In Proceedings of the National Conference on Educational Robotics, Norman, Oklahoma, June 28 – July 1, 2003.

Beyond Botball: Science Experiments with your Handy Board

Holly A. Yanco and Karl R. Wurst

Holly A. Yanco
Computer Science Department
University of Massachusetts Lowell
One University Avenue, Olsen Hall
Lowell, MA 01854
holly@cs.uml.edu

Karl R. Wurst
Computer Science Department
Worcester State College
486 Chandler Street
Worcester, MA 01602
kwurst@worcester.edu

Abstract

After the Botball program ends, your Handy Board and sensors can be used to perform science experiments or other data collection experiments. In this paper, we discuss different experiments, providing code for each. Arrays and structures in the C programming language are introduced and used in our examples.

1 Introduction

Botball 2003 may be over, but don't put that kit into storage! You can use the Handy Board and sensors in your kit to perform data collection experiments. You could measure the light levels near a plant, recording significant changes, in order to study plant growth. You could study traffic patterns by logging cars traveling in particular directions or by logging student travel at a corridor intersection between classes. You can test people's reaction time by measuring how long it takes them to hit a button after a beep is sounded or a message is written on the screen; you could also develop ways to simulate impairment during this experiment, perhaps by introducing noise into the environment.

In this paper, we'll show you how to measure, store, and analyze reaction times, entries into a room, the number of people in a room, and the light level near plants in a growing experiment.

2 Measuring Reaction Times

We can measure how long it takes a person to press a button after a beep is heard. So that the time of the beep is not predictable, we can have the beep come at a random time.

The code for this experiment is given in Figure 1. To run the experiment, put a switch sensor in digital port 15.

```
// Reaction Time Tester

// Beeps after 1-4 seconds and records how long it takes for
// the user to press the switch connected to digital(15)

void main()
{
    // play the game until the user holds down STOP
    while(!stop_button())
    {

        printf("Finger on button& press START\n");
        while(!start_button())
          ;  // do nothing until the user presses START

        printf("Press button on BEEP...\n");
        // wait 1000ms + 0-3000ms (1-4 secs)
        // random() returns an int, but msleep() needs a long
        // so cast it (long)
        msleep( (long)(random(3000)+1000) );
        beep();  // hit the button NOW!

        reset_system_time();  // reset the clock to zero
        while(!digital(15))
          ;  // wait for the user to hit the button

        // print the reaction time
        printf("%d ms START-again STOP-quit\n", mseconds());

        // to play again or quit
        while(!start_button() && !stop_button())
          ;  // wait for the user to press START or STOP
        while(start_button())
          ;  // wait for the user to let go

    } // end while
    printf("Thanks for playing\n");

}  //end main()
```

Figure 1: Code for measuring reaction times.

Each time you record a reaction time, it will be written to the screen on the Handy Board. If you wanted to collect many data points, you would need to write each one down on a piece of paper. You can certainly do this, but there is a better way: you can store your data points in a C data structure called an array.

3 Storing Your Data in an Array

We can make the Handy Board store many instances of the same data type in a data structure called an array. An array allows you to store multiple data points in a single variable name. For example,

```
int data[100];
```

creates an array that will hold 100 integers (whole numbers). Remember that all variable declarations go at the beginning of a function (either `main` or another user defined function). Since the Handy Board has a relatively small memory (32K, but only about 16K is left after the firmware is loaded), you can't store much more than 2500 integers in an array (or in total across all arrays, if you have more than one).

Once you've declared an array, you can use it in your code. As an example, let's look at how we could initialize the entire array to have values of 0 in every slot.[1]

```
for (i = 0; i< 100; i++)
   data[i] = 0;
```

One very important point to remember with arrays in C: the array index starts at 0. This means that the first slot you fill in will be slot 0, and the number of the last slot that you can put data in is one less than the number of slots (in the example above, 99).

3.1 Recording Switch Hits in an Array

The code in Figure 2 stores the times that a switch is pressed. To run the experiment, put a switch sensor in digital port 15.

```
// Button Press Logger

// Every time the switch connected to digital(15) is pressed
// the time is stored in an array
// Once the program ends, the data array can be uploaded to the
// PC using the Upload Array option in the Tools menu of IC 4

// record up to 100 presses
// using a constant allows us to make one change in the code
// later if we want to make a bigger or smaller array -- it's a
// good habit to learn
#define ARRAY_SIZE 100

// the array must be declared outside of main() to be uploadable
long data[ARRAY_SIZE];

void main()
{
    int pressNumber = 0;
    int i = 0;
```

[1] You'll find that IC4 automatically initializes arrays of type int or long to have all zeros. However, this is not true in standard C, so you should get used to initializing your arrays. Also, IC4 does not initialize arrays of other types, such as floats.

```
    // initialize the array
    for (i = 0; i< ARRAY_SIZE; i++)
       data[i] = 0;

    // wait for the user to press START
    printf("Press START\n");
    while(!start_button())
      ;  // do nothing until the user presses START

    reset_system_time();  // reset the clock to zero
    printf("Hold STOP, press button to quit\n");

    // record button presses until the user holds down STOP
    // or we run out of space to record times
    while(!stop_button() && pressNumber < ARRAY_SIZE)      {

        while(!digital(15) && !stop_button())
          ;  // wait for the user to hit the button

        if (!stop_button())
        {
            // record and print the time
            data[pressNumber] = mseconds();
            printf("%d ms\n", data[pressNumber]);
            pressNumber++;
        }

        while(digital(15) && !stop_button())
          ;  // wait for the user to let go of the button
             // (debounce)


    } // end while

    if (pressNumber < ARRAY_SIZE)
      printf("Done. Upload data in IC4\n");
    else
      // we used up all of the array slots
      // you can still upload the array into IC4
      printf("Ran out of space.\n");

}  //end main()
```

Figure 2: Code for storing times when the button is pressed into an array.

3.2 Uploading Arrays

Once you've run a program that stores data in an array, you can upload the data in that array to your computer. Connect your Handy Board to the computer, and then click on "Upload array" in the Tools menu of IC4. Select the name of the array that you wish to upload. Then select where you want to save the data; select "Save to CSV" to save to a Microsoft Excel file for data analysis. Once you've saved your data to an Excel spreadsheet, you can graph it or perform data analysis. Anything you can do in Excel with numbers can be done with the data you import from IC.

3.3 Storing Reaction Times

Now that we've seen how to store data points in an array, we can write a program that will log times from our reaction time experiment in an array. If you want to try this on your own, you can modify the code from Figure 1 to store the reaction times in an array (using pieces of the code in Figure 2). Stop reading now if you want to do this on your own first. The code is next, in Figure 3. To run the experiment, plug a switch sensor into digital port 15.

```
// Reaction Time Tester with Logging

// Beeps after 1-4 seconds and records how long it takes for
// the user to press the switch connected to digital(15)
// Stores the reaction times in an array
// Once the program ends, the data array can be uploaded to the
// PC using the Upload Array option in the Tools menu of IC 4

// record up to 100 plays
#define ARRAY_SIZE 100

// the array must be declared outside of main() to be uploadable
long data[ARRAY_SIZE];

void main()
{
    int playNumber = 0;
    int i = 0;

    // initialize values in the array to 0
    // IC4 will do this automatically for you with arrays of
    // type int or long, but this isn't true in C in general
    // It's a good practice to initialize your arrays
    for (i=0; i<ARRAY_SIZE; i++)
      data[i] = 0;

    // play the game until the user holds down STOP
    // or we run out of space to record times
    while(!stop_button() && playNumber < ARRAY_SIZE)      {

        printf("Finger on button& press START\n");
        while(!start_button())
          ;  // do nothing until the user presses START

        printf("Press button on BEEP...\n");
        // wait 1000ms + 0-3000ms (1-4 secs)
        // random() returns an int, but msleep() needs a long
        // so cast it (long)
        msleep( (long)(random(3000)+1000) );
        beep();  // hit the button NOW!

        reset_system_time();  // reset the clock to zero
        while(!digital(15))
```

```
          ;  // wait for the user to hit the button

        // record and print the reaction time
        data[playNumber] = mseconds();
        printf("%d ms START-again STOP-quit\n",
               data[playNumber]);
        playNumber++;

        // to play again
        // to play again or quit
        while(!start_button() && !stop_button())
          ;  // wait for the user to press START or STOP
        while(start_button())
          ;  // wait for the user to let go

    } // end while

    if (playNumber < ARRAY_SIZE)
      printf("Done. Thanks for playing\n");
    else
      // all slots in the array have been filled
      // can still upload data to IC4
      printf("Ran out of space.\n");

}  //end main()
```

Figure 3: Logging reaction times in an array.

4 Counting Entries and Exits from a Room

By putting an ET sensor on one side of a doorway, pointed across the doorway, we can measure when people come into a room. (Actually, with just one sensor, we can only count when someone crosses the beam. We'd need two sensors to measure whether someone has entered or exited the room.) In this section, we present the code for using one ET sensor in Figure 4, then using two sensors in Figure 5. To run the entry logger, plug an ET sensor into analog port 16.

```
// Entry Logger

// Uses an "ET" distance sensor to detect people walking through
// a door.  The program first records the distance across an
// empty doorway, then anytime the distance is less than that,
// the time is recorded in an array.  Once the program ends, the
// data array can be uploaded to the PC using the Upload Array
// option in the Tools menu of IC 4

// record up to 100 presses
#define ARRAY_SIZE 100

// some "wiggle room" in the distance
#define EPSILON 20

// the array must be declared outside of main() to be uploadable
// using seconds() returns a float, and will allow us to store
// longer times
float data[ARRAY_SIZE];

void main()
{
    int entryNumber = 0;
    int emptyDoorway;
    int i;

    // initialize the array of floats
    for (i=0; i<ARRAY_SIZE; i++)
      data[i] = 0;

    // wait for the user to press START
    printf("Position sensor, press START\n");
    while(!start_button())
      ;  // do nothing until the user presses START

    reset_system_time();  // reset the clock to zero

    // measure the distance across empty doorway
    emptyDoorway = analog(16);

    printf("Hold STOP, press button to quit\n");

    // record entries until the user holds down STOP
    // or we run out of space to record times
    while(!stop_button() && entryNumber < ARRAY_SIZE)
    {
        while(analog(16)<=emptyDoorway + EPSILON)
          ;  // wait for someone to walk through the door
        while(analog(16)>emptyDoorway + EPSILON)
          ;  // wait for them to get out of the door

        // record and print the reaction time
        data[entryNumber] = seconds();
        printf("%f sec\n", data[entryNumber]);
        entryNumber++;

    } // end while
    if (entryNumber < ARRAY_SIZE)
      printf("Done. Upload data in IC4\n");
    else
      printf("Ran out of space.\n");

}  //end main()
```

Figure 4: Code for measuring when the beam of an ET sensor across a doorway gets a shorter reading when a person goes through the doorway.

Now that you've seen the code for using one ET sensor, try using two sensors so that you can record when someone enters or exits. You'll need two arrays, one for entry times and one for exit times. The code is in Figure 5, but try it on your own before reading the code below. To run the code below, plug ET sensors into analog ports 16 and 17. The sensor in analog port 16 is the sensor at the front of the door (crossed first to log an entry), and the sensor in analog port 17 is the sensor at the back of the door (crossed first to log an exit).

```
// Entry Logger

// Uses an "ET" distance sensor to detect people walking through
// a door.  The program first records the distance across an
// empty doorway, then anytime the distance is less than that,
// the time is recorded in an array.  By using two sensors, we
// can determine if the person has come into the room or has
// exited the room.  Once the program ends, the data array can be
// uploaded to the PC using the Upload Array option in the Tools
// menu of IC 4

// record up to 100 presses
#define ARRAY_SIZE 100

// some "wiggle room" in the distance
#define EPSILON 20
#define FRONT 16
#define BACK 17

// the arrays must be declared outside of main() to be uploadable
float in[ARRAY_SIZE];
float out[ARRAY_SIZE];

void main()
{
    int i;
    int inNumber = 0;
    int outNumber = 0;

    int emptyFrontDoor;
    int emptyBackDoor;

    // initialize the arrays
    for(i = 0; i < ARRAY_SIZE; i++)
    {
        in[i] = 0.0;
        out[i] = 0.0;
    }

    // wait for the user to press START
    printf("Position sensors, press START\n");

    while(!start_button())
      ;  // do nothing until the user presses START

    reset_system_time();  // reset the clock to zero

    // measure the distance across empty doorway for both sensors
    emptyFrontDoor = analog(FRONT);
    emptyBackDoor = analog(BACK);

    printf("Hold STOP, press button to quit\n");
    // record entries until the user holds down STOP
    // or we run out of space to record times
    while(!stop_button() &&
          inNumber < ARRAY_SIZE &&
          outNumber < ARRAY_SIZE)
    {
        //Leaving - back sensor triggered before front
        if( analog(BACK) >= emptyBackDoor + EPSILON)
        {
            while(analog(FRONT)< emptyFrontDoor + EPSILON)
              ;  // wait for someone to walk through the door

            out[outNumber] = seconds();
            printf("Out: %f ms\n", out[outNumber]);
            outNumber++;

            while(analog(FRONT) >= emptyFrontDoor + EPSILON)
              ;
        }

        //Entering - front sensor triggered before back
        if( analog(FRONT) >= emptyFrontDoor + EPSILON)
        {
            while(analog(BACK)<emptyBackDoor + EPSILON)
              ;  // wait for them to get out of the door

            in[inNumber] = seconds();
            printf("In: %f ms\n", in[inNumber]);
            inNumber++;

            while(analog(BACK) >= emptyBackDoor + EPSILON)
              ;
        }

    } // end while

    if (inNumber < ARRAY_SIZE  && outNumber < ARRAY_SIZE)
      printf("Done. Upload data in IC4\n");
    else
      printf("Ran out of space.\n");
}  //end main()
```

Figure 5: Code for using two ET sensors to measure entries and exits into a room.

Once you run your program that logs entry and exit times, you can upload the data. Unfortunately, you can only upload one array at a time to an Excel file. If you wanted the data from two arrays in one Excel file, you'd need to do some cutting and pasting. We'll see in Section 5 how we can use arrays of structures so that you can upload all of your data at once.

You could also modify the code to keep track of the number of people in the room. You'd just need to add a counter, which you'd update in the part of the code that logs an entry or exit. Of course, there are ways to fool the system by jumping over sensors or waving your hands by the sensors. But if the sensors aren't tricked, you should be able to keep a fairly accurate count of how many people are in the room. (This is left as an exercise, so code is not provided.)

5 Measuring Light Levels for Plant Growing Experiments

In this example, we'll see how to use an array of structures so that we can upload all of our data at once (skipping a step of cutting and pasting). A structure is a data type in C that can hold many variables of different types.

Here's an example of a structure definition:

```
struct observation {
    float time;
    int light_level;
};
```

In the example, `observation` is the name of the new structure data type that we've created. This new structure has two parts inside it: a float variable called `time` and an integer variable called `light_level`.

Now we can use our new data type to declare an array of that type:

```
struct observation data[100];
```

The data type for this array is `struct observation`. In earlier examples, we saw arrays of type `int`, `long` and `float`.

Once you've declared an array of structures, how do you use it? You still need to reference a particular slot of the array (which still starts counting at 0), but then you'll also need to reference with variable in the structure that you want to use. We'll glue these two pieces of information together with a dot (or period). For example,

```
data[0].time = 0.0;
data[0].light_level = 0;
```

The code in Figure 6 shows the use of an array of structures for logging changes in light levels. We only log changes that are large enough to be considered significant; without checking, we'd record very minor fluctuations and fill up our array very quickly. To run this experiment, plug a light sensor into analog port 6.

```
// Light level data collection program

// Light sensor is plugged into analog(6)
// Every time the light level changes by some amount (epsilon)
// the light level and the time will be recorded.
// Once the program ends, the data array can be uploaded to the
// PC using the Upload Array option in the Tools menu of IC 4

// difference in light level needed to record new observation
#define EPSILON 5

// record up to 100 presses
#define ARRAY_SIZE 100

// structure to store a light reading observation
// stores the time of the observation and the light level
struct observation {
    float time;
    int light_level;
};

// declare our array of structures
struct observation data[ARRAY_SIZE];

void main()
{
    int i;

    // initialize values in the structure
    for (i=0; i<ARRAY_SIZE; i++)
    {
        observation[i].time = 0.0;
        observation[i].light_level = 0;
    }

    // give the user a chance to set-up the equipment
    // before starting to take readings
    printf("Position light sensor and press Start\n");
    while (!start_button())
      ; //do nothing while we wait for the user to press Start

    // record the light reading at start
    reset_system_time(); // reset the system clock to zero
    i = 0;  // reset i to 0, since we used it above
    data[i].time = seconds(); // save the time
    data[i].light_level = analog(6); //save the light reading
    i = i + 1;

    // record light changes until the user holds down STOP
    // or we run out of space to record data
    while(i<ARRAY_SIZE && !stop_button() )
      {
        // if the light level has increased or decreased by
        // at least the value of epsilon...
        if ((analog(6) > data[i-1].light_level + EPSILON)
             || (analog(6) < data[i-1].light_level - EPSILON))
        {
            data[i].time = mseconds();  // record the time
            data[i].light_level = analog(6); //record light level
            printf("New: light %d time %f\n",
                   data[i].light_level, data[i].time);
            i = i + 1;
            beep();
        } // end if

    } // end while

    if (i < ARRAY_SIZE)
      printf("Done. Upload data in IC4\n");
    else
      printf("Ran out of space.\n");
}
```

Figure 6: Code for the light logging experiment, which gives an example of an array of structures.

6 Conclusion

Now that you've seen how to store data in arrays and in arrays of structures, you can design new experiments using the sensors in your kits. You can also add other types of sensors to your kits.[2]

All of the code in this paper is available for downloading at http://www.cs.uml.edu/k12/beyond-botball.html. If you design new experiments that you'd like added to the web page, please send an e-mail to holly@cs.uml.edu.

Acknowledgements

Mike Baker and Brenden Keyes of UMass Lowell helped to write and revise some of the code in this paper.

[2] A good place to look for other sensor types is www.acroname.com. You will need to do some soldering for a connector to your board, but most of the sensors at Acroname have directions for using them with the Handy Board and include the appropriate drivers.