University of Florida

Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Deisgn Laboratory

‘RoBeats’

The Warzone Killa

Date:	**12/3/01**
Student Name:	**J. Bret Dennison**
TA:	**Scott Nortman**
	Aamir Qaiyumi
Instructor:	**A. A. Arroyo**

Table Of Contents

Abstract:

RoBeats is an autonomous robot based off of the character ‘Sweet Tooth’ from the Playstation game “Twisted Metal 2.” It roams around the “board” until it finds a line, representing the street, and follows the line until it comes across a barcode. It then makes firing noises. Robeats follows the line using 3 CdS cells. It has collision avoidance for objects in the way. The two back wheels are controlled by hacked servos while the front two wheels are free spinning. The 4 IR detectors in front are used for collision avoidance along with the bumpers. The servos, IR detectors, bump switches, CdS cells, and most of the behaviors are controlled by one processor while the barcode scanning and warzone sounds are controlled by another.

Introduction:

The main reasons for any robot are entertainment or making tasks easier for lazy humans. The purpose of my robot was strictly for entertainment. My favorite Playstation game at the time was Twisted Metal 2 and Sweet Tooth was the coolest looking character, so I based RoBeats off of him.

Integrated System:

The brains of RoBeats consist of the MRC11A64 board along with the MRSX01 Sensor Expansion board. The sensor expansion board has inputs coming in from the IR detectors, bump switches, and photoresistors (CdS cells). This tells the MRC11A64 board what is going on and then the MRC11 decides what behaviors should be done. For instance, when a line is found and being followed, the IR detectors do not need to be checked because one assumption is that RoBeats will not run into anything while following a line. The MRC11 controls when the servos are activated and when sensors need to be checked. The manuals for the MRC11A64 and MRSX01 can be found at http://www.mekatronix.com. The barcode scanning and sound producing are controlled by the CME11E9-EVBU board used in Microprocessors when the 68HC11 was used. When a barcode is read the CueCar barcode reader sends data on a data line which is connected to the pulse accumulator of the 68HC11. While in idle the CueCat sends a Constant valuethree or four times in a row. Once a barcode is read, this value changes a couple of times in a row. This is used to realize that a barcode was read. After reading the barcode, the 68HC11 sends a low signal to the warzone sound chip, ripped out of a bouncy ball from career fair and given leads.

Mobile Platform:

The platform structure of Robeats was designed in such a way to give maximum volume for components in a minimal amount of space. The dimensions are 10 x 6 x 6. The design takes the shape of an ice cream truck. Most of the components are placed in the back end of the robot for easier motion control, explained in Actuations below. The base and body were made out of 7-ply model airplane wood, which was cut on the ‘T-Tech’ machine in the lab. Hardware size was takin into consideration before cutting any wood to produce a clean fit. A problem with creating a body that surrounds all of the insides is that although it looks cool, you never absolutely know how much space you need until the very end. I took this into consideration but I still did not provide myself with enough room for everything I wanted to add. Another important consideration is holes for LED's, switches, sensors, and of course design. Many problems occurred because of the body but as time runs out, you just work around them.

Actuation:

Most of the components of Robeats are placed in the back end of the robot to give the back wheels more friction. The reason for this is for better steering. Robeats has 4 wheels. Motion control will be obtained using a standard servo on each of the back wheels. The front wheels are free spinning. Applying more friction (more weight) on the back wheels will make turning easier. Below is the servo information.

Servo Information:
RoBeats' design uses two standard servos, which drive four wheels. Each servo will drive both wheels on its corresponding side. This provides a way for steering.

$14.00 each from Mekatronix
Diamond Precision Servo (4.8-6V)
Torque: 43 oz / in
Speed: 0.20 sek

Sensors:

The four bump switches are basically buttons that complete a circuit when pushed, sending a signal to the processor. The four IR detectors used are the SHARP GP2D12, which contains both the emitter and the detector. The detecting distance is 10 to 80cm. They have three pins(power, ground, and signal). The IR's are used for collision avoidance. The three photoresistors used for line following read reflecting light at a short distance. Three white LED's were used to make reading easier. The ranges of the photoresistors were between 0 and about 40, 0 representing black and 40 representing white light. The following information tells where each sensor was bought and their price.

CDS Cells (3)
- free from IMDL Lab
- Line following for alignment with the barcodes.

Bumper Switches (4)
- free from IMDL Lab
- Collision Avoidance

IR Sensors (4)
- 0.00 each from Mekatronix

Special Sensor:

Cue Cat Barcode Reader:

The Cue Cat Barcode Reader is a device that connects to your PC between its keyboard and keyboard port. The device allows the user to swipe in the barcodes on books, product packaging, magazines, catalogs, coupons and ads to be taken to a Web site related to the product. I is a **free** item for a reason. Each of these devices contains with it a potential privacy hazard, a Globally Unique Identifier (GUID), a "secret code" that is assigned to each scanner. Every time an item is scanned, the serial number of your scanner along with whatever you scanned is sent to the parent company to track your personal scanning, buying, reading and even TV-viewing habits, and tie this information to a profile on you based on your scanner's GUID.

The information sent from the Cue Cat to the PC is as follows:

1.) Serial Number
2.) Barcode Type
3.) Tracking Code

An example of a Cue Cat scan is shown below. Each piece of data is seperated by a period.

```
.C3nZC3nZC3nWCxjWE3D1C3nX.cGf2.ENr7C3v7D3T3ENj3C3zYDNnZ.
```

Cue Cat Signals:

Figure 1 shows a Logic State Analyzer printout of the Cue Cat's signals during idle. The reason for not obtaining a printout during scanning a barcode is because the signals that change at the time a scan occurs (data lines and clock) change at random times also. Therefore you cannot trigger at the time of a barcode scan. However, information can be collected from the following. From the waveform you can see that there are two identical signals, Yellow and Black. These signals come from the clock output of the PC. An explanation of this is explained later. The Orange signal represents an 11 bit serial data line, which includes a start bit, stop bit, 8 data bits, and a parity bit. The waveform is showing two blocks of data being sent. There are 11 clock pulses for each block. The Red and Brown lines seem to be +5V.

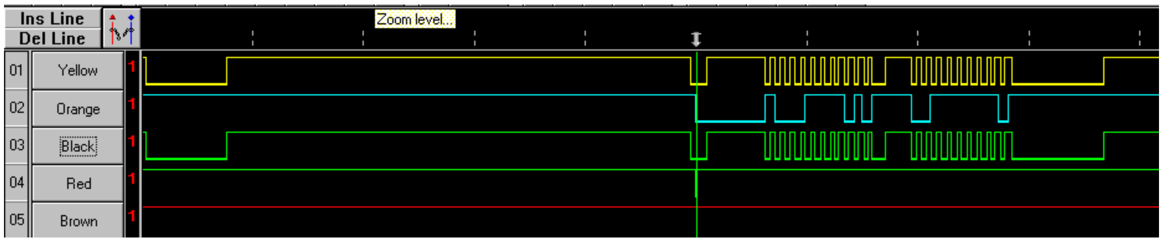


Figure 1. LSA printout of the cue cat at idle.

Figure 2 shows the inside of the Cue Cat. This one is currently being used as a door stop because it no longer works.

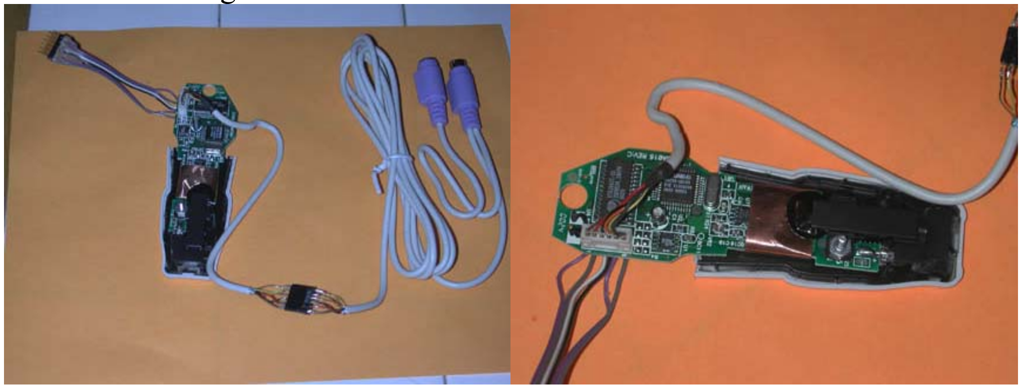


Figure 2 The inside of the Cue Cat.

Cue Cat Clock Signal:

Figure 3 shows the period of the clock signal of the Cue Cat. The Yellow line was cut before entering the Cue Cat (while the keyboard was unplugged) and it was connected to the oscilloscope. The other signals were connected to the oscilloscope and produced data like in **Figure 1**. The data line produced noise when scanning a barcode, showing the serial data transmitting.

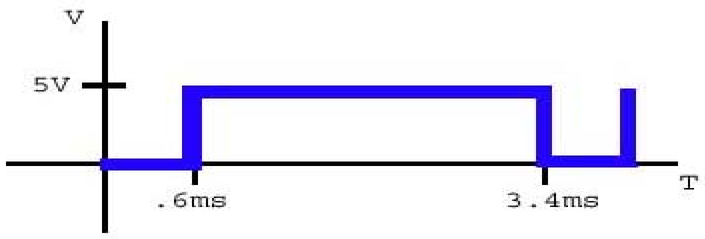
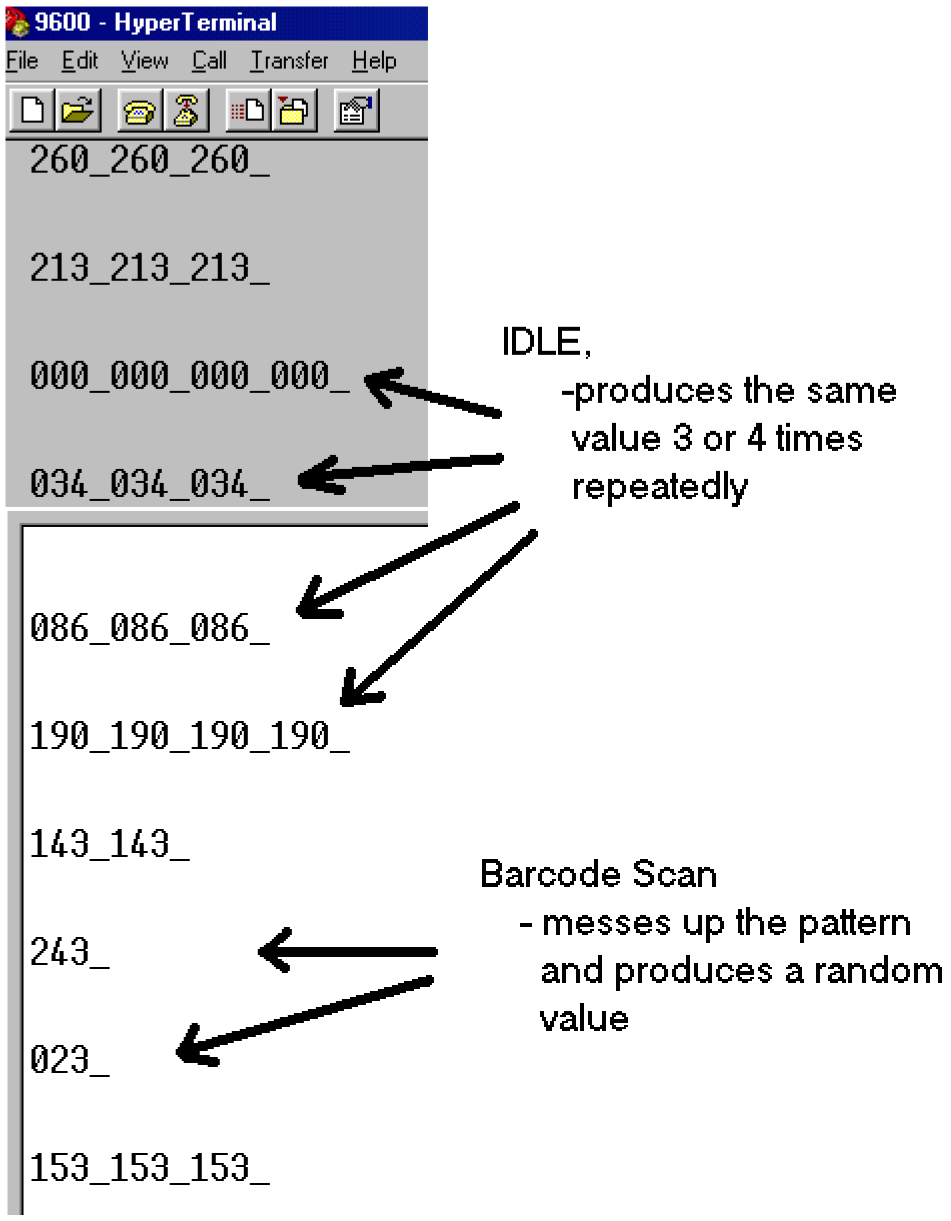


Figure 3: Period of the clock signal entering the cuecat from the PC.

Cue Cat Decoding Technique

After trying every possible solution I could think of to read the data from the Cue Cat, my roomate suggested, “Try using the pulse accumulator.” With some of his time and brains, I finally am able to see some output from the Cat. I connected the data line to Pulse Accumulator pin, and can see a difference when scanning a barcode. A problem with this is that the output is random when scanning the same barcode. **Figure 4** shows when the Cue Cat is in idle and when it scans a barcode. The Assembly code that produces this output follows in the Appendix.


9600 - HyperTerminal
File Edit View Call Transfer Help
IDLE,
-produces the same value 3 or 4 times repeatedly
Barcode Scan
- messes up the pattern and produces a random value

```
260_260_260_

213_213_213_

000_000_000_000_

034_034_034_

086_086_086_

190_190_190_190_

143_143_

243_

023_

153_153_153_
```

Figure 4 Output of the Cue Cat during idle and when scanning a barcode.

Behaviors:

RoBeats' main behavior is to avoid obstacles. While avoiding obstacles it continuously searches for a line to follow. Once a line is detected RoBeats will follow it until the end and then continue avoiding obstacles. While line following, RoBeats will scan barcodes which are aligned with the line, and produce warzone sounds.

Conclusion:

At the beginning of the semester, the idea for RoBeats was totally different then what it is now. The reason for this is due to problems occurring throughout the semester. The main goal was to be able to distinguish between different barcodes and produce their corresponding sounds, however getting any result was a big enough problem. If I started the project over I wouldn't have used the CueCat barcode reader because that took up most of my time. Everything about it is a problem. Now that I know how to do everything, I would have done it faster, giving me more time for the harder parts. I'd like ot thank Aamir and Scott for all of their help and ideas for getting feedback from the CueCat and any other help they gave me. I'd like to thank Jonathon Gamoneda for coming up with the idea and helping me with the Pulse Accumulator. I thank my sister for helping me with the design of the clown head.

Appendices:

* lilcat5.asm

```
*
*     Barcode Scanning sound producing code
*
* Outputs 11011111 to port d when a scan happens on the cuecat.
* Else outputs 11101111
* Bit5 connects to noise maker and must be high for no noise
* So when a scan occurs output a low to the noise maker and
* a high to the LED (Bit4)
* Uses Pulse Accumulator
*


BASE        EQU   $1000  ;BEGINNING OF REGISTERS
PACTL       EQU   $26    ;PULSE ACCUMULATOR CONTROL REG
PACNT       EQU   $27    ;PULSE ACCUMULATOR COUNTER REG
DDRD        EQU   $09    ; Data diraction for Port D
PORTD       EQU   $08    ; Port D

EOS         EQU   $04    ; User-defined End Of String (EOS) character
CR          EQU   $0D    ; Carriage Return Character
LF          EQU   $0A    ; Line Feed Character
BIT6        EQU   %01000000

            ORG   $00
DIFF        RMB   1
LAST        RMB   1

CNTR        RMB   1

*****************************************************
* Main program
*****************************************************
            ORG   $0100
MAIN        LDS   #$41
            LDX   #BASE

* INITIALIZATIONS
            LDAA #$0
            STAA LAST
            STAA DIFF
            LDAA #$2
            STAA CNTR
            LDAA #$FF
            STAA DDRD,X             ; set to output
            LDAA #%11101111   ; BIT5=1 no noise
            STAA PORTD,X            ; BIT4=0 LED off
```

```
* Initalize the Pulse Accumulator
                LDAA #$00
                STAA PACNT,X

                LDAA #BIT6          ; Turn on the pulse Accumlator
                STAA PACTL,X

HERE            LDX   #BASE
                LDAB PACNT,X                ; MAKE sure the SA value is in B
                JSR    Check
                JSR    DELAY

                BRA   HERE
*
*********************************************************************
* SUBROUTINE - Check
* Assumes current scan value is in reg B
*********************************************************************
*
Check           CMPB LAST  ; check new val with last val
                BNE   DIFFR
                BRA   SAME

DIFFR           DEC   CNTR
                LDAA CNTR
                BNE   Done

* OUTPUT A HIGH TO SOME OUTPUT
SCAN            LDAA #%11011111   ; BIT5=0 BIT4=1  SCAN OCCURRED
                STAA PORTD,X
                JSR    DELAY
                JSR    DELAY
                LDAA ##%11101111 ; set back to no scan on outputs
                STAA PORTD,X

SAME            LDAA #$2
                STAA CNTR

Done            STAB LAST  ; save current value
                RTS           ; return

***************************************************
* SUBROUTINE - DELAY
* Makes a .5 second delay
****************************************************
```

```
DELAY       PSHB
            PSHA
            LDAA #$4A
OUTER       LDAB #$FF
INNER       DECB
            DECB
            PSHY
            PULY
            PSHY
            PULY
            INCB
            BNE INNER
            DECA
            BNE OUTER
            PULA
            PULB
            RTS
```

```
/* RoAvoid5.c
   RoBeats code for obstacle avoidance and line following.
        4 IR Detectors
        4 Bump Switches - 2 front, 2 back
        3 Photoresistors
```

```
/*************************** Includes *********************************/

#include <tkbase.h>
#include <stdio.h>

/*********************** End of includes ******************************/
#define IR_THRESHOLD 70

void main(void)
/***************************** Main **********************************/
{
 int rspeed, lspeed,i;
 unsigned int IR_BOT_LEFT, IR_BOT_RIGHT;
 unsigned int FRONT_BUMP, BACK_BUMP;
 int j, cd3, cd4, cd5, cdflag, cdcntr;

 char clear[]= "\x1b\x5B\x32\x4A\x04";  /* clear screen */
 char place[]= "\x1b[1;1H";               /* Home cursor */

 init_analog();
 init_motortk();
 init_clocktk();
 init_serial();
 init_servos();

 printf("%s", clear);  /*clear screen*/
 printf("%s", place);  /*home cursor*/

 cdcntr = 0;
 cdflag = 0;

while(rear_bumper() != 23 );

while(1)
 {
  read_CDS();
  cd3 = CDS[3];
  cd4 = CDS[4];
  cd5 = CDS[5];

  read_IR();
  IR_BOT_LEFT = IRDT[3];
  IR_BOT_RIGHT = IRDT[5];

  FRONT_BUMP = front_bumper();           /*Returns front bumper value*/
  BACK_BUMP  = rear_bumper();      /*Returns rear bumper value*/
```

```
/* Check front and back bumpers for contact.  If front bumper is pushed,
        move back then turn. If back bumper is pushed, move foward and turn.  */
       if(FRONT_BUMP == 15)
       {
         rspeed = 4000;
  lspeed = 2000;

         for ( i=0; i<3; i++){
   servo(0,lspeed);
   servo(1,rspeed);
   wait(200);
  }

  rspeed = 4500;
  lspeed = 4500;

  for ( i=0; i<8; i++){
   servo(0,lspeed);
   servo(1,rspeed);
   wait(200);
         }
       }

       else if(BACK_BUMP == 23 )
       {
         rspeed = 2000;
  lspeed = 4000;

         for ( i=0; i<3; i++){
   servo(0,lspeed);
   servo(1,rspeed);
   wait(200);
  }

  rspeed = 1500;
  lspeed = 1500;

  for ( i=0; i<8; i++){
   servo(0,lspeed);
   servo(1,rspeed);
   wait(200);

         }
       }
```

```
        /* If no bumpers were hit, check CdS cells to find a line.  If left CdS detects,
 must catch up on right side. Likewise for right CdS.  If only the middle CdS
 detects, then continue straight.  */

else if(cd3 < 5 || cd4 < 5 || cd5 < 5){

    if(cdcntr==5){
      cdflag=1;
    }

    if(cd3 < 5){
      lspeed = 4300;    /* TURN LEFT A LITTLE */
          rspeed = 3000;

          servo(0,lspeed);
      servo(1,rspeed);

      if(cdflag==0){
      wait(10);
      cdcntr = cdcntr+1;
      }
      else {
       wait(40);
      }

       }
       else if(cd4 < 5){          /* go foward */
      lspeed = 4500;
      rspeed = 1500;
          servo(0,lspeed);
      servo(1,rspeed);

      if(cdflag==0){
        wait(10);
        cdcntr = cdcntr+1;
      }
      else {
       wait(80);
      }
       }
       else if(cd5 < 5){          /* TURN RIGHT A LITTLE */
      lspeed = 3000;
          rspeed = 2000;

          servo(0,lspeed);
      servo(1,rspeed);
```

```
          if(cdflag==0){
            wait(10);
            cdcntr = cdcntr+1;
          }
          else {
           wait(80);
          }
           }

    }
    else if((IR_BOT_LEFT > IR_THRESHOLD ) || (IR_BOT_RIGHT >
IR_THRESHOLD ) ) {

      if(cdcntr>=5){
        cdcntr=0;
      }


      if(IR_BOT_LEFT > IR_BOT_RIGHT) {
              rspeed = 1800;
              lspeed = 1800;
      }
      else {
              rspeed = 4300;
              lspeed = 4300;
              }

      for ( i=0; i<3; i++){
        servo(0,lspeed);
        servo(1,rspeed);
        wait(200);
      }

    }
    else {
      lspeed = 3500;
      rspeed = 2500;
    }

      servo(0,lspeed);
      servo(1,rspeed);

    wait(35);
  }
}
```

```
/**************************** End of Main ******************************/
```