

Using Stepper Motors for Motion Control

Introduction

This application note describes how to interface two stepper motors to a small hobbyist XY table so that a Rabbit-based controller and its Prototyping Board can be used to control the stepper motors. A sample program shows how to calibrate the motion of the stepper motors by determining the number of revolutions per inch of movement along each axis. Once the calibration is completed, the resulting values can be used in other programs to allow the stepper motors to provide precise motion control. Pick and place machines are but one example of where this precise motion control can be put to use.

Stepper Motor Operating Principles

A stepper motor relies on electromagnetism to work. Electromagnets or coils are set up in a circle as a stator around a permanent magnet axis or rotor. Figure 1 shows a simple two-pole bipolar stepper motor.

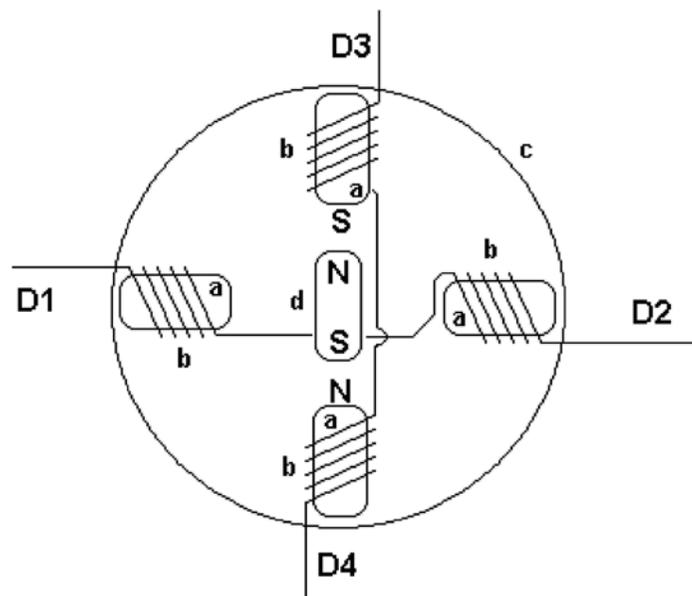


Figure 1. Basic Bipolar Stepper Motor
(a—teeth, b—winding or coil, c—stator, d—rotor)

As electric current flows through the coil, the polarity induced by the flowing current is determined according to Ampere's Law and the right-hand rule. Extend your right hand so that the fingers point in the direction of the current flow. The thumb on your right hand will point in the direction of the north pole of the coil—see Figure 2.

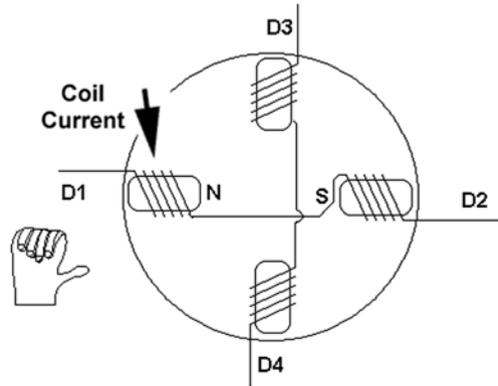


Figure 2. How to Determine Polarity

As one pair of the coils is energized, the rotor will try to orient itself with the induced magnetic field. As adjacent coils are energized and de-energized, the rotor will rotate to “follow the field.” The movement occurs when the polarity of only one pair of coils changes. Table 1 shows this.

Table 1. Stepper Motor Movement Pattern

Coil				Resulting hex Value
D1	D2	D3	D4	
+	-	+	-	0x0A
-	+	+	-	0x06
-	+	-	+	0x05
+	-	-	+	0x09

You can try different patterns.

Advanced Micro Systems Inc. has a good write-up at www.ams2000.com/stepping101.html that provides a more detailed explanation of stepper motors.

Assembly

Bill of Materials

Hardware	Vendor	Part Number
MicroLux X-Y Table Attachment	Micro-Mark	82389
2 stepper motors	Applied Motion Products	5017-009
2 STMicroelectronics L293D driver chips	Rabbit Semiconductor	660-0205
RCM3100	Rabbit Semiconductor	101-0157
RCM3300 Prototyping Board	Rabbit Semiconductor	101-0877
RCM3100 Programming Cable	Rabbit Semiconductor	101-0513
DE9 female to 10-pin IDC adapter cable	Rabbit Semiconductor	540-0009
9 V, 1 A DC power supply		
3/8" bar stock	Any Hardware Store	
11 mm 3/8" socket		
10 mm 3/8" socket		
Aluminum, 3/8" milling stock		
3/16" × 3/4" × 1/2" mounting bracket		
1/8" × 2" × 2-3/4" mounting bracket		
5/8" diameter, 3/8" high nylon spacers		

Construction Details

We had to machine a short section of 3/8" bar stock to act as an adapter between the round shaft of the stepper motor and the socket. We cut the bar stock into pieces of approximately 3/4". The pieces were then drilled axially to receive the shaft of the stepper. Machining was done for an Allen locking screw. The exterior of the bar stock was then filed for ease of entry into the socket and to make allowances for wobble caused by off-centered drilling—see Figure 3.

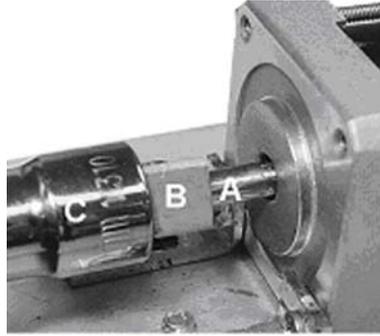


Figure 3. Stepper Motor Armature, Fitting, and Socket
(A—stepper motor armature, B—machined 3/8" fitting, C—10 mm socket)

The mounting bracket for the stepper of the y axis, the longer of two axes, was fashioned from the larger mounting bracket. Mounting brackets for each stepper motor were cut down to allow for a better fit to each stepper motor.

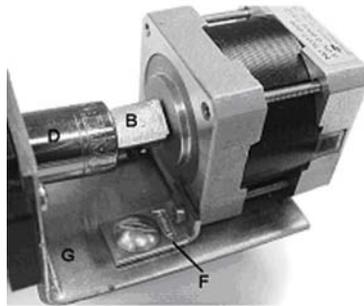


Figure 4. y Axis Mounting Detail
(B—machined 3/8" bar stock, D—11 mm socket,
F—small mounting bracket, G—large mounting bracket)

Figure 5 shows the completed XY table.

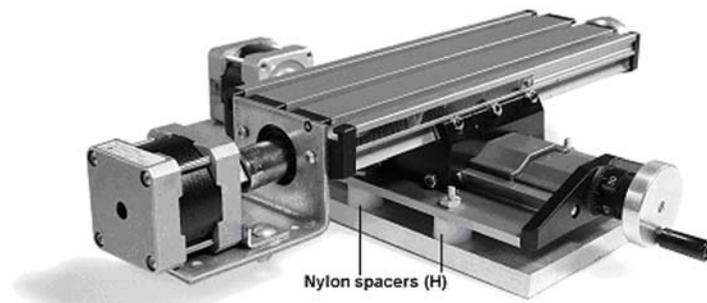


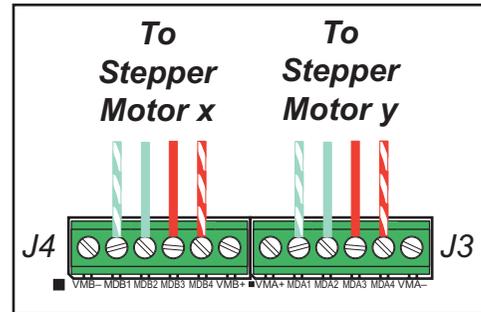
Figure 5. XY Table Completely Set Up

Connections

Stepper Motor Electrical Connections

Connect the stepper-motor connections as follows to screw-terminal headers J3 and J4 on the Prototyping Board.

- Green/White—MDX1
- Green—MDX2
- Red—MDX3
- Red/White—MDX4
- Black—no connect
- White—no connect

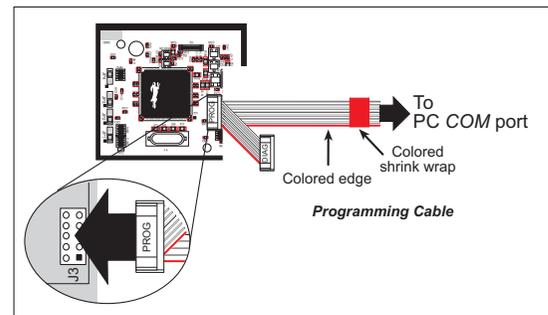


The stepper motors for the longer y axis and the x axis were connected to MDA and MDB respectively as shown in the diagram.

PC Connections

Once the RCM3100 is plugged in to the Prototyping Board, use the programming cable to connect your PC to header J1 on the RCM3100 via the **PROG** connector on the programming cable. Be sure to orient the marked (usually red) edge of the cable towards pin 1 of header J1.

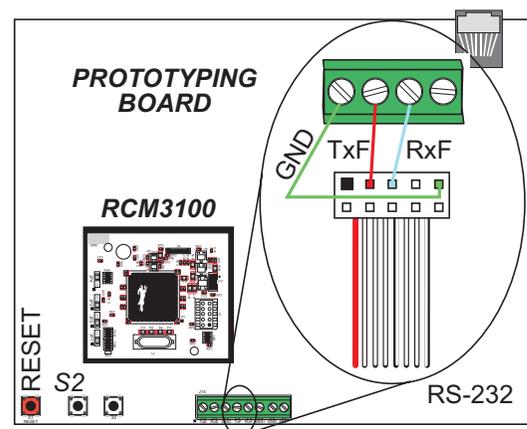
If you plan to use the same PC with only one COM port for the Serial Port F connection described below, disconnect the programming cable once a program is running.



Connect Serial Port F (TxF and RxF on screw-terminal header J14) on the Prototyping Board to the COM port on your PC as shown in the diagram. You can use a DE9 to 10-pin IDC adapter cable with your own hookup wire to connect the screw-terminal headers to the IDC plug.

You will use this RS-232 serial connection to interface with the sample programs to calibrate and use the stepper motors via a Tera Term or Hyperterminal interface on your PC. Use the following RS-232 settings for Tera Term or Hyperterminal.

- 8 data bits
- 1 stop bit
- No parity
- No flow control
- 19200 bps



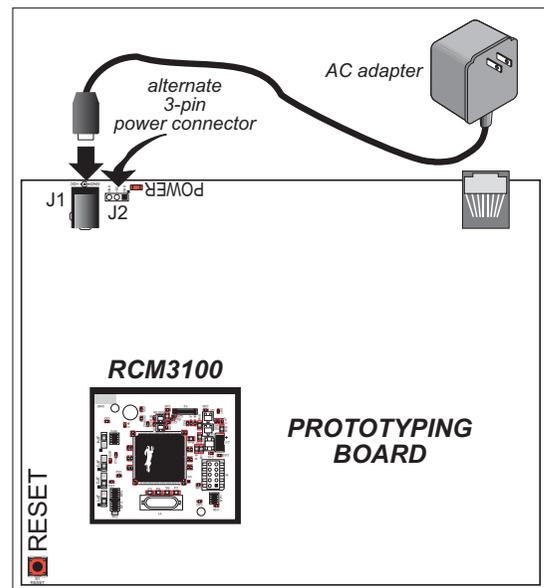
Power-Supply Connections

Connect the AC adapter to jack J1 on the Prototyping Board. Plug in the AC adapter. The power LED on the Prototyping Board should light up. The RCM3100 and the Prototyping Board are now ready to be used.

NOTE: A **RESET** button is provided on the Prototyping Board to allow a hardware reset without disconnecting power.

Alternate Power-Supply Connections

A power supply may also be connected to 3-pin header J2 on the Prototyping Board. The center pin of J2 is always connected to the positive terminal, and either edge pin is negative. The power supply should deliver 8 V to 30 V DC at 8 W.



Sample Programs

The sample programs discussed in this section are available in a zip archive (AN410.zip) that can be downloaded from www.rabbit.com/documentation/docs/refs/AN410/AN410.zip.

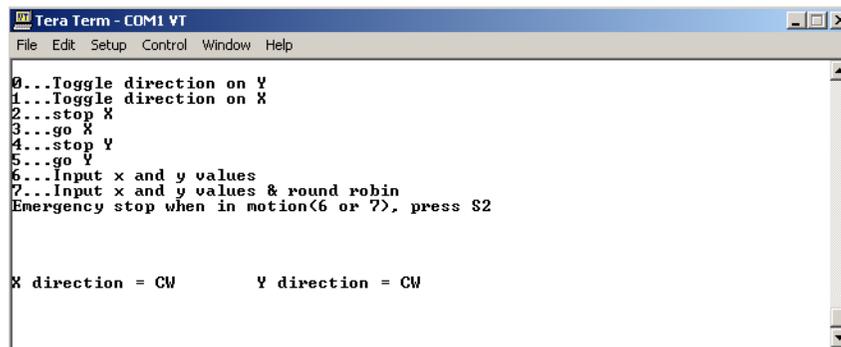
Running a Sample Program

To run a sample program, open it with the **File** menu (if the sample program is not already open), and then run it by selecting **Run** in the **Run** menu (or press **F9**). The RCM3100 must be plugged in to the Prototyping Board and must be connected to the PC running Dynamic C using the programming cable.

Calibrate Stepper Motors

The calibration program **XY_CAL.C** is used to calibrate the setup. A counter is incremented each time the program cycles through the interrupt service routine. Once the stepper motor stops moving, the distance moved is entered and is used to calculate the number of counts per unit distance.

Once the sample program is running, the following menu will be displayed in the Hyperterminal or Tera Term window.



```
Tera Term - COM1 VT
File Edit Setup Control Window Help
0...Toggle direction on Y
1...Toggle direction on X
2...stop X
3...go X
4...stop Y
5...go Y
6...Input x and y values
7...Input x and y values & round robin
Emergency stop when in motion(6 or ??), press S2

X direction = CW      Y direction = CW
```

Let's examine the menu step by step.

The first two options toggle the direction the *y* and *x* motors rotate. Initially, the direction is CW or clockwise. When you press **0** or **1**, you will toggle the *y* or *x* motor to rotate CCW or counterclockwise. The Hyperterminal or Tera Term window will display the direction once you have done the toggle.

Once you have established the direction the motors will rotate, you're ready to do the calibration. Note the starting position of the *x* motor, then enter **3** to get the motor turning. After the motor has advanced the desired amount, enter **2** to stop the motor. The Hyperterminal or Tera Term window will prompt you for inches or counts. Press **i** to indicate that you will enter the distance the motor advanced in decimal inches, or press **c** to display the counts.

```

Tera Term - COM1 VT
File Edit Setup Control Window Help

0...Toggle direction on Y
1...Toggle direction on X
2...stop X
3...go X
4...stop Y
5...go Y
6...Input x and y values
7...Input x and y values & round robin
Emergency stop when in motion<6 or ?>, press S2

for X axis,<i>nch or <c>ount
X OFF
X direction = CW      Y direction = CCW

```

When you provide the distance the motor moved in inches, enter the value (up to two decimal places), then press **<Enter>**. The Hyperterminal or Tera Term window will then display the calibration in counts/inch. Note this value so that you can use it in the **XY_TABLE.C** sample program.

```

Tera Term - COM1 VT
File Edit Setup Control Window Help

0...Toggle direction on Y
1...Toggle direction on X
2...stop X
3...go X
4...stop Y
5...go Y
6...Input x and y values
7...Input x and y values & round robin
Emergency stop when in motion<6 or ?>, press S2

for Y axis,Actual inches in decimal: 1.00
X Cts/in:8327
X OFF
X direction = CW      Y OFF
X counts      8329

```

Now you can repeat the process for the y motor by entering **5** to get the motor turning. After the motor has advanced the desired amount, enter **4** to stop the motor. Press **i** to enter the distance the motor advanced in decimal inches (up to two decimal places), then press **<Enter>**. The Hyperterminal or Tera Term window will then display the calibration in counts/inch. Note this value so that you can use it in the **XY_TABLE.C** sample program.

The **XY_CAL.C** program also allows you to move the motors by pressing **6** or **7**, then entering the distance in inches the x and y motors should move. Keep in mind that this distance should not exceed the maximum length of the XY table. While the motors are moving, you may stop them immediately (for example, in an emergency) by pressing the **S2** button on the Prototyping Board.

How the Sample Program Works

First, the series of values for the movement of the stepper motor in Table 1 is assigned to the `stepp_patt_bip()` function call.

```
const byte stepp_patt_bip[] = { 0xa,0x6,0x5,0x9 };
```

The motor will turn one way (“clockwise”) for the sequence shown, and will turn the opposite way (“counterclockwise”) if the order of the movement values is reversed. The stepper values are written during the interrupt service routine (ISR) for Timer B.

Next `display_menu()` presents the menu options.

```
void display_menu() {
    serFputs("\x1b[2J");

    goxys(1,2,"0...Toggle direction on Y\0");
    goxys(1,3,"1...Toggle direction on X\0");
    goxys(1,4,"2...stop X\0");
    goxys(1,5,"3...go X\0");
    goxys(1,6,"4...stop Y\0");
    goxys(1,7,"5...go Y\0");
    goxys(1,8,"6...Input x and y values\0");
    goxys(1,9,"7...Input x and y values & round robin\0");
}
```

The following code processes the **0** and **1** entries made in response to the menu.

```
if((i = serFgetc()) != -1) { // if a character has been received
    switch(i) {
        case '0':
            if(dir_y == CW) {
                direction_Y(CCW);
                dir_y = CCW;
            }
            else {
                direction_Y(CW);
                dir_y = CW;
            }
            break;
        case '1':
            if(dir_x == CW) {
                direction_X(CCW);
                dir_x = CCW;
            }
            else {
                direction_X(CW);
                dir_x = CW;
            }
            break;
    }
}
```

Menu entries **3** and **5** clear the variable `counter_x` and `counter_y` respectively, and the number of pulses is counted in the ISR. Entries **2** and **4** stop the counting when the motor is stopped, and a decision is made whether to translate the counts to counts per inch.

```
case '2':
    toggle_X(OFF);
    inch_count_choice(X);          // for determining counts
                                   // per inch, if needed
    counter_pointer = &dummy;     // counter_pointer
                                   // pointing to dummy variable
    display_results(X);
    break;
case '3':
    toggle_X(ON);
    counter_x = 0L;
    counter_pointer = &counter_x; // counter_pointer is pointing
                                   // to variable for monitoring
                                   // counts on X axis
    break;
case '4':
    toggle_Y(OFF);
    inch_count_choice(Y);          // for determining counts
                                   // per inch, if needed
    counter_pointer = &dummy;
    display_results(Y);
    break;
case '5':
    toggle_Y(ON);
    counter_y = 0L;
    counter_pointer = &counter_y; // counter_pointer is pointing
                                   // to variable for monitoring
                                   // counts on Y axis
    break;
```

Menu entry **6** will cause the stepper motors to rotate in one direction, one at a time, and menu entry **7** will also provide a return path to the “home” position.

```
case '6': // one direction
    get_axis_value(X);
    get_axis_value(Y);

    move_axis(X,dir_x);
    move_axis(Y,dir_y);
    break;
case '7': // round robin
    get_axis_value(X);
    get_axis_value(Y);

    move_axis(X,dir_x);
    move_axis(Y,dir_y);

    dir_x = dir_x == CW ? CCW : CW;
    dir_y = dir_y == CW ? CCW : CW;

    move_axis(X,dir_x);
    move_axis(Y,dir_y);
    break;
```

The `move_axis()` function call deals with the ramping up, the “cruise” section, and the ramping down of the stepper motor speed.

The speed calculation is based on determining the slope of the increase (or decrease) in speed you are seeking over a given distance. Hence the equation would be

$$\text{increment} = (\text{LOW_SPEED_DIVISOR} - \text{HI_SPEED_DIVISOR}) / \text{RAMP_WIDTH}$$

```
void move_axis(int axis, int dir) {
    shared unsigned long actual_counts;
    unsigned long scratch;
    char divisor, old_divisor, increment;

    movement_struct *t;

    if(axis == X)
        t = &X_axis; // set pointer to x_axis structure
    else
        t = &Y_axis; // set pointer to y_axis structure

    if(!counter_x || !counter_y) // if not calibrated
        return; // then return
```

This section shows how to find the value by which the divisor variable will be incremented:

```
old_divisor = 0;

scratch = (LOW_SPEED_DIVISOR - HI_SPEED_DIVISOR)/RAMP_WIDTH;

increment = (char)scratch;

actual_counts = 0L;

counter_pointer = &actual_counts; // counter_pointer now to update
                                   // actual counts

if(axis == X) {
    direction_X(dir);
    toggle_X(ON);
}
else {
    direction_Y(dir);
    toggle_Y(ON);
}

divisor = LOW_SPEED_DIVISOR;

if(divisor != old_divisor) {
    set_speed(divisor);
    old_divisor = divisor;
}
```

The stepper motors have now started to move, and the Hyperterminal or Tera Term displays are being updated.

```
while(actual_counts < t->target_count) {
    if(!BitRdPortI(PGDR, 0)) { // emergency stop
        toggle_X(OFF);
        toggle_Y(OFF);

        counter_pointer = &dummy;
        return;
    }

    if(axis == X) {
        sprintf(buf, "target:%10lu", t->target_count);
        goxys(1,17,buf);

        sprintf(buf, "actual:%10lu", actual_counts);
        goxys(1,18,buf);

        if(t->inch_or_count == INCH) {
            sprintf(buf, "target inches:%.3f", t->target_inch);
            goxys(1,19,buf);

            sprintf(buf, "actual inches:%.3f",
                (float)(actual_counts)/x_cts_per_in);
            goxys(1,20,buf);
        }
    }
}

else {
    sprintf(buf, "target:%10lu", t->target_count);
    goxys(25,17,buf);

    sprintf(buf, "actual:%10lu", actual_counts);
    goxys(25,18,buf);

    if(t->inch_or_count == INCH) {
        sprintf(buf, "target inches:%.3f", t->target_inch);
        goxys(25,19,buf);

        sprintf(buf, "actual inches:%.3f",
            (float)(actual_counts)/y_cts_per_in);
        goxys(25,20,buf);
    }
}
```

This code ramps up the stepper motor before the “cruise” phase.

```
if(actual_counts < RAMP_WIDTH) {
    if(divisor > (HI_SPEED_DIVISOR + increment))
        divisor -= increment;

    else
        divisor = HI_SPEED_DIVISOR;
}
```

This code controls the “cruise” phase.

```
if(actual_counts < (t->target_count - RAMP_WIDTH) &&
    actual_counts > RAMP_WIDTH)
    divisor = HI_SPEED_DIVISOR;
```

This code ramps down the stepper motor after the “cruise” phase.

```
if(actual_counts > (t->target_count - RAMP_WIDTH)) {
    if(divisor < (LOW_SPEED_DIVISOR - increment))
        divisor += increment;

    else
        divisor = LOW_SPEED_DIVISOR;
}

if(divisor != old_divisor) {
    set_speed(divisor);
    old_divisor = divisor;
}
```

Movement along the selected axis has finished.

```
if(axis == X)
    toggle_X(OFF);
else
    toggle_Y(OFF);

counter_pointer = &dummy; // point to dummy variable when done
}
```

Operate Stepper Motors

The `XY_TABLE.C` sample program accepts an array of x,y coordinates via Serial Port F. Once the stepper motors are “homed” to their starting position, the sample program calculates the motor movement and direction to move to from one x,y position to another. Operator input is then requested at that location. The process is repeated until the last numerical pair has been processed.

Before you compile and run the sample program, there are several things to do.

1. Enter the maximum range of motion and the calibration values in counts/inch that you established for the x and y motors when you ran the `XY_CAL.C` sample program. These `#define` statements are near the beginning of the `XY_TABLE.C` sample program.

```
#define X_MAX 1.5    // max inches on X
#define Y_MAX 3.5    // max inches on Y

#define X_INCH_FACTOR 8329 // counts per inch
#define Y_INCH_FACTOR 8101
```

Remember to save the sample program (**File > Save**) once you have changed the entries.

2. Create the array of x,y coordinates relative to the home position to be entered via Serial Port F. You can do this as a txt file in Notepad with each x value on a line by itself followed by the corresponding y value on a line by itself, followed by the next x value on a line by itself, etc. Use the entry $(x,y) = (0,0)$ for when you wish the stepper motors to be in the home position. Remember to keep the x,y coordinates within the maximum range of motion specified in Step 1; otherwise the sample program will not drive the stepper motors as a safety precaution.

Now you may compile and run the `XY_TABLE.C` sample program.

Now with Hyperterminal or Tera Term running, you’re ready to respond to the prompts in the Hyperterminal or Tera Term window. The first question you get is to tell the sample program how many x,y coordinates you created in Step 2.

How many pairs of entries?

Provide the answer and press **Enter**. Next the RCM3100 reports that it is ready to receive the information.

Ready to receive 5 pairs...

Depending on whether you’re using Hyperterminal or Tera Term, do **Transfer > Send File** or **File > Send File** and select the txt file you created in Step 2.

Next, you will be prompted to “home” the stepper motors.

Press + or - for CW or CCW on X axis or 0 for done

Keep tapping **+** or - keystrokes until the x motor is in its home position, then press **0**.

Press + or - for CW or CCW on Y axis or 0 for done

Now do the same for the y stepper motor.

The sample program will now advance the stepper motors to the first set of x,y coordinates and will prompt you.

press a key to continue

Once the stepper motors have proceeded through all the x,y coordinates, the sample program will report “All done.” Press the **RESET** button on the Prototyping Board to relive the experience.

How the Sample Program Works

Variables and structures are used for the incoming serial data:

```
float buf[MAX_ENTRY][2]; // for incoming serial data

struct {
    unsigned long x_counts;
    unsigned long y_counts;
    char x_dir;
    char y_dir;
}steps[MAX_ENTRY];
```

The following code in `main()` first looks to receive the number of *x,y* pairs to receive, then checks to see if the data are within limits. If the data are not within the limits, the process is repeated.

```
do {
    done = 1; // assume a successful transmission of data
              // will occur.

    serFputs("\x1b[2J"); // clear screen

    serFputs("How many pairs of entries?\n\r");

    index = 0;

    memset(incoming,0,sizeof(incoming)); // clear incoming array
    listen(incoming,&index, sizeof(incoming) - 2);

    count = atoi(incoming); // recover number of pairs to receive.
```

```

if(count) { // if count not zero, then...
    sprintf(outgoing,"Ready to receive %d pairs...\n\r", count);
    serFputs(outgoing);
    for(i = 0; i < count; i++) {
        index = 0;
        memset(incoming,0,sizeof(incoming));
        listen(incoming,&index, sizeof(incoming) - 2);
        float_var_x = atof(incoming);

        index = 0;
        memset(incoming,0,sizeof(incoming));
        listen(incoming,&index, sizeof(incoming) - 2);
        float_var_y = atof(incoming);

        printf("%.2f%.2f\n",float_var_x, float_var_y);
        // if values are less than max, store them.

        if(float_var_x < X_MAX && float_var_y < Y_MAX) {
            buf[i][0] = float_var_x;
            buf[i][1] = float_var_y;
        }
        else {
            serFputs("out of range, please press the 'r' key to
                restart\n\r");
            while(serFgetc() != 'r');
            done = 0;
            break;
        }
    }
}
else
    done = 0;
}while(!done);

```

The next step is to “home” the x and y axes by pressing + or - to rotate the particular stepper motor **HOMING_STEP** counts either clockwise (for+) or counterclockwise (for -).

```
void find_home() {
    int i;

    sprintf(buffer, "Press 8 or 2 for CW or CCW on X axis or 0 for
done\n\r");
    goxys(1,2,buffer);
    do {
        if((i = serFgetc()) != -1) {
            switch(i) {
                case '+':
                case '\r':
                    move_axis(X, CW, HOMING_STEP);
                    sprintf(buffer, "CW on X\n\r");
                    goxys(1,3,buffer);
                    break;
                case '-':
                case 0x2c:
                    move_axis(X, CCW, HOMING_STEP);
                    sprintf(buffer, "CCW on X\n\r");
                    goxys(1,3,buffer);
                    break;
                case '0':
                    goxys(1,2,"");
                    goxys(1,3,"");
                    break;
            }
        }
    }while(i != '0');

    sprintf(buffer, "Press 8 or 2 for CW or CCW on Y axis or 0 for
done\n\r");
    goxys(1,2,buffer);
}
```

```

do {
  if((i = serFgetc()) != -1) {
    switch(i) {
      case '+':
      case '\r':
        move_axis(Y, CW, HOMING_STEP);
        sprintf(buffer, "CW on Y\n\r");
        goxys(25,3,buffer);
        break;
      case '-':
      case 0x2c:
        move_axis(Y, CCW, HOMING_STEP);
        sprintf(buffer, "CCW on Y\n\r");
        goxys(25,3,buffer);
        break;
      case '0':
        goxys(1,2,"");
        goxys(25,3,"");
        break;
    }
  }
}while(i != '0');

```

The data values received serially are processed and the results are stored in memory.

```
void calculate_steps(int num_of_steps) {
    int i;
    float x_dif, y_dif;

    x_dif = buf[0][0];
    y_dif = buf[0][1];

    steps[0].x_counts = (unsigned long)(x_dif * X_INCH_FACTOR);
    steps[0].y_counts = (unsigned long)(y_dif * Y_INCH_FACTOR);

    steps[0].x_dir = CW;
    steps[0].y_dir = CW;

    for(i = 1; i < num_of_steps; i++) {
        // find distance from previous data pair
        x_dif = buf[i][0] - buf[i - 1][0];
        y_dif = buf[i][1] - buf[i - 1][1];

        // if value of move is less than previous
        // change direction of movement and
        // change sign of value of move.
        if(x_dif < 0.00) {
            steps[i].x_dir = CCW;
            x_dif *= -1.00;
        }
        else
            steps[i].x_dir = CW;
        if(y_dif < 0.00) {
            steps[i].y_dir = CCW;
            y_dif *= -1.00;
        }
        else
            steps[i].y_dir = CW;
        steps[i].x_counts = (unsigned long)(x_dif * X_INCH_FACTOR);
        steps[i].y_counts = (unsigned long)(y_dif * Y_INCH_FACTOR);
    }
}
```

Rabbit Semiconductor Inc.

www.rabbit.com