ME 3200 Mechatronics I Laboratory

Lab 5: Introduction to the Handy Board

Introduction

The purpose of this lab is to give you experience with microcontrollers. Their small size, relatively inexpensive price, and ever increasing speed makes them ideal for control applications where a large computer is not necessary or the space is not available. Microcontrollers are now used to control automobile systems (engines, transmissions, braking, collision avoidance, traction control, etc), microwave ovens, robotic systems, and many other systems. The microcontroller used in this laboratory and in your robotics project is the Handy Board. The following introduction and laboratory procedure are presented to help students become acquainted with the basic functions and operation of the Handy Board.

The Handy Board

The Handy Board is a small single board computer developed by Fred Martin of the MIT Media Laboratories. The main features of the Handy Board are a Motorola 68HC11 microprocessor chip, four DC motor outputs, seven 8-bit analog inputs, nine digital inputs, a 16x2 character LCD screen, and an internal 9.6 Volt NiCad battery pack. The Handy Board is programmed using Interactive C, a C programming language, developed by Randy Sargent of MIT. A diagram of the Handy Board depicting its main features is shown below in Figure 1. The following sections describe some of the basic steps needed to prepare the Handy Board for programming and the more useful pre-programmed functions of the Handy Board. More detailed information about the capabilities of the Handy Board and its programming interface can be found at the Handy Board web site (http://handyboard.com) and on the mechatonics class webpage under project handouts. The *Handy Board Technical Reference* and the *Interactive C for the Handy Board Manual* in the Docs section of the Handy Board website are of particular value for understanding the Handy Board and using it in robotic control applications.

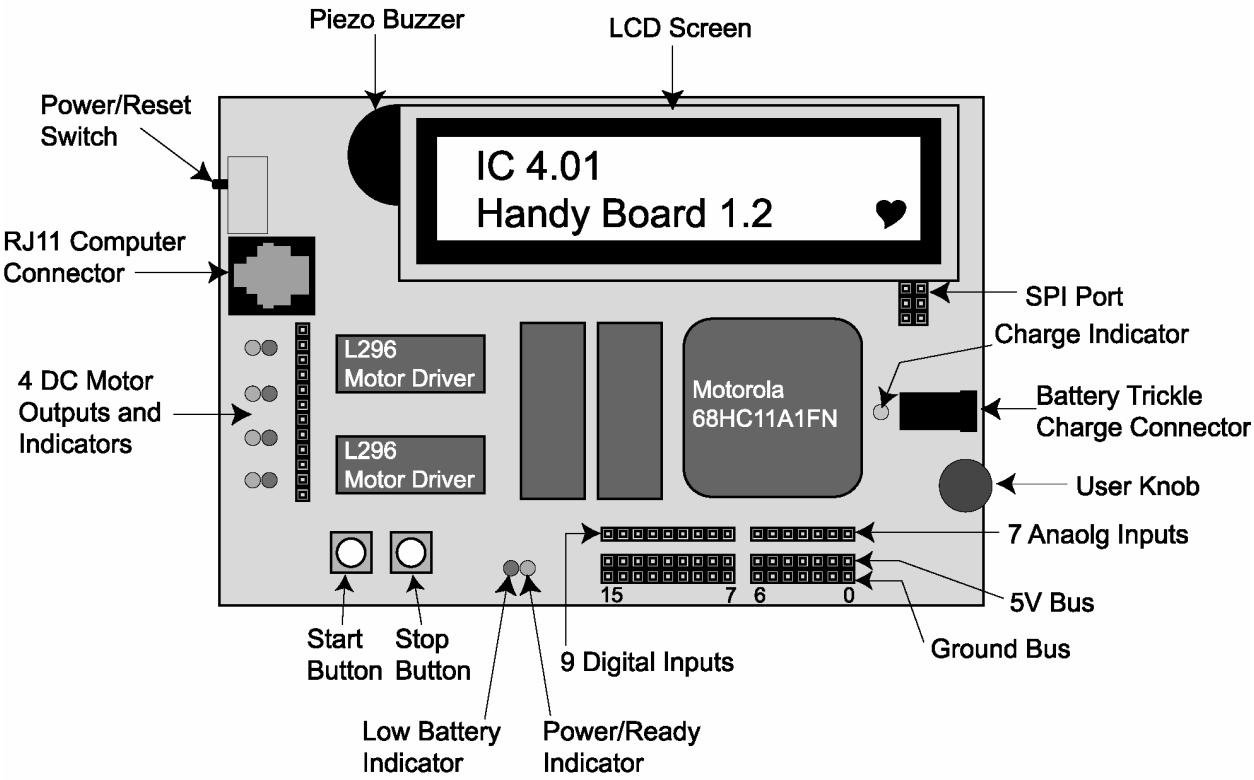

Figure 1: A diagram of the Handy Board's major features.

Preparing the Handy Board for Use

The following instructions are for use with Interactive C 4.01 and a Handy Board without an expansion board. Before you begin using the Handy Board, make sure that you have the following:

- A Handy Board
- An RJ11 modular cable (looks like a telephone cable)
- A serial interface and battery charger board
- A 12 Volt, 500mA DC transformer.
- Interactive C 4.01 installed on a host computer
- A serial cable

The procedure to prepare the Handy Board for computer interfacing and programming is as follows:

1. Begin charging the Handy Board by inserting one end of the DC transformer into an electrical outlet and the other into the trickle charger connector on the Serial Interface & Battery Charger Board. This step is always recommended as it will maintain the charge on the Handy Board's battery while you are working on it, but is not necessary if the batteries are fully charged.

2. Connect the Handy Board to the host computer by

   a. Connecting one end of the RJ11 modular cable into the computer connector port on the Handy Board (see Figure 1) and the other end into the serial interface and battery charger board.

   b. Using the Serial Cable, connect the serial interface board to the serial port of the computer.

   Note: If a connection cannot be made between the Handy Board and the computer, a loose serial connection or faulty RJ11 cable are likely causes.

3. Turn on the Handy Board and start Interactive C on the host computer by double clicking the icon on the desktop or through the Start menu.

If the Handy Board has not been used for some time, the internal memory has been corrupted, or the internal battery doesn't have enough charge to retain the system memory, it may be necessary to reload the Handy Board's operating system (called the firmware). An error in the firmware is indicated when the LCD screen displays a line of boxes, Interactive C can't communicate with the Handy Board, the heart in the bottom right corner of the LCD screen is not "beating," or if some other kind of error persists while operating the Handy Board. If the Handy Board's LCD display resembles Figure 2 and the heart is beating, you can proceed to the section entitled "Running the Handy Board from the PC"; otherwise, load the firmware according to the following instructions.

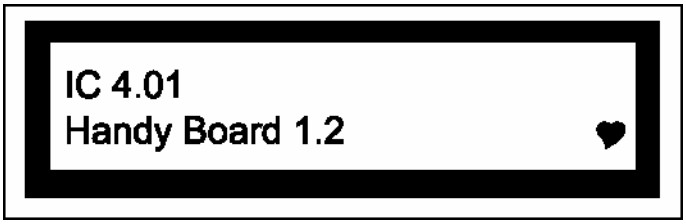

Figure 2: Heathy Handy Board start-up screen.

Loading the Operating System (Firmware)

To reload the firmware, do the following:

1. Connect the Handy Board to the host computer using the serial interface card, power cord, and RJ11 cable as directed in the previous section. If Interactive C is already open select Tools > Download Firmware from the window menu; otherwise, start Interactive C and select *Handy Board (no expansion)* from the opening window.

2. Select the communication port from the next window and press the "Download Firmware" button.

3. Follow the prompts for the next five steps selecting "Next" between each window. After the fifth step a window displaying the progress of the download appears and the Handy Board will beep when the firmware has loaded successfully.

   Note: After the Handy Board is restarted, the LCD screen should look like Figure 2 with a beating heart icon in the bottom right corner. This screen tells you that the firmware loaded successfully and that the Handy Board is ready for programming.

Upon completion of the preceding steps, the Handy Board is ready for programming in interactive mode described below.

Running the Handy Board directly from the PC (Interactive Mode)

If the firmware is correctly loaded, you can place the Handy Board in interactive mode by turning off the Handy Board and pressing the START button while switching the Handy Board back on (thus preventing the execution of the `main()` function on start-up). Interactive mode enables the user to send commands directly to the Handy Board from the command line of Interactive C. For example, typing "`printf("Hello World!\n");`" at the command prompt and pressing enter will display the text "Hello World!" on the LCD screen. Executing Handy Board commands in interactive mode is an effective way to explore unfamiliar functions or commands. This mode is also a useful tool for troubleshooting programs by executing the code one line at a time.

Downloading Programs to the Handy Board

With the Handy Board in interactive mode, you are now ready to write a program and download it to the Handy Board. Handy Board program files are created using the text editor included in Interactive C or Windows Notepad. Program files must be saved with a ".c" or a ".ic" file extension (i.e. *sample.c*). For example, suppose that you created a program called *download.c* with the text editor in Interactive C that contained the following text:

```
void main(){
   printf("You now know how to download\n");
   }
```

After the program code has been entered and saved in Interactive C, pressing the download button will perform two tasks. First, it compiles the C code of *download.c* and, second, it loads the code into the memory of the Handy Board. If the program code has syntax errors, an error message is displayed in the Interactive C window and the file is not transferred to the Handy Board. Once the program is successfully downloaded, reset the Handy Board by turning it off and then back on to execute the `main()` program: then, in the case of this example, the LCD screen should read "You now know how to download". Each time the board is turned off and then back on, the `main()` program will be executed until it is replaced by another `main()` program or the Handy Board is reset in interactive mode or download mode.

Programs can be named any name that Windows allows but must have a "**.c" or ".ic"** DOS file extension. Multiple programs can be loaded (up to 32K) onto the Handy Board, but only one may contain a `main()` function; the `main()` function is always executed after a hard reset and must be present for the Handy Board to function as desired. Other functions may be called either from the `main()` function, other functions, or while in interactive

mode. This feature allows the Handy Board to perform multitasking, run several subroutines simultaneously, and allows code to be developed in modules and loaded separately using a list file. A list file is a text file that contains a list of the programs (with file extension) that you want loaded onto the Handy Board in a single download; it must have a ".lis" file extension. Downloading the list file to the Handy Board compiles and loads all the listed files at once.

Handy Board Functions

The Handy Board can be controlled using a wide variety of commands. These commands can be given directly in the interactive mode or used as function calls in C programs downloaded to the Handy Board. The most useful method, however, is creating programs that are loaded directly onto the Handy Board for execution without interaction with the PC host. A brief description of the more useful functions and how to use them are given below. For a more complete listing of the available functions, see the *Interactive C for the Handy Board Manual* available on the Handy Board web site.

`printf(format-string, [arg-1], . . . , [arg-N])`

Printing information to the screen can be a useful debugging tool, but it also slows down program execution significantly. Suppose the value of parameter **x** is needed for debugging your code. The `printf()` command would be used as follows:

```
printf("Value is %d\n",x);
```

The character `\n` at the end of the string is an *end-of-line* character. When an *end-of-line* character is printed, the LCD screen will be cleared when a subsequent `printf` command is issued. The special form `%d` is used to format the printing of an integer in base ten form. A listing of other formats is given below in Table 1.

Table 1: Number printing format commands.

Format Command	Data Type	Description
%d	int	base ten number
%x	int	hexadecimal number
%f	float	floating point (decimal) number
%s	char	character array (string)

`motor(int m, int p)`

This function turns on motor port *m* at power level *p*. There are four motor ports on the Handy Board, labeled Motor-0, Motor-1, Motor-2, Motor-3 (see Figure 1 and Figure 3 for location). The power level argument can range from 100 for full forward to –100 for full reverse; however, there are actually only 15 motor speeds: seven forward, seven reverse, and stop (Table 2).

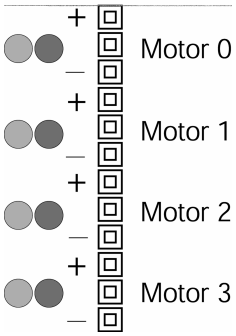

Figure 3: Motor port designation. Note that the middle pin for each motor connector is not used.

Table 2: Motor output levels.

Power Level *	Forward Motor Speed
0-10	off
11-24	1
25-38	2
39-52	3
53-66	4
67-80	5
81-94	6
94-100	7

**A negative power level corresponds to the appropriate Motor Speed, but in reverse.*

The motor driver chips (Texas Instruments SN754410NE chips) supply a five volt pulse width modulation (PWM) output with a maximum current output of one ampere. A higher current demand is normally required for DC motors under load, and an external power source and motor driver circuit should be considered. The motor drivers use PWM outputs that switch the full voltage on and off at a high frequency to control the motor speed; the ratio of on to off time for each cycle determines the power level of the motor command. The effective voltage seen by a motor (which has a slow response time compared to the frequency of the PWM) is proportional to the average time spent in the high state (full voltage). Note that if you measure the output with an oscilloscope, you will see a high frequency signal.

`alloff() or ao()`

The `alloff()` command turns off all motors simultaneously. `ao()` is a recognized abbreviation which is helpful when sending motor commands in interactive mode.

`off(int m)`

This command turns off motor m.

`digital(int p)`

This function returns the value of a digital sensor connected to one of the nine digital input ports (ports 7-15 of Figure 1). Each port is an active low digital input: meaning, if the port is shorted to ground, it returns true (1) and if more than 2.5 volts are applied or the port is not connected, the value returned is false (0).

`analog(int p)`

The `analog(int p)` function reads the voltage level on the specified analog input port (ports 0-6 of Figure 1) on the analog input bus, which is connected to an 8-bit analog-to-digital converter. The value returned is an integer between 0 and 255. If five volts or more is supplied to an analog port, `p`, or it is left open, the `analog(int p)` function returns a value of 255, while a short to ground returns a value of 0.

Refer to Figure 1 for the location of the analog and digital ports. Note that these ports are the top of the three strips in that area, and are numbered from right to left. The bottom strip is ground, and the middle strip is a +5 volt supply voltage that can be used to power sensors. The top strip is for the sensor signal (this is the value that will be read by the `analog()` and `digital()` functions). All the analog and digital input ports are connected to the five-volt bus through an internal 47-kΩ pull-up resistor. This internal resistor can change the behavior of some sensors. Specifically, a voltage divider is created when one lead of an analog sensor is inserted into an analog input and the other into ground (refer to the *Computer Data Collection* lab for a description of a voltage divider). The voltage across the internal pull-up resistor is the voltage read by the eight-bit ADC of the Handy Board.

`knob()`

This function returns the position of the user knob (see Figure 1) as a value between 0 and 255. The knob is a potentiometer whose wiper is connected to an 8-bit analog-to-digital converter. This potentiometer divides the voltage between the five-volt bus and ground.

`start_button()`

The `start_button()` function returns the digital state of the button labeled "Start" (see Figure 1): 1 if pressed and 0 if released.

`stop_button()`

The `stop_button()` function returns the digital state the button labeled "Stop" (see Figure 1): 1 if pressed and 0 if released.

`start_process(function-call(...),[TICKS],[STACK-SIZE])`

The `start_process()` function is used for multi-tasking. It allows more than one function to be running at a time by allotting processor time to all running functions. When the optional arguments (the bracketed variables above) are omitted the processor automatically manages the number of clock cycles (`TICKS`) and the amount of memory (`STACK-SIZE`) allotted to each program. Multitasking can be useful if you would like to have a timer or a sensor checking algorithm running while another process determines and sends motor commands. Each call to `start_process()` returns an integer which identifies the process in memory; this integer is referred to as the process ID or the PID. If this value is stored in a global variable then it can be accessed from anywhere in your code.

`kill_process(int PID)`

The `kill_process()` function is used to end a process that was invoked using `start_process()` before the process reaches its natural end. However, the process id of the function to be killed must be stored in an accessible variable. The `kill_process()` function could be used, for example, to kill a timer if a task is completed before the timer expires.

`kill_all()`

The `kill_all()` function is similar to the `kill_process()` function. It terminates execution of all processes that were invoked using the `start_process()` function and is handy to use in interactive mode.

Additional Programming Information

Digital Outputs

There are a limited number of ports on the Handy Board that may be used as digital outputs (digital port 9, TOC 3, and the serial peripheral interface). Each of these ports can be controlled to provide a voltage of 0 or 5 volts (off or on, low or high). These ports, however, are only capable of supplying a few milliamps of current. Transistors, or other means, must be used to amplify the signal if it is used to supply any significant power. The digital output ports are controlled using direct memory location access; this process is described in the FAQ section at the end of the *Handy Board Technical Reference*.

Type Conversions

Unlike ANSI C and other programming languages, Interactive C does not handle dynamic type conversions. This means that the expression "`5*1.5`" (an integer times a floating point number) causes a syntax error in Interactive C. The correct syntax for this expression is either "`5.0*1.5`" (returning a floating point number) or "`5*(int)1.5`" (returning an integer). Other common syntax errors include:

- `int a = 3.0;` (float assigned to an integer)
- `float b = 4;` (integer assigned to a float)
- `sleep(5);` (integer used instead of a float)
- and `6^2;` (the "^" function is only defined for floats).

If you need to convert one data type to another Interactive C supports the use of explicit conversions. The following are a list of common type conversions and their outputs:

- (int)5.0 = 5
- (float)6 = 6.0
- (int)(6.0^2.0) = 36

Keep these conversion options in mind when debugging code which seems to be otherwise correct.

Other Handy Board Functions

There are many other functions that might be useful in programming the Handy Board. A complete listing of the Handy Board functions is given below; for a complete description of these functions, see the *Interactive C Manual for the Handy Board* on the Handy Board web site or the class website under handouts. This manual also contains a fundamental C programming tutorial for those who are not familiar with C programming and additional information about the Handy Board, ranging from schematics, modifications, new functions, and much more.

Another section of this document that might be useful is the section that describes binary program files. These files are more difficult to create, but are also much more efficient than compiled C code files. Therefore, rather than using an existing, efficient C program, you could convert that program to a binary program file to increase the speed at which the algorithm executes.

LCD Screen Printing

```
printf(format-string, [arg-1], . . . , [arg-N])
```

DC Motors

```
void fd(int m)
void bk(int m)
void off(int m)
void alloff()
void ao()
void motor(int m, int p)
```

Servo Motors

```
void servo_a7_init(n)
int servo_a7_pulse
void servo_a5_init(n)
int servo_a5_pulse
```

Sensor Input

```
int digital(int p)
int analog(int p)
```

User Buttons and Knob

```
int stop_button()
int start_button()
void stop_press()
void start_press()
int knob()
```

Infrared Subsystem

```
int sony_init(1)
int sony_init(0)
int ir_data(int dummy)
```

Time Commands

```
void reset_system_time()
long mseconds()
float seconds()
void sleep(float sec)
void msleep()
```

Tone Functions

```
void beep()
void tone(float frequency, float length)
void set_beeper_pitch(float frequency)
void beeper_on()
void beeper_off()
```

Multi-Tasking

```
int start_process(function-call(. . .), [TICKS],[STACK-SIZE])
int kill_process(int pid)
void kill_all
```

Process Management

```
void hog_processor()
void defer()
```

Arithmetic

```
float sin(float angle)
float cos(float angle)
float tan(float angle)
float atan(float angle)
float sqrt(float num)
float log10(float num)
float log(float num)
float exp10(float num)
float exp(float num)
(float) a^(float) b
```

Memory Access

```
int peek(int loc)
int peekword(int loc)
void poke(int loc, int byte)
void pokeword(int loc, int word)
void bit_set(int loc, int mask)
void bit_clear(int loc, int mask)
```

Laboratory Exercises

Prepare Your Handy Board

1. Refer to the sections *Preparing the Handy Board for Use* and *Loading the Operating System* above, locate all the necessary equipment for your lab station, and prepare the Handy Board for use.

2. Practice loading the firmware onto the Handy Board if you haven't done so already.

Programming Assignments

3. Create a program that generates a tone that is dependent on knob position by modifying the code listed in step (a) below. Save this program onto your lab disk; remember to give it the proper file extension.

   a. The following code measures the knob position and displays the position as an integer between 0 and 255 on the LCD screen. Download it to the Handy Board, reset the Handy Board, and check to make sure the program functions correctly.

```
int x;                                  /*Declare all variables*/
void main()                             /*Define the main program*/
{
while(1){
     printf("Press START\n");
     while(!start_button()){}           /*Wait for START to be pressed*/

     while(stop_button()==0){
     x=knob();                          /*define x as the knob position*/
     printf("knob is at %d\n", x);
     }                                  /*Close while loop. */
}                                       /*Closes while(1) loop. */
}                                       /*Closes main*/
```

4. Modify the code given above to include a conditional statement that makes the Handy Board create a tone with a higher frequency when the knob is above the halfway point, and a lower frequency tone when below the halfway point (use the tone command)—insert the following code just below the line where `x` is defined.

```
if(x<128){
     tone(200.,0.05);      /*generates a 200Hz tone for 50msec*/
}
else{
     tone(300.,0.05);      /*generates a 300Hz tone for 50msec*/
}
```

   Save the new program, load it onto the Handy Board, and demonstrate its behavior to your TA.

5. Modify the code you created in step four to create a tone whose frequency and duration depend linearly on the knob position according to the following equations:

$$f = x + 150$$
$$d = \frac{f}{1000} \qquad (1)$$

   where f is the frequency of the tone, x is the knob position, and d is the tone duration. Remember that f and d must be floating point variables. It may be necessary to convert one of the variables to a floating point variable by including the `(float)` command just before the variable you wish to convert.

6. Modify the code in step five so that it only runs for twenty seconds by modifying the `while (stop_button() == 0)` statement to include a Boolean AND statement:

```
reset_system_time(); while (stop_button()==0 && (int)mseconds()<20000)
```

The `mseconds()` function returns a long format number, which is a 32-bit integer. Since a Boolean logical operator can only compare numbers of the same format, the `mseconds()` result requires the `(int)` conversion. Save the program, run the code on your Handy Board, and demonstrate its behavior to your TA before you continue.

7. Create a new program that will help you measure how much time it takes to read the knob value, perform a calculation, output the value to the Handy Board speaker using the tone command, and then display the time delay on the LCD screen. Use the following code for this activity.

```
float x,y;
void main()
{
   reset_system_time();
   y=(float)knob();
   x=y*250./3.5;
   tone(x,0.01);
   printf("Delay time is %d msec\n",mseconds());
}
```

Don't forget to reset the system time at the beginning of your program. Save this program for your records.

   a. Make note of how much time it took to run these commands (time displayed on the LCD screen) in the space provided.

      Modify the code above to measure the time it takes to print text to the screen by repeating the `printf` statement at the end of your program. How much time is lost when you use the `printf` command?

      What percentage of the total time for the modified program did it take to run the `printf` command?

8. Write a program of your own design that generates a tone on the system speaker that meets the specifications listed below. Save the program for your own records and demonstrate that it functions properly to your TA.

   a. The program will start when the start button is depressed.

   b. The tone frequency will vary linearly with knob position from 600 Hz at the lowest position to 200 Hz at the highest position.

   c. The tone will last fifty milliseconds if the knob value is less than 156 and 100 milliseconds if greater than 156.

   d. The LCD screen will display the knob position and the number of seconds since the program began.

   e. The program will run for only twenty seconds and then can be restarted by pressing the start button again.

9. Clean up your station and answer the questions on the next page.

Questions

Refer to the lab handout, the *Interactive C Manual for the Handy Board*, and the *Handy Board Technical Reference* on the Handy Board website to answer the following questions.

1. In the table below, list the three different ways to charge the Handy Board's internal battery pack, the time it takes for each method to fully charge the battery, and the pros and cons for each.

Charging Method	Time to Full Charge	Pros	Cons
1.			
2.			
3.			

2. How fast is the Motorola 68HC11 microprocessor chip?

   How will the speed of the processor effect the control scheme for your robot project?

   Describe at least two methods of increasing program speed and efficiency.

3. Using the list of functions in this handout and the descriptions in the *Interactive C Manual for the Handy Board*, list five or more Handy Board functions that you think will be helpful in programming your robot and describe what they would do for your robot (you may want to discuss this as a project group).