

Chapter #6: Accelerometer Projects

There are three types of projects in this chapter. The first type is a direct application of hardware and programs that were used in earlier chapters. The second type requires datalogging of the acceleration measurements, and so several activities devote themselves to a datalogging program. The third type of project requires datalogging to figure out what kind of measurements the accelerometer will report. Then, based on the results of the datalogging, you will have enough information to write a program to make the device work reliably.

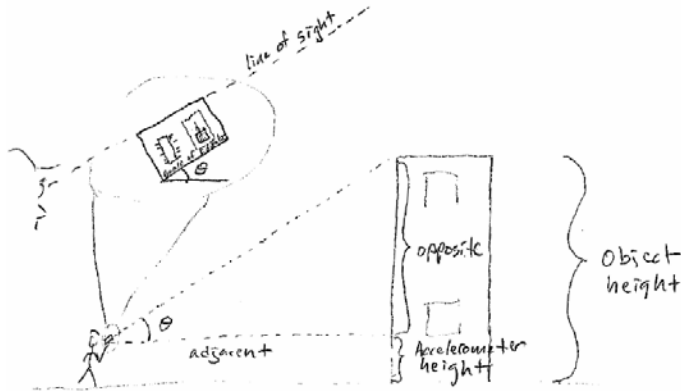
ACTIVITY #1: MEASURE HEIGHTS OF BUILDINGS, TREES, ETC.

Climbing to the top of an object to measure its height is not always convenient, practical, or even safe. This activity introduces a novel way to use some of the accelerometer measurements developed in Chapter #3 to make height measurements from a safe vantage point on the ground.

Sighting the Top and Determining Height

Figure_Next shows a scheme for measuring the height of an object with the an accelerometer. While running the LCD display version of the vertical angle of rotation program from, Chapter 3, Activity #5, site the top of the object with the edge of your board, and record the measured angle. Then, measure the distance between the spot you took your measurement and the object, which is the adjacent side shown in Figure_next. The adjacent distance, the angle θ , and the height of the accelerometer above the ground are the three key pieces of information you need to calculate the object height.

Figure_Determining Height with Line of Sight



Parts List and Circuit

Use the parts list and circuit from Chapter #3, Activity #2 for your rotating site with LCD display.

Example Program

Use the example program from Chapter #3, Activity #5, along with the modifications for LCD display introduced in the Your Turn - LCD Display section.

Procedure

- ✓ Use your board to site the top of the object and record the angle.
- ✓ Measure the distance between the site point and the object.
- ✓ Measure the height at which the accelerometer was held.
- ✓ Use the calculations introduced next to determine the object's height.

Calculations

We know from earlier chapters that $\tan \theta$ is the opposite side divided by the adjacent side of a right triangle. Multiplying both sides by the adjacent distance results in an expression for solving the opposite height. It's the adjacent distance multiplied by the tangent of the angle.

$$\tan \theta = \frac{\text{opposite}}{\text{adjacent}}$$

$$\text{opposite} = \text{adjacent} \tan \theta$$

After determining the opposite height (shown in Figure_Previous), all you have to do is add to that, the height from which you took the measurement.

$$\begin{aligned} \text{object height} &= \text{opposite} + \text{accelerometer height} \\ &= \text{adjacent} \tan \theta + \text{accelerometer height} \end{aligned}$$

Example

Let's say that the adjacent distance to an object is 10 m, and at that distance the accelerometer was held 1.5 m from the ground to get the line of sight of the top of an object. The angle reported by the accelerometer unit was 61° . From this, we can estimate the height of the object to be 19.54 m, as shown below.

Example:

Adjacent distance = 10 m

Accelerometer measured $\Theta = 61^\circ$

Accelerometer height = 1.5 m

$$\begin{aligned} \text{opposite} &= \text{adjacent} \tan \Theta \\ &= 10 \text{ m} \tan 61^\circ \\ &= (10 \text{ m})(1.804) \\ &= 18.04 \text{ m} \end{aligned}$$

$$\begin{aligned} \text{object height} &= \text{opposite} + \text{accelerometer height} \\ &= 18.04 \text{ m} + 1.5 \text{ m} \\ &= 19.54 \text{ m} \end{aligned}$$

ACTIVITY #2: RECORD AND PLAYBACK

With accelerometer projects, it will often be necessary to record and play back lots of accelerometer measurements. In some cases, recording the value is the desired function, like datalogging how a radio controlled car handles a turn. In other cases, like to detect the human walking motion, it will be necessary to understand what the measurements are before a program can be written that tracks steps. In either case, recording and playing back acceleration measurements is a necessary ingredient. This activity introduces a program with subroutines that demonstrate how to record, play back, and erase values stored in the unused portion of the BASIC Stamp's EEPROM program memory.

EEPROM Storage with DATA, WRITE and READ

While not required for recording and playing back measurements, DATA directives can be used to set aside chunks of unused program memory. The DATA directive's optional Symbol name is especially useful for recordkeeping. The Records DATA directive does not actually store any values in EEPROM addresses 0 to 9. It just reserves these bytes for your PBASIC code, and gives the address of the first byte the name Records. The RecordsEnd DATA directive reserves a single byte of at EEPROM address 10.

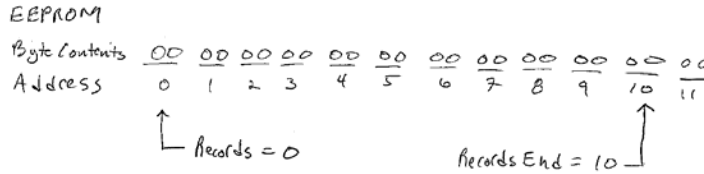
```

Records      DATA    (10)
RecordsEnd   DATA

```

The Symbol names (Records and RecordsEnd) become constants that store the starting address of the EEPROM DATA directives they precede. Figure_Next shows how it works for the two DATA directives. Since Records is the first DATA directive, it sets aside the first ten bytes (addresses 0 to 9). Since address 0 is the beginning address, Records becomes a constant for the value 0 in the program. Likewise, since the RecordsEnd DATA directive sets aside a byte at address 10, RecordsEnd becomes the constant value 10 in the program.

Figure_DATA Directives and EEPROM Addresses



The EEPROM bytes don't necessarily contain zero. With the command Records DATA (10), whatever values are already there will not be changed. If you want to initialize the EEPROM values to zero, use Records DATA 0 (10). This will store 0 in EEPROM addresses 0 to 9. The BASIC Stamp Editor only does this when it downloads the program. If you press and release your board's Reset button or disconnect and reconnect power, no values are written to those EEPROM addresses. This is a handy feature, as you will see in the next activity.

The Clear_Data subroutine in the next example program has a FOR...NEXT loop that repeats from Records to RecordsEnd (0 to 10). Each time through the loop, the eeIndex variable increases by 1, so WRITE eeIndex, 100 stores 100 in each of the EEPROM bytes, from address 0 to address 10.

```

Clear_Data:
  FOR eeIndex = Records TO RecordsEnd
    WRITE eeIndex, 100
  NEXT
  DEBUG CR, "Records cleared."
  PAUSE 1000
  RETURN


```

The Record_Data subroutine in the next example program collects values that you enter into the Debug Terminal's transmit windowpane. In the next activity, this subroutine will

be modified to store accelerometer values instead. The FOR...NEXT loop again starts at Records and repeats until eeIndex exceeds RecordsEnd. Each time through the loop, the value variable receives a signed decimal number from the Debug Terminal's transmit windowpane and stores it in the EEPROM address selected by eeIndex with WRITE eeIndex, value.

```
Record_Data:
  DEBUG CR, "Enter values from -100 to 100", CR
  FOR eeIndex = Records TO RecordsEnd
    DEBUG "Record ", DEC eeIndex, " >"
    DEBUGIN SDEC value
    value = value + 100
    WRITE eeIndex, value
  NEXT
  DEBUG CR, "End of records.",
    CR, "Press Enter for menu..."
  DEBUGIN char
  RETURN
```

Before each value variable's contents is copied to EEPROM, 100 is added to it. So instead of a value between -100 and 100, a value between 0 and 200 is stored in the EEPROM.

	<p>value = value + 100</p> <p>Before each value variable's contents is copied to EEPROM, 100 is added to it. So instead of a value between -100 and 100, a value between 0 and 200 is stored in the EEPROM. Byte size values between 0 and 255 can be stored in each EEPROM memory cell.</p> <p>Word size values can also be stored with DATA directives if you place the Word modifier before the DataItem. For example WRITE eeIndex, Word value. Keep in mind that this command uses two EEPROM bytes to store the word size value, so eeIndex will have to be incremented by 2 before the next value is written.</p>
---	---

To retrieve and display the values that were stored, the Display_Data subroutine has a FOR...NEXT loop with READ eeIndex, value. Since 100 was added to each value before it was stored with the write command, 100 is subtracted from the value variable after the READ command to bring value back into the -100 to 100 scale.

```
Display_Data:
  DEBUG CR, "Index Record",
    CR, "-----",
    CR
  FOR eeIndex = Records TO RecordsEnd
    READ eeIndex, value
    value = value - 100
```

```

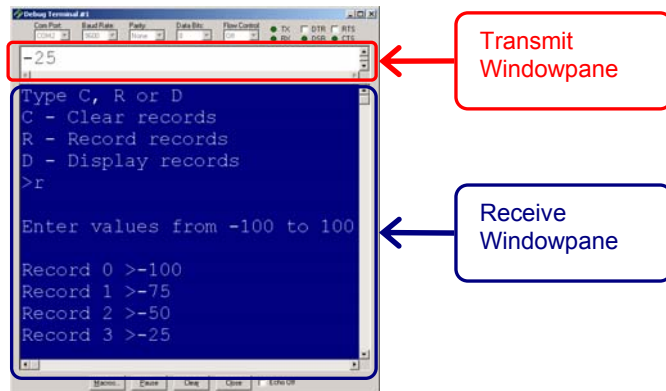
    DEBUG DEC eeIndex, CRSRX, 7, SDEC value, CR
NEXT
DEBUG CR, "Press Enter for menu..."
DEBUGIN char
RETURN

```

Example Program: EepromDataStorage.bs2

This example program displays a three choice menu in the Debug Terminal's receive windowpane shown in Figure_Next. By typing C into the Debug Terminal's transmit windowpane, the values in the EEPROM set aside for storage are cleared. If R is typed, the program records values you enter into the receive windowpane in EEPROM. If D is typed, the values that were stored in EEPROM are displayed.

Figure 6-1 Entering Values for EepromDataStorage.bs2



- √ Enter, save, and run EepromDataStorage.bs2.
- √ Click the Debug Terminal's transmit windowpane.
- √ Type R, and then enter eleven values between -100 and 100. Press the Enter key when prompted after the eleventh value to get back to the menu.
- √ Type D and verify that the values you entered are correctly displayed. Press the enter key to return to the menu.
- √ Type C to clear the memory.
- √ Type D to verify that the memory values have been cleared (set to zero).

```

' -----[ Title ]-----
' Smart Sensors and Applications - EepromDataStorage.bs2
' Demonstrates storing, retrieving and erasing values in EEPROM memory.

```

```

'{$STAMP BS2}
'{$PBASIC 2.5}

' -----[ DATA Directives ]-----
Records      DATA      (10)
RecordsEnd   DATA

' -----[ Variables ]-----
char         VAR        Byte
eeIndex     VAR        Word
value       VAR        Word

' -----[ Main Routine ]-----
DO
  DEBUG CLS,
    "Type C, R or D", CR,
    "C - Clear records", CR,
    "R - Record records", CR,
    "D - Display records", CR,
    ">"

  DEBUGIN char
  DEBUG CR

  SELECT char
    CASE "C", "c"
      GOSUB Clear_Data
    CASE "R", "r"
      GOSUB Record_Data
    CASE "D", "d"
      GOSUB Display_Data
    CASE ELSE
      DEBUG CR, "Not a valid entry.",
        CR, "Try again."
      PAUSE 1500
  ENDSELECT

LOOP

' -----[ Subroutine - Clear_Data ]-----
Clear_Data:
  FOR eeIndex = Records TO RecordsEnd
    WRITE eeIndex, 100
  NEXT
  DEBUG CR, "Records cleared."

```



```

PAUSE 1000
RETURN

' -----[ Subroutine - Record_Data ]-----
Record_Data:
  DEBUG CR, "Enter values from -100 to 100", CR
  FOR eeIndex = Records TO RecordsEnd
    DEBUG "Record ", DEC eeIndex, " >"
    DEBUGIN SDEC value
    value = value + 100
    WRITE eeIndex, value
  NEXT
  DEBUG CR, "End of records.",
    CR, "Press Enter for menu..."
  DEBUGIN char
  RETURN

' -----[ Subroutine - Display_Data ]-----
Display_Data:
  DEBUG CR, "Index Record",
    CR, "-----",
    CR
  FOR eeIndex = Records TO RecordsEnd
    READ eeIndex, value
    value = value - 100
    DEBUG DEC eeIndex, CRSRX, 7, SDEC value, CR
  NEXT
  DEBUG CR, "Press Enter for menu..."
  DEBUGIN char
  RETURN

```

Your Turn - How Many Bytes do You Want to Store?

EepromDataStorage.bs2 uses the Records and RecordsEnd for all loops that perform READ and WRITE operations. Because of this, you can change the number of values the program stores by simply changing the number of elements in the Records DATA directive.

- √ Try changing the number of elements the program stores from 11 to 7. All you have to do is change Records DATA (10) to Records DATA (6).
- √ Test and verify that it works.

In activity #4, we'll use this feature to change the number of records the program stores to 1000 with Records DATA (1000).

ACTIVITY #3: USE EEPROM TO TOGGLE MODES

This activity introduces an EEPROM trick you can use to turn the Board of Education's Reset button into a switch for selecting different program modes.

Code that Makes the Reset Button a Mode Selector

If you set aside one byte of EEPROM, it can give you the ability to select between up to 256 different program modes. In the next example program, we'll just use two modes, a menu mode, and a mode that jumps to datalogging after a slight delay. Here is a DATA directive that names an EEPROM byte Reset, and initializes the value stored by this byte to zero.

```
Reset          DATA    0
```

The simplest form of the initialization is an on/off switch configuration. This is where the value from the Reset EEPROM byte is read, 1 is added to it, and then the modified value is written to the Reset byte. The modified value is also examined to see if it is odd or even with IF value // 2 = 0 THEN... In the example below, if that condition is true, the program ends right there. The next time you press and release your board's Reset button, value will be odd, the condition will be false, and the code block will not halt the program before it has gotten to the main routine. If the Reset button is pressed and released yet again, the code block will halt the program again. The time after that, it does not halt the program, and so on. So the program converts your Board of Education's Reset button into an on/off toggle button.

```
READ Reset, value
value = value + 1
WRITE Reset, value
IF value // 2 = 0 THEN END
```

Below is an example that uses the code block in a different way. Instead of halting or allowing the program to continue, the IF...THEN code block is skipped the first time the program is run, then executed the second time the program is run (after pressing and releasing the Reset button). It is then skipped the next time and executed again the time after that. The net effect is that the program either counts down and jumps straight to the Record_Data subroutine, or moves on to the main menu in the program, depending on whether your Board's Reset button has been pressed/released an odd or even number of times.

```
' -----[ Initialization ]-----
```

```

READ Reset, value
value = value + 1
WRITE Reset, value

IF value // 2 = 0 THEN

  FOR char = 15 TO 0
    DEBUG CLS, "Datalogging starts", CR,
      "in ", DEC2 char, " seconds",
      CR, CR,
      "Press/release Reset", CR,
      "for menu..."
    FREQOUT 4, 50, 3750
    PAUSE 950
  NEXT

  GOTO Record_Data

ENDIF

```

Example Program: EepromDataStorageWithReset.bs2

This program demonstrates how to use an address in EEPROM to control the way the program behaves, depending on whether the program has been run or re-run an odd or even number of times. The number of times the program has been run will be controlled by the Reset button after download. If the Reset button has been pressed/released an even number of times, the program starts with the menu from the previous activity. If it has been pressed/released an odd number of times, it performs a countdown, and then calls the Record_Data subroutine.

- √ Open and run EepromDataStorageWithReset.bs2.
- √ Verify that you can toggle the mode the program starts in by pressing and releasing the reset button.
- √ Test the program's features, and make sure they all work.

```

' -----[ Title ]-----
' Smart Sensors and Applications - EepromDataStorageWithReset.bs2
' Demonstrates storing, retrieving and erasing values in EEPROM memory.

'{$STAMP BS2}
'{$PBASIC 2.5}

' -----[ DATA Directives ]-----
Reset          DATA    0
Records       DATA    (10)

```

```

RecordsEnd      DATA

' -----[ Variables ]-----
char            VAR      Byte
eeIndex        VAR      Word
value          VAR      Word

' -----[ Initialization ]-----

READ Reset, value
value = value + 1
WRITE Reset, value

IF value // 2 = 0 THEN

  FOR char = 15 TO 0
    DEBUG CLS, "Datalogging starts", CR,
              "in ", DEC2 char, " seconds",
              CR, CR,
              "Press/release Reset", CR,
              "for menu..."
    FREQOUT 4, 50, 3750
    PAUSE 950
  NEXT

  GOTO Record_Data

ENDIF

' -----[ Main Routine ]-----

DO

  DEBUG CLS,
        "Press/Release Reset", CR,
        "to arm datalogger ", CR, CR,
        " - or - ", CR, CR,
        "Type C, R or D", CR,
        "C - Clear records", CR,
        "R - Record records", CR,
        "D - Display records", CR,
        ">"

  DEBUGIN char
  DEBUG CR

  SELECT char
  CASE "C", "c"
    GOSUB Clear_Data
  CASE "R", "r"

```

```

        GOSUB Record_Data
    CASE "D", "d"
        GOSUB Display_Data
    CASE ELSE
        DEBUG CR, "Not a valid entry.",
            CR, "Try again."
        PAUSE 1500
    ENDSELECT
LOOP

' -----[ Subroutine - Clear_Data ]-----
Clear_Data:
    FOR eeIndex = Records TO RecordsEnd
        WRITE eeIndex, 100
    NEXT
    DEBUG CR, "Records cleared."
    PAUSE 1000
    RETURN

' -----[ Subroutine - Record_Data ]-----
Record_Data:
    DEBUG CR, "Enter values from -100 to 100", CR
    FOR eeIndex = Records TO RecordsEnd
        DEBUG "Record ", DEC eeIndex, " >"
        DEBUGIN SDEC value
        value = value + 100
        WRITE eeIndex, value
    NEXT
    DEBUG CR, "End of records.",
        CR, "Press Enter for menu..."
    DEBUGIN char
    RETURN

' -----[ Subroutine - Display_Data ]-----
Display_Data:
    DEBUG CR, "Index Record",
        CR, "-----",
        CR
    FOR eeIndex = Records TO RecordsEnd
        READ eeIndex, value
        value = value - 100
        DEBUG DEC eeIndex, CRSRX, 7, SDEC value, CR
    NEXT
    DEBUG CR, "Press Enter for menu..."
    DEBUGIN char
    RETURN

```

Your Turn - The DATA Directive's Automatic EEPROM Addressing

Did you notice that the record numbers changed in this program? Instead of 0 to 10, they were 1 to 11. Try moving the Reset DATA directive after the other two. Then, run the modified program and examine the result. Draw diagrams similar to Figure_Previous that illustrate the values stored by Reset, Records, and RecordsEnd. Draw the first diagram to illustrate the original program, and the second one to illustrate the modified program in which you changed the order of the DATA directives.

ACTIVITY #4: REMOTE DATALOGGING ACCELERATION

In this activity, you will add a piezospeaker to the existing accelerometer circuit. Then, you will modify the program so that it provides you with a remote datalogging tool that's easy to operate.

A Piezospeaker to Indicate Countdown, Start and Stop

The accelerometer circuit will be the same one used in Chapter #3, and the piezospeaker will be added below it on the breadboard.

Parts List

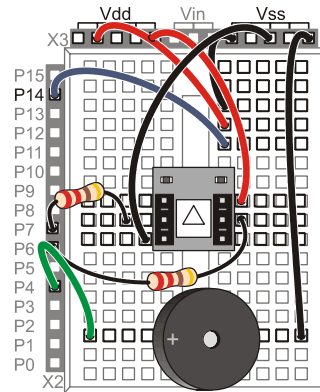
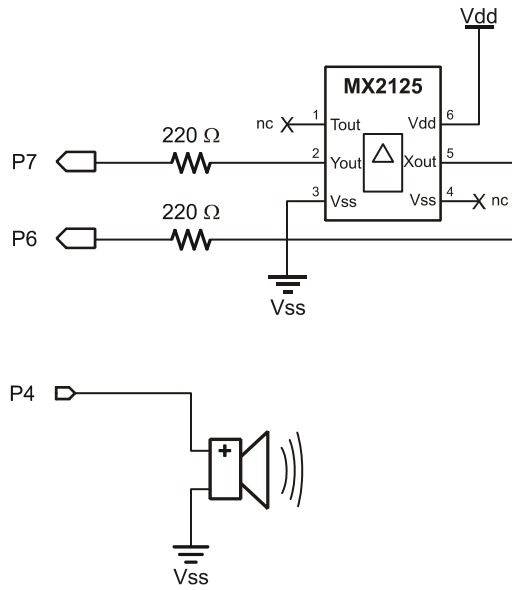
- (1) Memsic 2125 Accelerometer
- (2) 220 Ω resistors
- (7) jumper wires

Circuit

Figure_Next shows the schematic and a wiring diagram for the accelerometer and piezospeaker.

√ Build it.

Figure_Accelerometer and Piezospeaker Circuit and Wiring Diagram



Program Modifications

The next example program has been modified so that you can start, stop and restart datalogging with your board's Reset button. You can disconnect the board from your computer to perform the datalogging, and reconnect it to display the measurements in the Debug Terminal. This is a crucial feature for taking field measurements and then displaying them later.

DataloggingAcceleration.bs2 has a modified initialization section that makes the piezospeaker beep every second for ten seconds before it starts datalogging.

```
' -----[ Initialization ]-----
Init:
.
.
FOR char = 10 TO 0
.
.
.
  FREQOUT 4, 50, 3750
  PAUSE 950
```

```

NEXT
.
.
.

```

DataloggingAccelerator.bs2 also has a modified Record_Data subroutine that gets the x and y values from the accelerometer, scales them to (-100 to 100), and writes both of them to EEPROM. The FOR...NEXT loop increments in steps of 2 with the STEP 2 argument since each time through the loop, the routine saves two bytes. The Display_Data subroutine has similar modifications so that it displays both the x and y values in a table.

```

Record_Data:

    FREQOUT 4, 75, 4000
    PAUSE 200
    FREQOUT 4, 75, 4000

    DEBUG CLS, "Recording..."

    FOR eeIndex = Records TO RecordsEnd STEP 2

        PULSIN 6, 1, x
        PULSIN 7, 1, y

        x = (x MIN 1875 MAX 3125) - 1875 ** 10538
        y = (y MIN 1875 MAX 3125) - 1875 ** 10538

        WRITE eeIndex, x
        WRITE eeIndex + 1, y

    NEXT

    FREQOUT 4, 200, 4000

    DEBUG CR, "End of records.",
           CR, "Press Enter for menu..."
    DEBUGIN char
    RETURN

```

The piezospeaker also beeps twice at a higher pitch right at the beginning of the datalogging. One important feature of this ten second countdown is that you can stop the datalogging before it starts by simply pressing and releasing your board's Reset button. Then, to restart the countdown, just press and release the Reset button a again.

Example Program: DatalogAcceleration.bs2

This program takes and stores 500 accelerometer x and y-axis measurements in about 15 seconds. This equates to a sampling rate of about 33 measurements per second. This is good for a variety of measurements. To measure longer and slower processes, the Record_Data subroutine can be slowed down with a PAUSE command.

- √ Open and run DatalogAcceleration.bs2.
- √ Click the Debug Terminal's transmit windowpane.
- √ Type R to start recording, and tilt your accelerometer this way and that for fifteen seconds.
- √ When prompted, press Enter to return to the program's menu.
- √ Type D to display the measurements. Review them and verify that they correspond to how you tilted the accelerometer.
- √ Disconnect your board from the serial cable. If it starts beeping as you do so, press and release the reset button to make it stop.

When you are ready to start tilting the accelerometer for fifteen seconds, press and release the Reset button. The datalogger will beep for a ten second countdown, then end with two higher pitched beeps signaling the start of the datalogging. It will make a single high-pitched beep when it's finished.

- √ Press and release the reset button. Wait the ten seconds, then tilt your accelerometer in a pattern that you can remember for 15 seconds.
- √ Plug your accelerometer back into your computer. If it starts beeping, press and release the reset button to stop the countdown.
- √ Click the BASIC Stamp Editor's Run button to download the program to the BASIC Stamp and refresh the Debug Terminal's Menu display.
- √ Type D to display the datalogged measurements.
- √ Compare them to the directions you tilted the board and make sure they correspond.

```
' -----[ Title ]-----
' Smart Sensors and Applications - DatalogAcceleration.bs2
' Datalogs 500 x and y-axis acceleration measurements.

'{$STAMP BS2}
'{$PBASIC 2.5}

' -----[ DATA Directives ]-----
```

```

Reset          DATA      0
Records        DATA      (1000)
RecordsEnd     DATA

' -----[ Variables ]-----

char           VAR        Byte
eeIndex        VAR        Word
value          VAR        Word
x              VAR        value
y              VAR        Word

' -----[ Initialization ]-----

Init:

READ Reset, value
value = value + 1
WRITE Reset, value

IF value // 2 = 0 THEN

  FOR char = 10 TO 0
    DEBUG CLS, "Datalogging starts", CR,
      "in ", DEC2 char, " seconds",
      CR, CR,
      "Press/release Reset", CR,
      "for menu..."
    FREQOUT 4, 50, 3750
    PAUSE 950
  NEXT

  GOSUB Record_Data

ENDIF

' -----[ Main Routine ]-----

DO

  DEBUG CLS,
    "Press/Release Reset", CR,
    "to arm datalogger ", CR, CR,
    " - or - ", CR, CR,
    "Type C, R or D", CR,
    "C - Clear records", CR,
    "R - Record records", CR,
    "D - Display records", CR,
    ">"

```

```

DEBUGIN char
DEBUG CR

SELECT char
  CASE "C", "c"
    GOSUB Clear_Data
  CASE "R", "r"
    GOSUB Record_Data
  CASE "D", "d"
    GOSUB Display_Data
  CASE ELSE
    DEBUG CR, "Not a valid entry.",
    CR, "Try again."
    PAUSE 1500
ENDSELECT

LOOP

' -----[ Subroutine - Clear_Data ]-----
Clear_Data:
DEBUG CR, "Clearing..."
FOR eeIndex = Records TO RecordsEnd
  WRITE eeIndex, 0
NEXT
DEBUG CR, "Records cleared."
PAUSE 1000
RETURN

' -----[ Subroutine - Record_Data ]-----
Record_Data:

FREQOUT 4, 75, 4000
PAUSE 200
FREQOUT 4, 75, 4000

DEBUG CLS, "Recording..."

FOR eeIndex = Records TO RecordsEnd STEP 2

  PULSIN 6, 1, x
  PULSIN 7, 1, y

  x = (x MIN 1875 MAX 3125) - 1875 ** 10538
  y = (y MIN 1875 MAX 3125) - 1875 ** 10538

  WRITE eeIndex, x
  WRITE eeIndex + 1, y

NEXT

```

```

FREQOUT 4, 200, 4000

DEBUG CR, "End of records.",
        CR, "Press Enter for menu..."
DEBUGIN char

RETURN

' -----[ Subroutine - Display_Data ]-----
Display_Data:

DEBUG CR, "Index  x-axis  y-axis",
        CR, "-----  -----  -----",
        CR
FOR eeIndex = Records TO RecordsEnd STEP 2
  READ eeIndex, x
  x = x - 100
  READ eeIndex + 1, y
  y = y - 100
  DEBUG DEC eeIndex, CRSRX, 7, SDEC x, CRSRX, 14, SDEC y, CR
NEXT
DEBUG CR, "Press Enter for menu..."
DEBUGIN char
RETURN

```

Your Turn - Datalogging Rotation Angle

Chapter #3, Activity #5 introduced vertical rotation measurements with the accelerometer. Since binary radians are values from 0 to 255, you can store a single angle measurement in one EEPROM byte. This will double the number of measurements the application will take. It only takes a few modifications to DatalogAcceleration.bs2 to make it store rotation angle instead. Here's how:

- √ Save DatalogAccleration.bs2 as DatalogAngle.bs2.
- √ Update the comments in the Title section.
- √ Remove the STEP 2 arguments from the FOR...NEXT loops in the Record_Data and Display_Data subroutines.
- √ In the Record_Data subroutine, replace these two WRITE commands

```

WRITE eeIndex, x
WRITE eeIndex + 1, y

```

with this ATN operation and WRITE command

```
value = x ATN y
WRITE eeIndex, value
```

- √ Modify the display heading in the Display_Data subroutine so that it looks like this

```
DEBUG CR, "Index  angle ",
        CR, "-----  -----",
        CR
```

- √ Replace these four commands

```
READ eeIndex, x
x = x - 100
READ eeIndex + 1, y
y = y - 100
DEBUG DEC eeIndex, CRSRX, 7, SDEC x, CRSRX, 14, SDEC y, CR
```

with these

```
READ eeIndex, value
value = value */ 361
DEBUG DEC eeIndex, CRSRX, 7, SDEC x, CRSRX, 14, SDEC y, CR
```

- √ Save your changes and test the modified program.

ACTIVITY #5: RC CAR ACCELERATION STUDY

This activity demonstrates how use DatalogAcceleration.bs2 from the previous activity to analyze the acceleration forces on a radio controlled (RC) car during a variety of maneuvers.

Parts, Equipment and Circuit Diagrams

In addition to the parts for Activity #4, you will need an RC car and controller. The circuit diagrams that should be built on your board are at the beginning of Activity #4 in this chapter.

Hardware and Setup

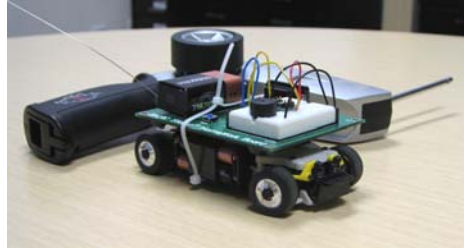
Figure_Next (a) shows an inexpensive RC car that can be obtained at many hobby shops and retail electronics outlets. Figure_Next (b) shows how the board was mounted. Rubber feet were affixed to the underside of the board in a way that prevented any of it's electrical connections from coming in contact with any of the RC car's electrical metal

parts and . Another option would be to use double-stick tape to affix the board to the roof of the plastic shell.

Figure RC Car with Acceleration Datalogger



(a)



(b)

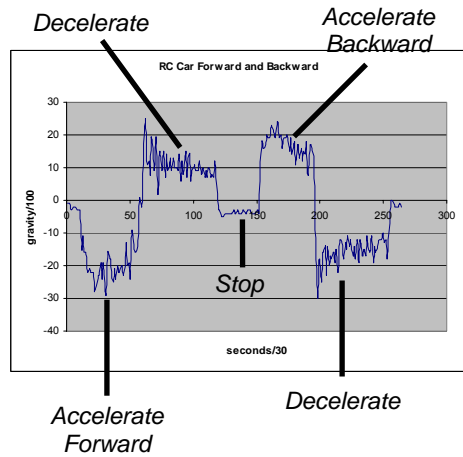


Avoid accidental short-circuits. Make sure your board is mounted on the car so that exposed metal underneath the board has no way of coming in contact with any of the RC car's metal parts or electrical connections.

How it Works

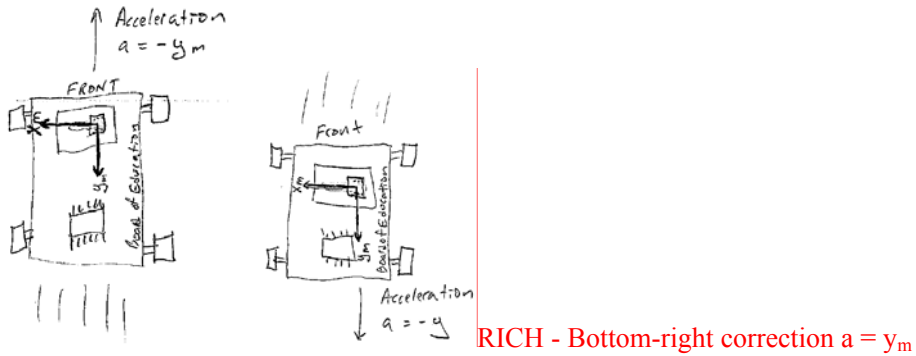
Figure_Next shows a graph of the accelerometer's y-axis measurements as the car accelerated forward, slowed to a stop, and then accelerated backwards. The measurements were acquired with `DatalogAcceleration.bs2` from Activity #4. After displaying them in the Debug Terminal, they were shaded, copied and pasted into Windows Notepad. From there, they were imported into the Microsoft Excel spreadsheet program and then graphed.

Figure 6-2 Forward/Backward Acceleration Measurements



The reason the forward acceleration is negative is because the ym sensing axis is pointing to the back of the RC car as shown in Figure_Next. So, as the car is accelerating forward, the acceleration is negative. When a car slows down, it is actually accelerating backwards. This is shown in Figure Previous. First, the car accelerated forward, then, it applied the breaks and slowed down (decelerated). The y measurement was positive, so acceleration was negative. After a brief stop, the car accelerated backwards. Notice that the y is again negative. Then, when it slows down (decelerates) from its backwards speed to stop again, the car is, in effect, accelerating forward, and the y measurement is negative again.

Figure_Acceleration vs. Accelerometer Sensing Axes

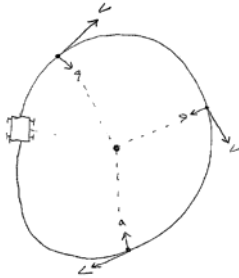


(a)

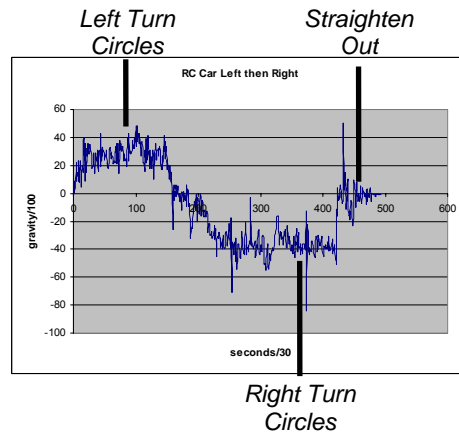
(b)

If you're in a passenger in a car, when the car accelerates forward, you can feel the seat pushing you forward. Well, if the driver makes a sharp left turn, the right side of the car pushes you to the left. That's because you are accelerating left as you turn. This is shown in Figure_Next, which illustrates how an object can be traveling forward at a constant velocity, and to make it turn, it always has to be turning toward the center of the circle it is traveling in.

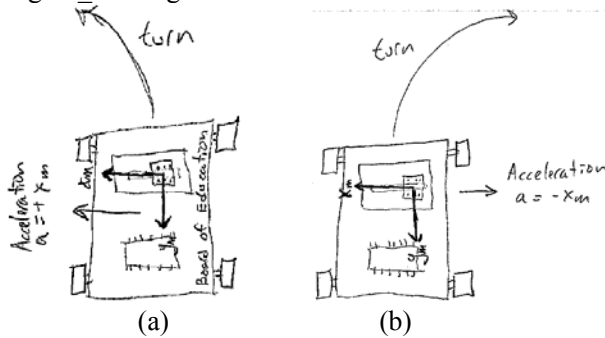
Figure_Traveling in a Circle, Always Accelerating Toward the Center.



Figure_Next shows a graph of the accelerometer's x-axis measurements as the RC car is first driven in circles turning left, then in circles turning right. Notice how the x-axis measurement shows positive acceleration as the RC car circles to the left, and negative acceleration as the car circles to the right.

Figure 6-3 Accelerometer Measurements while Driving in Circles

Figure_Next shows how the accelerometer's x-axis is oriented, and the acceleration it measures. For a left turn, the car is accelerating to the left, which for the accelerometer is a positive x-axis acceleration measurement. When it turns right, acceleration is in the opposite direction of the positive x-axis, so the x-axis measurement is negative.

Figure Sensing Acceleration in Turns

Procedure

The procedure for measuring and then graphing RC car acceleration is as follows.

- ✓ Attach your board to the RC car.
- ✓ Download DatalogAcceleration.bs2 into the BASIC Stamp.

- √ Take it to an open area and press/release the board's Reset button.
- √ Wait for the countdown to indicate that datalogging has started.
- √ Accelerate the car forward, then come to a stop. Accelerate the car backward then come to a stop, then drive an a figure-eight. When the board beeps again (after about fifteen seconds) it means the datalogging is over.
- √ Connect the board back to your PC.
- √ Run DatalogAcceleration.bs2 again.
- √ Click the Debug Terminal's transmit windowpane.
- √ Type D to display the data.
- √ Use your mouse to shade the table headings and all the measurements in the Debug Terminal's blue receive windowpane. (Don't shade the menu.)
- √ Press CTRL + C to copy the records.
- √ Open Notepad.
- √ Click Edit and select Paste.
- √ Save the file.

These next instructions explain how to import the .txt file into Microsoft Excel 2002 and graph it. If you are using a different spreadsheet program, the keywords such as space delimited, XY scatter plot may provide leads on how to accomplish it with your particular spreadsheet software.

- √ Click File and select Open.
- √ In the files of type field, select All files (*.*).
- √ Find the .txt file you saved with notepad, select it, and click the Open button.
- √ In Text Import Wizard Step 1, click the Delimited radio button, then click Next.
- √ Click the checkbox next to Space to indicate that the file is space delimited.
- √ Make sure the checkbox for "Treat consecutive delimiters as one" box is also checked, then click next.
- √ Make sure the radio button for General column data format is selected, then click finish.
- √ Your spreadsheet should be three columns wide and about 503 rows long.

The next step, which is also documented for Microsoft Excel 2002, is to run the chart utility and tell it what to graph and how you want it to look.

- √ Place the cursor in a cell somewhere to the right of your three columns of data.
- √ Click Insert and select Chart.

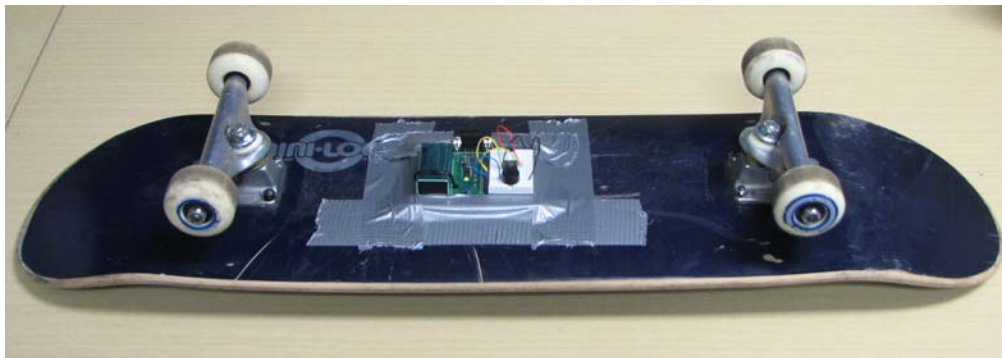
- √ In the Standard Types tab, select XY (Scatter). Also click the graphic that configures it to "Scatter with data points connected to smoothed Lines without markers". Then, click Next.
- √ Assuming your y-axis data begins in C3 and ends in C503, type C3..C503 in the Data range. Click the radio button next to Columns to indicate that the series of data points is in a column. Then, click Next.
- √ Fill in the chart title and axis information, then click Finish.
- √ Repeat for the x-axis.

!-box: **Only portions of each graph are relevant.**

Keep in mind that the data that will make sense for the y-axis is the portion of time the car accelerated forward and backward. Likewise, the part of the graph that will make sense for the x-axis is the portion of the graph when the car was turning.

ACTIVITY #6: SKATEBOARD TRICK ACCELERATION STUDY

This activity looks at a second acceleration study example. This one datalogs a skateboard trick called the ollie. The setup for datalogging the ollie shown in Figure_Next is a BASIC Stamp HomeWork Board duct taped to the underside of a skateboard.



About the Ollie

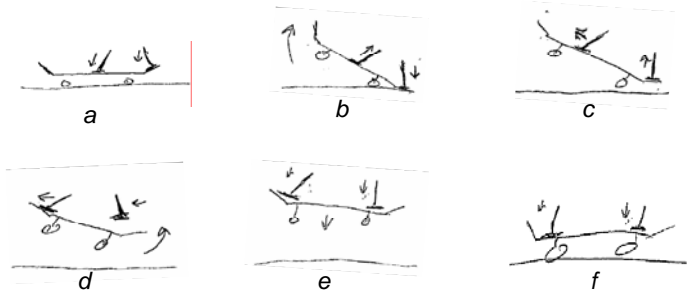
The first documented ollie was done by Allan (Ollie) Gelfand in the late 1970s. Gelfand pioneered it in ramps and bowls. The flatland version of the ollie evolved in the early 1980s, and when a skater does an ollie, he jumps, and it looks like his board is attached to

his feet, even though it's not. Regardless of the environment or skating style, most skateboard tricks today are variations of the ollie.

Ollie Mechanics

Figure_Next shows the mechanics of an ollie. As the skater jumps, (a) his feet are both pushing the board down. Just before the skater is about to become airborne, (b) he lifts his front foot and at the same time extends his back foot to tiptoe, and the tail of the board smacks the concrete. The momentum of the front of the board keeps it rising (c), and the skater now lifts his back foot, and kicks his front foot forward. This causes the back of the board to rise (d), and move slightly forward. As the deck meets the skater's back foot (e), the skater applies just enough pressure to keep the board against his feet as it falls back to the ground (f). The highest ollie to date, performed by Danny Wainright, was in excess of five feet high.

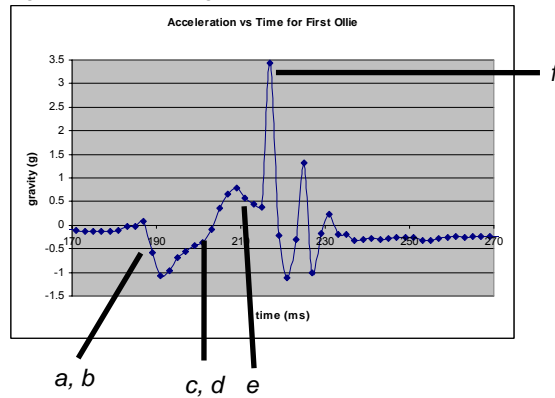
Figure 6-4 Ollie Mechanics



Graphing Ollie Acceleration

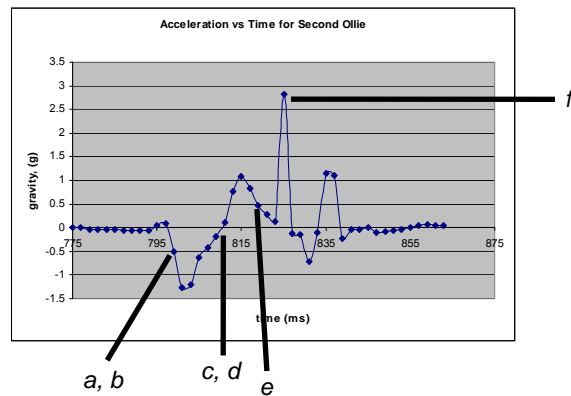
Figure_Next shows a graph of the accelerometer's y-axis for the first of two ollies that were datalogged with the next example program. Each step from Figure_Previous is marked on the graph. This particular ollie was a little deficient in Figure_Previous steps b and c, so the back of the board didn't quite meet the back foot in step e. Note that the impact of the board during step f was 3.5 g. The highs and lows that follow step f, resemble the oscillations when a bell is struck. This is partially due to the board's vibration and partially due the turbulence of the gas inside the accelerometer caused by the impact.

Figure 6-5 Figure with Drawing Canvas



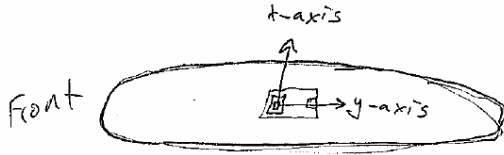
Figure_Next shows the data for a slightly better ollie. It was a little higher, and it made it to step e gracefully. Notice that step a to b is steeper, and gets to -1.25 g before rising to over 1 g for steps c and d. These values, which are larger than the ones from the previous graph, indicate a higher ollie. Notice also that the impact was below 3 g because the skater was not trying to catch up with the board on the way down.

Figure 6-6 Figure with Drawing Canvas



Datalogging an Ollie

Figure_Next shows how the accelerometer's y sensing axis is aligned to sense the skateboard's various tilts and rotations. This is the only axis we want to log in the next example program.



The program from the Activity #4 was modified to store just the raw y-axis accelerometer measurements with no scale or offset. The value of y would range from 1875 to 3125 (for ± 1 g) if no acceleration is involved. When the acceleration measurement is 3.5 g, that would result in a measurement of 4687. In any event, these are word values, and so the WRITE command in the Record_Data subroutine has to be modified so that it stores word variables. Since a word takes up two bytes, the FOR...NEXT loop still has to count in steps of 2.

```
FOR eeIndex = Records TO RecordsEnd STEP 2
  PULSIN 7, 1, y
  WRITE eeIndex, Word y
NEXT
```

Similar modifications are made to the FOR...NEXT loop in the Display_Data subroutine shown here.

```
FOR eeIndex = Records TO RecordsEnd STEP 2
  READ eeIndex, Word y
  DEBUG DEC eeIndex, CRSRX, 7, SDEC y, CR
NEXT
```

Example Program: DatalogYaxisUnscaled.bs2

This next example program was used to log the data graphed in Figure_Previous_2 and Figure_Previous_3. It gives you about ten seconds of datalogging, which is enough time for two or three ollies. Moving the data to a spreadsheet and graphing it is based on the

procedure in Activity #5. The spreadsheet was modified to generate the graphs shown in this activity by adding a column with a formula that takes the y-axis data, subtracts 2500 from it, and then divides it by 625. This gives a measurement in units of earth-gravity (g).

```
' -----[ Title ]-----
' Smart Sensors and Applications - DatalogYaxisUnscaled.bs2
' Datalogs 500 word size y-axis acceleration measurements.

'{$STAMP BS2}
'{$PBASIC 2.5}

' -----[ DATA Directives ]-----

Reset          DATA    0
Records        DATA    (1000)
RecordsEnd     DATA

' -----[ Variables ]-----

char           VAR      Byte
eeIndex        VAR      Word
value          VAR      Word
x              VAR      value
y              VAR      Word

' -----[ Initialization ]-----

Init:

READ Reset, value
value = value + 1
WRITE Reset, value

IF value // 2 = 0 THEN

  FOR char = 10 TO 0
    DEBUG CLS, "Datalogging starts", CR,
      "in ", DEC2 char, " seconds",
      CR, CR,
      "Press/release Reset", CR,
      "for menu..."
    FREQOUT 4, 50, 3750
    PAUSE 950
  NEXT

  GOSUB Record_Data

ENDIF
```

```

' -----[ Main Routine ]-----
DO

  DEBUG CLS,
    "Press/Release Reset", CR,
    "to arm datalogger ", CR, CR,
    " - or - ", CR, CR,
    "Type C, R or D", CR,
    "C - Clear records", CR,
    "R - Record records", CR,
    "D - Display records", CR,
    ">"

  DEBUGIN char
  DEBUG CR

  SELECT char
    CASE "C", "c"
      GOSUB Clear_Data
    CASE "R", "r"
      GOSUB Record_Data
    CASE "D", "d"
      GOSUB Display_Data
    CASE ELSE
      DEBUG CR, "Not a valid entry.",
        CR, "Try again."
      PAUSE 1500
  ENDSELECT

LOOP

' -----[ Subroutine - Clear_Data ]-----
Clear_Data:

  DEBUG CR, "Clearing..."

  FOR eeIndex = Records TO RecordsEnd
    WRITE eeIndex, 0
  NEXT

  DEBUG CR, "Records cleared."
  PAUSE 1000

  RETURN

' -----[ Subroutine - Record_Data ]-----
Record_Data:

```



```

FREQOUT 4, 75, 4000
PAUSE 200
FREQOUT 4, 75, 4000

DEBUG CLS, "Recording..."

FOR eeIndex = Records TO RecordsEnd STEP 2

    PULSIN 7, 1, y

    WRITE eeIndex, Word y

NEXT

FREQOUT 4, 200, 4000

DEBUG CR, "End of records.",
        CR, "Press Enter for menu..."
DEBUGIN char

RETURN

' -----[ Subroutine - Display_Data ]-----
Display_Data:

DEBUG CR, "Index  x-axis  y-axis",
        CR, "-----  -----  -----",
        CR

FOR eeIndex = Records TO RecordsEnd STEP 2

    READ eeIndex, Word y
    DEBUG DEC eeIndex, CRSRX, 7, SDEC y, CR

NEXT

DEBUG CR, "Press Enter for menu..."
DEBUGIN char

RETURN

```

Your Turn - What Makes a High Ollie?

It would be interesting to datalog and compare different skaters' ollies. The best way to do it would be to take video of each ollie, and then watch the video and examine the graph at the same time. Another thing that can be measured is the time in the air, which is the time between steps a and f in the graphs.

ACTIVITY #7: BICYCLE DISTANCE

Figure_Next shows how the board and accelerometer can be mounted inside a bicycle wheel. As the bicycle is upright, this might at first seem like an angle of rotation problem, like in Chapter #3, Activity #5. However, there is also an acceleration toward the center of the wheel that the axes will measure. This is because the accelerometer is traveling in a circular path, just like the RC car from the previous activity. This acceleration toward the center of the wheel will be different at different speeds, and will result in skewed angle measurements. The accelerometer measurements will also be effected when the bike rider applies the brakes, speeds up, and leans into turns. In addition, what criteria should be used to add one to the number of full circles the bike wheel has turned? This activity introduces hysteresis as a way of measuring wheel rotation and demonstrates how the datalogging techniques used in earlier activities can be used to examine each of these issues and test for prototype reliability.

Figure Bike Wheel Test Setup



!-Box: Do not let the metal on the underside of your board come into contact with the spokes. Use an insulating material (such as the folded up plastic bag shown in Figure_Previous) to insulate the underside of the board from the spokes.

Counting Wheel Revolutions with Hysteresis

One problem with counting wheel revolutions is making sure that the program doesn't advance the count if the wheel hasn't turned full circle. The most common mistake that is made when measuring wheel revolutions is setting a single threshold. What if the rider is waiting at a stop light, and is moving his/her bike back and forth by an inch or two? If there is a single threshold, the wheel revolution counter will keep increasing every time the rider rocks back and forth.

The next example program demonstrates a way of solving this problem with hysteresis. Hysteresis is the process of setting two different values that have to be crossed before a change in state occurs. In our case, the change of state is an increase in the wheel revolution count. With hysteresis, the measurement must fall below a low value, and then the program waits until it has risen up above a higher value before acknowledging an upward change. Then, the measurement will have to go below the low threshold again before a change from high to low is acknowledged. Each time the program acknowledges that the measurement went below the low value and then above the high value, it increases the wheel revolution count by 1.

Here is some code that performs hysteresis. In the first of the two nested DO...LOOPS, the program waits until the y axis rises above 2650. Then, the second of the two nested DO...LOOPS waits until the y axis measurement drops below 2350. Only then will it add 1 to the counter variable. After that, the program makes the peizospeaker beep, and then repeats the outer DO...LOOP. At this point, the program is back to waiting for the y axis measurement, which was below 2350 to rise back above 2650 again. Keep in mind that this is not necessarily the optimum way to measure wheel revolutions. That's for you to determine.

```

DO
    DO UNTIL y > 2650
        PULSIN 7, 1, y
    LOOP

    DO UNTIL y < 2350
        PULSIN 7, 1, y
    LOOP

    counter = counter + 1
    FREQOUT 4, 200, 3750

LOOP

```

i-box: The range between 2350 and 2650 in the code block above is referred to as deadband.

Example Program: TestWheelCounter.bs2

- √ Mount your board inside a bicycle wheel as shown in Figure_Previous. Make sure to keep a good insulator between the spokes and the underside of the board.
- √ Enter, save, and run WheelCounter.bs2.

√ Spin the wheel, and verify that it beeps once per revolution.

```
' TestWheelCounter.bs2
' Tracks bicycle wheel revolutions.

'{$STAMP BS2}
'{$PBASIC 2.5}

x          VAR      Word
y          VAR      Word
counter    VAR      Word

DEBUG CLS

DO

  DO UNTIL y > 2650
    PULSIN 7, 1, y
  LOOP

  DO UNTIL y < 2350
    PULSIN 7, 1, y
  LOOP

  counter = counter + 1
  FREQOUT 4, 200, 3750

LOOP
```

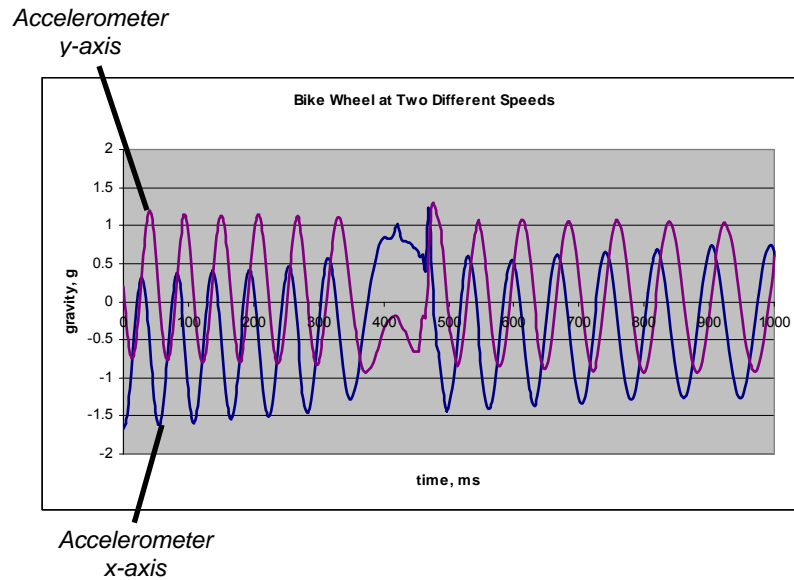
Datalogging Various Operating Conditions

It might seem at this point like the application is ready for some code that converts revolutions to distance, and maybe an LCD display and a couple of buttons for selecting LCD menu items. The problem here is we only examined the wheel turning two speeds. What about when the rider is leaning into sharp turns. Does the acceleration change then? What about in cold and hot temperatures. Will they cause the measurements to be different? It certainly wouldn't do to have a product on the market that only tracked bicycle distance some of the time. The product would get a bad reputation very quickly.

Figure_Next shows a datalogged acceleration study for the bicycle at two slightly different speeds. The area around 400 ms is where the wheel was slowed down. The important thing to note from this graph is the offset of the x and y-axis measurements. At the higher speed, the y-axis signal varied between 1 and -0.5 g while the x-axis measurements varied between -1.5 and 0.25 g. After slowing the wheel down, the y-axis

measurement varied between 1 and -1 g, while the x-axis measurement varied between about 0.7 and -1.3 g.

Figure 6-7 Figure with Drawing Canvas



Figure_Previous is just a graph of two different speeds. True, the hysteresis code from wheel counter works under both of these conditions, but does it work under ALL conditions? With this kind of question, engineers might apply a few equations to predict the accelerations under various extreme conditions that they can predict. Simulation software could also be used to verify the outcomes. Even if this kind of expertise is available, the product still has to be tested in a variety of "real life" conditions, especially to rule out the possibility of incorrect assumptions on the part of the engineers. That's where datalogging comes back into the picture. The actual prototype still has to be taken through the various conditions that it might experience on anybody's bicycle before it's safe to make the investment in the plastic case, refined electronic design that features low cost parts, mass production and inventory costs.

With this in mind, we are back to performing acceleration studies, under as many different situations as possible. Here is the program that was used to log the data for the

graph in Figure_Previous. Notice that it is logging Word values for both the x and y-axis measurements. The spreadsheet is given the job of changing the raw accelerometer PULSIN measurements into gravity measurements.

Example Program: BikeWheelAcceleration.bs2

As a project, test the bicycle meter in different temperatures, riding conditions, turns, up hill, down hill, slow, fast, etc. Look for a sequence of changes in measurements that can be tracked regardless of the conditions. If there is not a hysteresis range for all conditions, your code may need to periodically update the most recent high and low values, and then look for hysteresis within that range.

```
' -----[ Title ]-----
' Smart Sensors and Applications - BikeWheelAcceleration.bs2
' Datalogs 500 x and y-axis acceleration measurements.

'{$STAMP BS2}
'{$PBASIC 2.5}

' -----[ DATA Directives ]-----
Reset          DATA    0
Records        DATA    (1000)
RecordsEnd     DATA

' -----[ Variables ]-----
char           VAR      Byte
eeIndex        VAR      Word
value          VAR      Word
x              VAR      value
y              VAR      Word

' -----[ Initialization ]-----

Init:

READ Reset, value
value = value + 1
WRITE Reset, value

IF value // 2 = 0 THEN

  FOR char = 10 TO 0
    DEBUG CLS, "Datalogging starts", CR,
      "in ", DEC2 char, " seconds",
      CR, CR,
      "Press/release Reset", CR,
```

```

        "for menu..."
    FREQOUT 4, 50, 3750
    PAUSE 950
NEXT

GOSUB Record_Data

ENDIF

' -----[ Main Routine ]-----
DO

    DEBUG CLS,
        "Press/Release Reset", CR,
        "to arm datalogger ", CR, CR,
        " - or - ", CR, CR,
        "Type C, R or D", CR,
        "C - Clear records", CR,
        "R - Record records", CR,
        "D - Display records", CR,
        ">"

    DEBUGIN char
    DEBUG CR

    SELECT char
    CASE "C", "c"
        GOSUB Clear_Data
    CASE "R", "r"
        GOSUB Record_Data
    CASE "D", "d"
        GOSUB Display_Data
    CASE ELSE
        DEBUG CR, "Not a valid entry.",
            CR, "Try again."
        PAUSE 1500
    ENDSELECT

LOOP

' -----[ Subroutine - Clear_Data ]-----

Clear_Data:
    DEBUG CR, "Clearing..."
    FOR eeIndex = Records TO RecordsEnd
        WRITE eeIndex, 0
    NEXT
    DEBUG CR, "Records cleared."
    PAUSE 1000
    RETURN

```

```

' -----[ Subroutine - Record_Data ]-----
Record_Data:

  FREQOUT 4, 75, 4000
  PAUSE 200
  FREQOUT 4, 75, 4000

  DEBUG CLS, "Recording..."

  FOR eeIndex = Records TO RecordsEnd STEP 4

    PULSIN 6, 1, x
    PULSIN 7, 1, y

    WRITE eeIndex, Word x
    WRITE eeIndex + 2, Word y

  NEXT

  FREQOUT 4, 200, 4000

  DEBUG CR, "End of records.",
          CR, "Press Enter for menu..."
  DEBUGIN char

  RETURN

' -----[ Subroutine - Display_Data ]-----
Display_Data:

  DEBUG CR, "Index  x-axis  y-axis",
          CR, "-----  -----  -----",
          CR

  FOR eeIndex = Records TO RecordsEnd STEP 4
    READ eeIndex, Word x
    READ eeIndex + 2, Word y
    DEBUG DEC eeIndex, CRSRX, 7, SDEC x, CRSRX, 14, SDEC y, CR
  NEXT
  DEBUG CR, "Press Enter for menu..."
  DEBUGIN char

  RETURN

```


Your Turn

Another thing to examine is how vertical plane rotation measurements performs under the various bicycle wheel conditions.

- √ The Your Turn section of Activity #4 datalogs brad measurements. Use it to datalog your bicycle wheel rotation in brads.
- √ Graph the rotation over time in under the various riding conditions discussed in this Activity.

Is there an angle measurement behavior that can be hysteresis can be applied under all riding conditions?

SUMMARY

This chapter introduced a variety of accelerometer applications and datalogging techniques that can be used to study the accelerometer's measurements in various conditions, and in some cases, use them to refine your programs. When sighting the top of an object, vertical plane rotation measurements can be used with the distance to the object and some trigonometry to determine the object's height.

DATA directives with optional Symbol names were introduced as a way to simplify recordkeeping in datalogging programs. They can be used to define ranges of unused EEPROM program memory. Since Symbol names store the starting address of DATA directives, they can be used in FOR...NEXT loops that perform READ/WRITE operations over the range of EEPROM bytes defined by the beginning and ending DATA directives.

A technique was also introduced for using a DATA directive to set aside one byte for setting the program mode. Each time the program starts, an initialization routine reads the byte, adds one to it, and replaces the old value in EEPROM with the modified value. Each time the program is restarted by pressing and releasing the board's Reset button, the program can use the new value in EEPROM to select between different modes. For toggling a feature in the program on and off, an IF...THEN statement that examines whether the remainder of the value divided by two is zero is used. This makes it possible to start and stop datalogging without being connected to the computer.

Accelerometer applications with datalogging included RC car acceleration, skateboard trick measurements, and bicycle wheel measurements. Each of these used a program that was a variation of the remote datalogging program introduced in Activity #4. The data displayed in the Debug Terminal was shaded, copied, and pasted into text files. The text files were then imported into a spreadsheet program and graphed. The graphs were analyzed to determine to examine accelerations, tilts, and angles involved RC car, skateboard, bicycle wheel motions.

Questions

1. What three pieces of information do you need to measure the height of a building from a distance?
2. How many bytes does this DATA directive set aside? DATA (50)
3. What's the difference between DATA (100) and DATA 20 (100)?

4. What does the command WRITE eeIndex, 100 do? What function does it serve in EepromDataStorage.bs2?
5. What's wrong with this command? WRITE eeIndex, 1000. How can you fix it?
6. What data directive is used for mode selection with the Reset button in this chapter's activities?
7. What other directives and commands have to be present for IF value // 2 = 0 THEN... to make it possible to toggle program modes with your board's Reset button?
8. How does IF value // 2 = 0 THEN... allow you to toggle between different program modes?
9. What does the peizospeaker do in DatalogAcceleration.bs2?
10. What does the SELECT...CASE code block in DatalogAcceleration.bs2 do?
11. How can you modify a DATA directive to make it set aside more values?
12. What's the storage capacity difference between datalogging x and y byte values and datalogging brad angle measurements?
13. How does forward acceleration differ from forward deceleration?
14. How is backward acceleration similar to forward deceleration?
15. When driving in circles at a constant velocity and radius, what direction is the acceleration?
16. If you drive around in circles in the opposite direction, how does the acceleration change and how does it stay the same?
17. How does the datalogging program that measures a skateboarder's ollie differ from the program that measures RC car motions? How are they similar?
18. What changes with speed in bicycle rotation measurements?

Exercises

1. The top of a building was sighted to be 75° from a vantage point 15 m from the building and 1 m from the ground. How tall is the building?
2. A tower was sighted to be 20° from a vantage point 100 m from the its base and 2 m from the ground. How tall is the tower?
3. Write a pair of DATA directives that reserve 1501 bytes. Use symbol names.
4. Write a FOR...NEXT loop that retrieves and displays 1500 bytes. Assume your DATA directive Symbol names are Begin and Quit.
5. Write a FOR...NEXT loop that retrieves 751 words. Assume that your DATA directive Symbol names are StartData and EndData.
6. Write a segment of code that will allow you to toggle between three different program modes with your Board's Reset button. Hint: use SELECT (value // 3), and then use CASE statements for 0, 1, and 2.

7. Modify a block of code in DatalogAcceleration.bs2 so that its countdown is five seconds.
8. Extend the Your Turn section of Activity #4 so that the program stores degree measurements from 0 to 359.

Projects

1. Use Google to find the slope above which snow is likely to avalanche. Prototype a measuring device that warns you if a slope is too steep.
2. Complete the bicycle wheel project in Activity #7.
3. Use the datalogging and programming techniques introduced in this chapter to develop a pedometer. This is a device that calculates the distance you have walked based on how many steps you have taken.
4. Figure_Next shows an RC airplane. Review servo control in What's a Microcontroller, then write a program that causes the wing flaps to correct for tilt to keep the plane level.

<<<<Rich, we need a picture of the Parallax airplane in the purple room>>>>>

17.

Chapter #7: LCD Bar Graphs for Distance and Tilt

2

Defining and displaying custom characters with the Parallax Serial LCD was introduced in Chapter #1, Activity #4. This chapter introduces some more custom character techniques, and then applies them to bar graph displays. These displays will indicate the distance of an object from the Ping))) ultrasonic sensor and the tilt of the Memsic 2125 Dual Axis Accelerometer.

ACTIVITY #1: CUSTOM CHARACTER SWAPPING

The Parallax Serial LCD can display up to eight custom characters at any given time. However, there can be many more than eight custom characters in the application because custom characters can be defined and redefined as needed. The only limitation is that only eight can be displayed at any given time, and eight is ample for most projects.

The place where you can define and store more than eight custom characters is in the part of the BASIC Stamp's EEPROM memory that does not used for program storage. Since PBASIC programs rarely fill the BASIC Stamp's entire EEPROM memory, there is typically room for all the custom characters an application might need.

One important trick to getting for conserving custom characters is using just one of the LCD's eight custom character definitions to display a sequence of custom characters that are stored in the BASIC Stamp's EEPROM. This is especially useful for animation, but it will also be important for bar graph displays. This activity provides an animation example.

From EEPROM Storage to LCD Character Memory

The next example program will demonstrate a convenient way to store custom characters in the BASIC Stamp's EEPROM. Two of the program's fifteen custom character definitions are shown below. Each custom character gets a unique Symbol name, like **Char0**, **Char1**, **Char2**, and so on, up through **Char14**. Each of these Symbol names represent the EEPROM address of the first byte in the **DATA** directive. The subroutine that transfers the custom character definition from EEPROM to the LCD's custom character memory uses these Symbols as a reference point for reading the bytes from EEPROM. After reading each byte from EEPROM, the subroutine sends it to the serial LCD.