Lab 10: Introduction to Programming Autonomous Agents

OBJECTIVES

The objective of this lab is to introduce the student to controlling a mobile, autonomous robot based on the MC68HC11.

MATERIALS

- Talrik Jr. robot (TJ)
- MB2325 Communication Board
- RS-232 cable
- 40khz IR transmitter (i.e., TV remote control)
- TJ servo program (available on the web), **servotj.asm**

INTRODUCTION

The Talrik Jr. (TJ, see Figure 1) robot consists of a wooden body, a single-chip computer board containing an E2 processor, two servos for motors, two infrared detectors, and 2 infrared LED emitters. The LEDs are not used in this lab and will not be mentioned further.

Figure 1: Talrik Jr. (TJ).

Servos

The robot is driven by two independent servo motors. The servo motors are attached to the wheels and mounted to the sides of the robot. A servo is essentially a DC motor with feedback circuitry, which allows you to control the position the motor turns toward. To tell the servo where to go, a 50Hz signal is sent to the servo, and the duty-cycle of this signal tells the internal circuitry how the servo should be facing. The servo generally turns 180 degrees. However, these servos have been hacked so the feedback circuitry always believes the servo is in the middle position and it can turn 360 degrees. The angular velocity of the hacked servo is proportional to how far from the center position of its range it is instructed to move.

For the purposes of this lab, it will generally be sufficient to use full forward, full back, and off. No duty cycle at all (0) will cause the servo to not change its position.

SHARP IR receivers

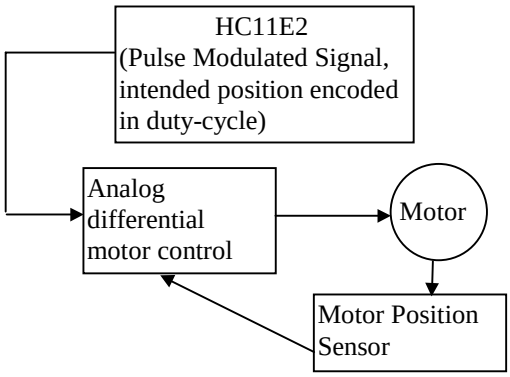

Figure 2: Unmodified servo controller.

The robot has the ability to detect infrared light modulated at 40Khz. Two forward-looking sensors, one mounted on the right and one on the left, allow the robot to tell from which direction the IR is coming. The sensor is a modified version of the standard receiver in a remote control device (like a TV or a VCR). It has been modified in such a way as to give an analog voltage corresponding to the amount of IR received. The nominal value of the sensors is 88 (decimal) on the A/D port and the saturation value is about 130. **The left mounted sensor is on PE6 and the right mounted sensor is on PE7.**

MC68HC11E2

This chip is similar to the ‘E9 we have been using all semester. The most noticeable difference is that Buffalo is not on this chip. Instead, to download a program the chip is put in special-bootstrap mode, and a program is downloaded to it. This program, called PCBug, is much like BUFFALO in that it allows you to download programs, assemble code, modify registers, etc. The use of this downloader will be demonstrated in the lab.

Another distinct difference between the ‘E2 and ‘E9 is that the E2 has EEPROM from $F800 to $FFFF. The programs you will right will be downloaded to these locations. You will notice that the interrupt vectors on this microprocessor are in EEPROM, unlike the ‘E9. Instead of programming pseudo-vectors, you will directly set the interrupt vectors. For example, it will be critical to have the reset vector set to the start of your program. That way when the chip is reset in single-chip mode it will begin executing your program code. Assuming the label that starts your program is named “start”, setting the vector would look like:

```
        ORG     $FFFE
        FDB     start
```

Your code is very likely to start at $F800. Another point to keep in mind about EEPROM is that you

Lab 10: Introduction to Programming Autonomous Agents

can't modify it on the fly (actually this can be done, see the Reference manual, but it is not as simple as just writing to a location). Do not try to write to an EEPROM location in your program. All variables and the stack must be in the 256 bytes of RAM in page0 memory ($0000-$00FF).

The 'E2 only has 256 bytes of RAM (as opposed to 512 in the 'E9, or 32k for expanded boards). Just be careful not to overrun this limit.

Code

The lab includes code to control the servos. This code is non-trivial, and it is best not to modify it in any way. There are three pieces to the code. The first piece, "_INIT_SERVOS" sets up the appropriate bits to allow the service routine to happen and clears the variables associated with the code. It is critical that this is called BEFORE any other servo routines are called.

The next routine for discussion is the interrupt service routine that drives the servo. This ISR controls both servos. It is critical that you set the OC2 interrupt vector to point to location of "_SERVO_HAND". It is similar to the output compare lab completed earlier this year. It outputs a 50Hz signal to each motor, connected to pins in Port B. Since the position the servo tries to take is determined by the uptime pulse-width modulation, the service routine checks a global variable called "width" that contains the uptime for each servo in E-clocks. It is not necessary to modify this variable directly.

Instead, a routine called "_SERVO**"** can be called. Simply pass the index of the motor whose duty-cycle you wish to change in register D (0 is left, 1 is right) and then push the new duty-cycle as a two byte number (low-byte first) on to the stack. Then execute "JSR _SERVO". A duty-cycle of **3000** E-clocks corresponds to the middle (not moving) position of the servo. The following table summarizes the full forward and full reverse of each motor. You will notice that the duty-cycle times are reversed between the right and left motors.

Motor	Full Forward	Full Backwards
0 (left)	2000	4000
1 (right)	4000	2000

Duty-cycles between 3000 and full forward will make the servo spin at a fraction of it its full speed. Remember also that a duty-cycle of zero will make the motor not spin at all, as will 3000 (the midpoint setting). For this lab it is best to use only full foward, backwards, and stop.

PRE-LAB REQUIREMENTS

1. Read the **ENTIRE** lab handout.
2. Write a complete, syntax error free program to meet the program requirements.
3. **DO NOT** start this program the day it is due. You will **not** be able to finish it.

PROGRAM REQUIREMENTS

In this lab, you will write two programs to accomplish the following tasks: first to move the robot in a simple pattern, and then to follow an IR emitting beacon.

Program Lab10A.ASM: Simple pattern generation-this will allow you to progress to the next program:

1. Initialize the servo drivers.
2. Go forward
3. Delay a while.
4. Turn.
5. Delay a while
6. Repeat steps 2~5, so that TJ moves in a simple pattern (e.g. triangle, rectangle, circle, etc.).

Program Lab10B.ASM: Follow an infrared(IR)-emitting device:

1. Initialize the A/D system.
2. Initialize the servo drivers.
3. Go forward.
4. Read the IR receivers on PE6 and PE7.
5. Move in the direction of the higher reading. If neither is higher, go straight.
6. Delay a while.
7. Repeat steps 4~6, so that TJ follows the IR beam.

INFORMATION

Helpful hints to write the program 1:

1. Write three routines for motion: one to go straight., one to make the robot spin left, and one to make the robot spin right.
2. As in the last lab you can make your delay any way you like. However the most straightforward and therefore most likely to work method is to use a simple looping delay subroutine. Make your delay a reasonable period, like a second or two.
3. Include SCI routines from previous labs in your code so you can debug your program. Remember, there is no BUFFALO so you will have to use the routines you wrote yourself. (They are also on the web).

Helpful hints for program 2:

1. Modify the routines you wrote to change the direction of the robot so that instead of simply spinning in place, they keep the robot moving forward and turn more gradually. Have one motor spin and the other motor simply turn off.

Lab 10: Introduction to Programming Autonomous Agents

2. You may chose to delay a little between reading the analogs. Otherwise, the robot may be oversensitive.
3. It is best that you look for a difference greater than 2 between the analog readings before turning. This keeps the effects of noise and mismatch between sensors to a minimum.

TESTING

Test all your code in SIM11 to the best of your ability. Make sure nothing crashes. (You may have a minor problem simulating in that the servo ISR reads port B before it sets it. The simulator apparently doesn't reset port B to zeros. It may be necessary to set $1004 to 0 in SIM11). Carefully think out all your code. You will have limited opportunity to download in the lab, and downloading to EEPROM is slow. Get every part right you can BEFORE coming to the lab.

QUESTIONS

1. Considering that there are IR-emitters on your robot, propose how you could use code similar to the IR following code to make the robot avoid obstacles in its path.
2. Looking carefully at the robot, explain why the duty-cycles for forward are different between right and left motors.
3. Propose a possible task(s)/application(s) using TJ.

WRITE – UP

(Due at the end of the lab period)

You will submit the following to your TA:

1. The printout of your LAB10A.LST & LAB10B.LST.
2. Answers to the questions.

Your name, lab section (day and time), TA name, and date should be written in the upper right-hand corner.

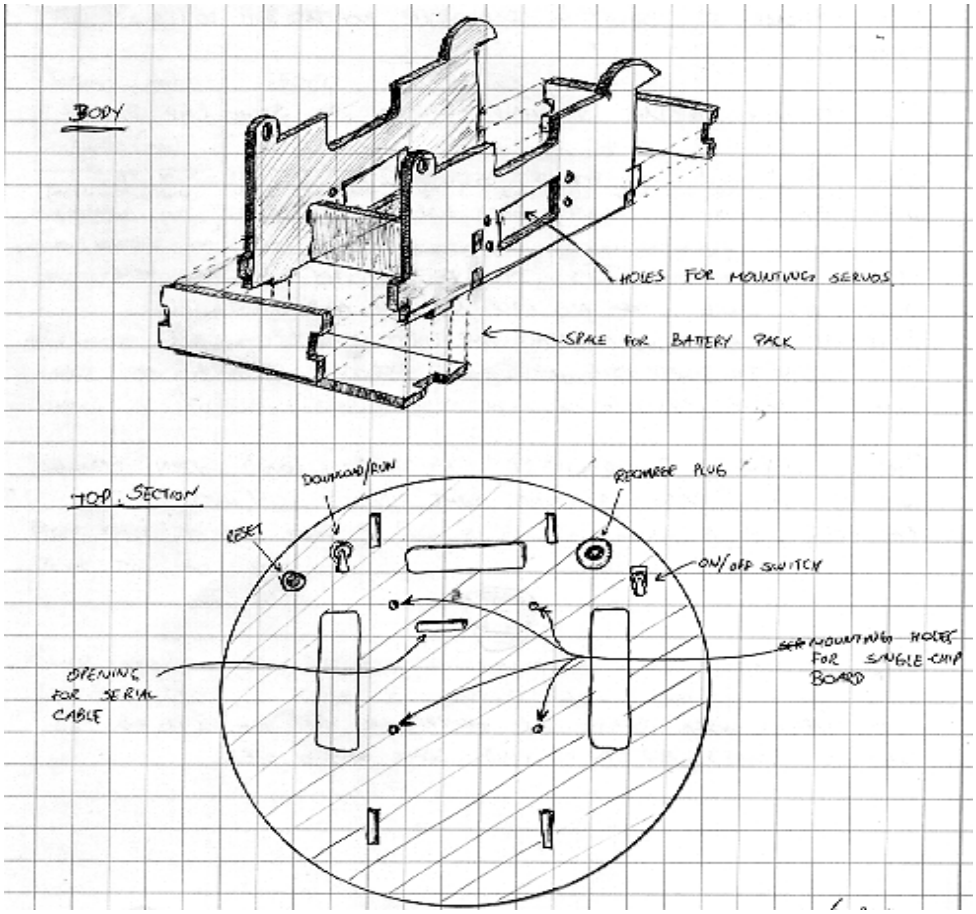

Figure 3: Drawings of a TJ body.