the reverse limit switch changes state. When this happens, the axis decelerates to a controlled stop. Next, the axis moves in the forward direction towards the index pulse which is closest to the reverse limit switch. This search must be done slowly (500 counts/second) to ensure that the final home position is accurate to the count. After the index is found, the axis is bought to an instantaneous stop. To define where the zero position is, the program issues the DP (define position) command. The values given to this command must be "taught" when the machine is set up for the first time. Once the DP command is issued, we can safely issue position commands. Note that it is not necessary to actually move the axes to the zero position.

Dispense Cycle

The dispense cycle is a series of point-to-point moves over the entire XY grid. *Figure 6* shows the dispense pattern executed by the program as well as the actual path traverse by the pipette. A snake pattern is chosen to minimize the total move time required to dispense an entire tray (as opposed to starting the X position from the left-most cell for each Y position).

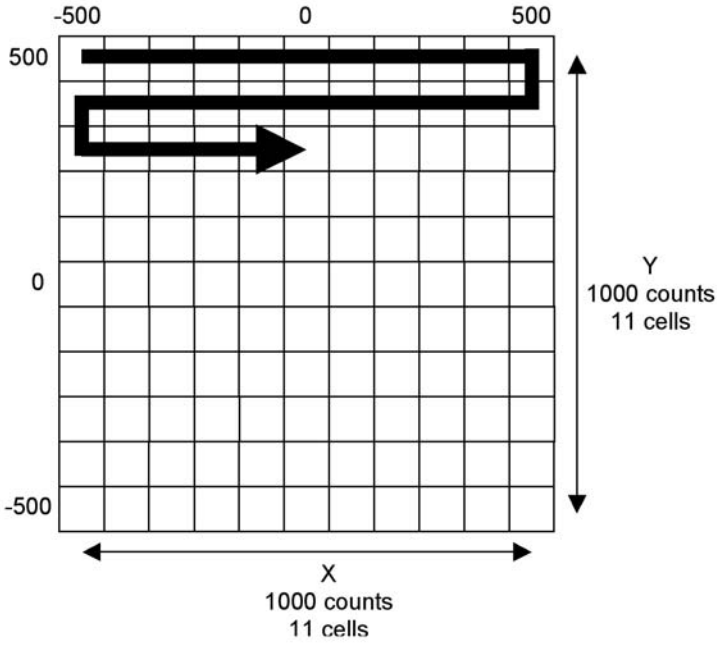

Figure 6. Dispense pattern and path

Pause and Resume

If the door is opened during a dispense cycle, the machine is paused for safety. This is implemented by using #ININT and by setting the speed SP to zero.

#ININT is an interrupt routine that is armed with the II command. II1 indicates that #ININT will run when the first digital input goes to the low state. The #ININT routine pauses the execution of the of the main program until #ININT returns. When #ININT returns, the main program is resumed right where it left off.

The first thing #ININT does is save the current slew speed (_SPn) and then sets the speed of all axes to zero (SP*=0). This will cause all axes to decelerate to a controlled stop. All axes will remain stopped until the speed is set to a non zero value. #ININT sets the speeds to their saved values once the operator closes the door. When this happens, the axes accelerate back to their original speeds to finish their moves.

Program Listing

The complete program used for this article is included here. The comments explain the code in detail:

```
#AUTO                       ;'runs on controller power up and does one-time initialization
  'initialize constants
  Grid = 11                 ;'set grid size to 11x11
  Trayy = 1000              ;'set tray size  to 1000 counts by 1000 counts
  Trayx = 1000
  Zdist = 1000              ;'distance pipette must move into the tray to dispense

  'axes are homed to a the reverse limit switch, index pulse, and offset
  'find limit switches
  JG*=-10000                ;'jog towards reverse limit switch
  BGXYZ                     ;'begin motion towards limit
  AMXYZ                     ;'wait until we hit the limit

  'find index pulses
  JG*=500                   ;'move slowly towards the index pulse
  FIXYZ                     ;'find index
  BGXYZ                     ;'begin motion towards index
  AMXYZ                     ;'wait until we hit the index.  Position is set to 0.
  DP -825, -3783, -1581     ;'define offset where zero is

#Dispens                    ;'runs once per sample tray
  II0                       ;'disarm input interrupt #ININT for door (digital input 1)
  AI-1                      ;'wait for operator to open door
  AI1                       ;'wait for operator to close door after placing new tray
  II1                       ;'arm input interrupt #ININT for door (digital input 1)

  N = 0                     ;'reset x counter
  M = 0                     ;'reset y counter
  SP*=10000                 ;'set speed
  #Loop                     ;'runs once per tray cell
    x = (-Trayx / 2) + ((Trayx / (Grid - 1)) * N) ;'compute x location
    y = (-Trayy / 2) + ((Trayy / (Grid - 1)) * M) ;'compute y location
    PA x, y                 ;'move to the absolute position specified
    BGXY                    ;'begin motion
    AMXY                    ;'wait until motion is complete

    PAZ = Zdist             ;'move pipette into tray cell
    BGZ                     ;'begin pipette motion
    AMZ                     ;'wait until pipette motion is complete

    SB 2                    ;'dispense
    WT 200                  ;'wait for the correct amount of liquid
    CB 2                    ;'stop dispense

    PAZ = 0                 ;'move pipette out of tray cell
    BGZ                     ;'begin pipette motion
    AMZ                     ;'wait until pipette motion is complete
```

(Continued next page)

(Continued from page 4)

```
      'the following if statement produces the snake pattern
      IF (@FRAC[M / 2] = 0)  ;'M (y position) is even
        N = N + 1            ;'increment x position
        Check = (N <= (Grid - 1))
      ELSE                   ;'M (y position) is odd
        N = N - 1            ;'decrement x position
        Check = (N >= 0)
      ENDIF
   JP#Loop, Check ;'loop x
   IF (@FRAC[M / 2] = 0)     ;'M (y position) is even
      N = Grid - 1           ;'reset x position
   ELSE                      ;'M (y position) is odd
      N = 0                  ;'reset x position
   ENDIF

   M = M + 1                 ;'increment y position
  JP#Loop, M <= (Grid - 1)   ;'loop y
JP#Dispens
EN

#ININT                       ;'runs when the door opens.  pause execution
   Xspeed = _SPX             ;'store the current slew speeds
   Yspeed = _SPY
   Zspeed = _SPZ
   SP*=0                     ;'pause the motion
   AI1                       ;'wait until the door is closed
   SP Xspeed, Yspeed, Zspeed ;'resume execution
RI1
```