EE 101	*Fall 2008*	Date: Lab Section #
Lab #10		Name:
Control and Motion Sequencing		Partner: NO PARTNERS THIS WEEK

Abstract

The ECEbot microcontroller has been pre-programmed for some simple motion control strategies. For example, you have seen how the bumper switches allow the robot to detect a collision, back up, turn, and then continue. In this lab a different control strategy is used: a list of motion commands is loaded into the microcontroller's memory, then executed one after the other. Much more complicated motion control programs are possible, but the simple methods can serve as building blocks.

Introduction and Theory

The microcomputer used in your robot is known as a *microcontroller* because it includes input/output pins for monitoring and controlling external devices, like switches and motors. We do not have enough time in EE101 to learn all the details of computer programming, but in this experiment you will determine a set of motion control instructions using a PC, download the instructions into your robot's memory, then observe and verify the resulting behavior.

Digital computers, such as the microcontroller that is part of your robot, execute a series of *machine instructions*. Each machine instruction causes data to be moved from one location to another inside the processor's memory, an arithmetic operation to occur, a logical test of some sort to determine the next instruction, or some combination of these actions. A computer programmer can write a sequence of instructions, called a computer *program*, directly using these native machine instructions, or more commonly using a *high-level language* that is easier to understand and maintain.

Digital computers perform arithmetic and logic operations on *binary* numbers. Unlike the base-10 (decimal) number system we use for most hand calculations, the binary number system is base-2.

Base-10 (or *decimal*): each column contains digits 0-9, and each column is weighted by 10^n.

1000's	100's	10's	1's
10^3	10^2	10^1	10^0

Base-2 (or *binary*): each column contains digits 0-1, and each column is weighted by 2^n.

8's	4's	2's	1's
2^3	2^2	2^1	2^0

Binary counting is appropriate because digital computers use electronic switches (transistors) that are either 'on' or 'off'. If we choose the 'on' and 'off' states to correspond to the '1' and '0' of the binary system, we can implement arbitrary arithmetic and logic functions using collections of transistor switches.

We can convert between decimal and binary by selecting the appropriate combination of digits and columns so that the numerical values match. For example, to convert 5 from Base-10 into Base-2, we first choose the largest column in Base-2 that does not exceed the numerical value: 5 is bigger than 4 but less than 8, so the most significant binary digit (or *bit* for short) is in the 4's column. The remainder between 5 and 4 is 1, so we also need a 1 in the 1's column. Thus, $5_{10} = 101_2$. Similarly, we can convert from binary to decimal by adding up the binary column weights. For example, $11010_2 = 1\times16 + 1\times8 + 0\times4 + 1\times2 + 0\times1 = 26_{10}$.

Binary bits are typically collected into groups of 8: a group of 8 bits is called a *byte*. Computers may also use collections of several bytes, which is known as a binary *word*. A word may contain two bytes (16 bits), three bytes (24 bits), four bytes (32 bits), or some other number chosen by the hardware designer.

One practical problem with writing numbers in binary is that it is difficult for humans to see whether or not two strings of bits match. For example, consider the following 16-bit numbers:

0110111010110101 and 0110111010100101

They do differ, but it is hard to notice with a quick glance. If we separate the 16-bit strings into smaller groups, say, groups of four, the difference is more noticeable.

0110 1110 101$\underline{1}$ 0101 and 0110 1110 101$\underline{0}$ 0101

By noticing that groups of four bits can represent 16 different combinations, it is reasonable to express the groups using base-16. In a rather unfortunate combination of Greek and Roman terms, base-16 is referred to as *hexadecimal*, or just *hex* for short. Since we need 16 symbols and there are only 10 decimal digits (0-9), hexadecimal number use the letters A-F for the additional digits.

Binary	*Hex*
0000	0
0001	1
0010	2
0011	3

Binary	*Hex*
0100	4
0101	5
0110	6
0111	7

Binary	*Hex*
1000	8
1001	9
1010	A
1011	B

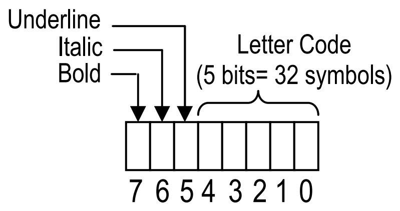
Binary	*Hex*
1100	C
1101	D
1110	E
1111	F

Thus, the two 16-bit strings from above when expressed in hex become:

6 E B 5 and 6 E A 5

and the difference is now more clearly visible.

In addition to storing numerical values, computers also use binary storage to hold *encoded data*. That is, there may be sub-groups of bits within a byte or word that represent different pieces of information. For example, we could invent a compact representation of the 26 capital letters A-Z and the period, comma, colon, semicolon, exclamation point, and question mark (total of 32 symbols) using 5 bits ($2^5 = 32$). If we then use three bits to indicate bold, italic, underline, we could pack the information into a single byte.

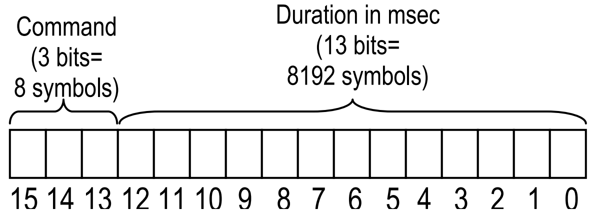


Computers don't typically use this particular text representation, but the encoding concept can be helpful in several different applications. As another example, the pre-installed software in your robot's microcontroller is designed to understand a set of specially coded commands.

The robot understands six commands, as shown in the table below. Since there are six commands, we need at least 3 bits to represent which command is to be used. Three bits can actually represent 8 commands, so two of the possible codes are not needed (extra Sleep modes!).

Command	*Code*
Forward	000
Left	001
Right	010
Back Up	011
Sleep	100
(Sleep)	101
(Sleep)	110
HALT	111

The microcontroller can operate on 16-bit words, so with three bits reserved for the command code there are 13 bits remaining in one word. The 13 bits are used to represent the number of milliseconds (0.001 seconds) in duration. 13 bits can represent 2^{13} numbers, 0 through 8191, so each command can have a duration of up to 8.191 seconds. The encoded data format is shown below.



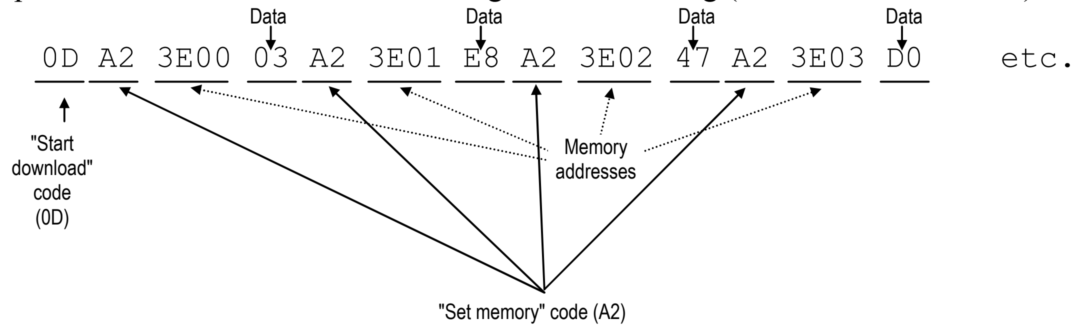
In order for the robot to use the encoded commands, the data must be loaded into a specific place within the microcontroller's memory. You will download the commands to the robot from a PC using a serial data cable.

You will use a pre-written spreadsheet program to create the command sequence. The spreadsheet will take the encoded commands and combine them with the *address* (memory location) and some special download commands. For example, if it is desired to have the robot perform the sequence:

1.	Forward for 1 second	`0000 0011 1110 1000`	hex: `03E8`
2.	Right for 2 seconds	`0100 0111 1101 0000`	hex: `47D0`
3.	Back Up for 2.375 seconds	`0110 1001 0100 0111`	hex: `6947`
4.	HALT	`1110 0000 0000 0000`*	hex: `E000`

*(*note that the duration of the HALT command is irrelevant)*

The spreadsheet would create the following command string (shown in hexadecimal):


Data
Data
Data
Data
0D A2 3E00 03 A2 3E01 E8 A2 3E02 47 A2 3E03 D0 etc.
"Start download" code (0D)
Memory addresses
"Set memory" code (A2)

Equipment

Your ECEbot assembled with the main electronic components, chassis parts, bumper switch modules, and interconnection cables, plus the programming software, special serial cable, and PC located in the lab.

Procedures

P1. First of all, take several measurements to determine how fast your robot travels (distance per time, e.g., feet per second or centimeters per second). Can you think of a convenient way to do this in the lab room? One idea would be to set your robot for mode _5_ (DIP switches 1 and 3 on) and use a wristwatch to measure how long it takes the robot to travel a known distance. The floor tiles in the hallway are 1 foot by 1 foot and make a pretty good ruler. Another approach would be to measure the circumference of one of the robot's wheels, then observe how many rotations occur in a certain time interval as the robot rolls along.

Choose one of these methods--or think of your own method--and record your robot's forward speed. Each robot will be slightly different. Explain your method and your results using complete sentences.

P2. → Each student needs to get the ECEbot Lab spreadsheet and copy it into an accessible project folder on the PC: name it with your initials.

→ Launch Excel and open the Lab spreadsheet. If you are sharing a computer, each student should open his or her own instance (window) of the spreadsheet. Save each instance with your initials to keep them straight.

→ Use the spreadsheet controls to create a sequence consisting of the following five commands:

a. Forward for 3 seconds
b. Right for 1 second
c. Left for 1 second
d. Backward for 3 seconds
e. HALT

The spreadsheet takes the commands and generates the sequence of control bytes.

a. In Excel, save your spreadsheet but don't close it: you will modify the commands again later.
b. Tools → Macros → Security → Choose low security.
c. Tools → Add-Ins → Choose Analysis ToolPak.
d. In the settings spreadsheet at the bottom of the excel page, make certain that COM PORT 1 is chosen.
e. Save, close, then reopen your excel program.
f. With the power off, set your robot for mode 6 (switches 2 and 3 on, all the others off). Next, hold the robot so that its wheels are off the table, then power up the robot. The display will show the software revision number (e.g., "3.1"), then the mode number ("6"), then a 4-digit pattern. As soon as the 4-digit pattern appears, press SW2 (left switch) on the CSMB12 microcontroller module. The robot will be on and the display will show a bright "0". The robot is now waiting for you to download the commands!
g. With the help of your TA or lab instructor, carefully connect the serial cable to the DB9 jack located on the side of the microcontroller module.
h. Now use the mouse to click the "Download" button in the spreadsheet. This command sends the data bytes into the processor's memory. The robot should flash "1111", indicating that it has received the command sequence.

Do not turn off the robot, as this would erase the robot's RAM memory!

Carefully remove the serial cable, hold the robot so that the wheels can turn freely, and press the Reset button: the robot should momentarily show its startup display, then begin the forward, right, left, backward, halt sequence that you downloaded. If this does not work, repeat the download and test sequence, and seek help from your TA or lab instructor.

Find an open space in the lab or in the hallway, set your robot on the floor, and press reset. Verify that the motion matches the command sequence. If you want to re-run the same sequence, just press reset and the microcontroller will perform the commands again.

P3. Now do some programming experiments to accomplish the following tasks. Debug the command sequences *iteratively*: if the results aren't correct, determine which commands and durations to change, copy, paste, and save the command string, and re-download to the robot for further testing.

When reprogramming your robot, simply alter the spreadsheet commands, save the spreadsheet file, and return to step (f) above. After the first time, there is no need to close Excel or to change the settings in the tool menu. If you want to reprogram the robot, press the "reset" button on the microcontroller and remember to press SW2 as soon as the robot displays the 4-digit pattern.

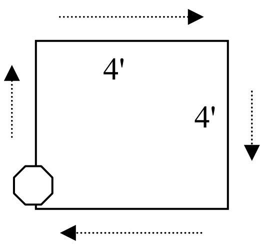
a. The maximum duration for any command is 8.191 seconds, but you can create longer intervals by using a repeated sequence of the same command. Make a sequence that will move forward for 10 seconds, then back up for 10 seconds.

b. You can press RESET to restart the sequence without having to download the commands again (as long as you don't switch the power off!) What happens if the bumper hits something during the sequence?

Write some test commands that will allow you to determine the duration for left and right turns so that the rotation is 90, 180, 270, and 360 degrees.

90°:	Duration: ________	270°:	Duration: ________
180°:	Duration: ________	360°:	Duration: ________

c. Using the 90º turn rate results and the forward speed results from **P1**, create, test, and debug a control sequence that causes the robot to follow a square trajectory, starting and stopping at the same point, with sides 4 feet in length and 90º corners. It is normal that your robot might drift left or right as it moves because the motors may not be perfectly matched. Try putting in some course corrections to compensate for any drift or veer in your robot's motion.



Robot maneuvers in a 4 foot square: Instructor/TA initials ____________

P4. You are now ready to program your robot to follow the route that is mapped out on the floor.

Robot successfully negotiates the route: Instructor/TA initials ________________

CONGRATULATIONS ON SUCCESSFULLY COMPLETING THE ROBOT EXERCISES!!