**University of Florida**
**Department of Electrical and Computer Engineering**
**EEL 5666**
**Intelligent Machines Design Laboratory**

# Pet Buddy

Final Report

Date: 5/1/2001
Student Name: David Martin
TA: Scott Nortman
Rand Chandler
Instructor: A.A. Arroyo

# Table of Contents

## Abstract:

The goal of my project is to build a robot that detects the location of a pet and responds to its actions. The robot must also avoid obstacles while interacting with the pet. By using motors, sensors and a microcontroller, the robot will be able to move around and make decisions based on the surrounding environment.

## Executive Summary:

Pet Buddy never completely worked. For the in class demo, all of the sensors were working, but there were problems with the motor driver board; so the robot was unable to move. But the sonar was able to locate the "pet collar board." Then, for the media demo day, I installed servos, which worked great. But the sonar stopped working, I suspect a problem with the board itself (a broken trace maybe). So now, all the robot can do is obstacle avoidance.

## Introduction:

My first interest in micro-controllers and micro-controlled systems was when I took EEL 4744. I enjoyed the material and wanted to pursue it further. This course will prove to be challenging and educational. The experienced gained in this course will prove invaluable. My motivation for Pet Buddy, is that it would be very challenging to build a system that could locate a moving object such as a pet. A pet toy will be a practical application of this system.

**Integrated System:**

The following table is a summary of the sensors, actuators, and behaviors for Pet Buddy.

Computer: Motorola MC68HC11 (32k SRAM)
Language: C

| No. | Sensor Type | Range | Function | Location |
|---|---|---|---|---|
| 3 | IR Detectors | 1 ft. | Short range obstacle detection | front/rear |
| 1 | Polaroid 6500 sonor ranger | 1-30 ft. | Long range obstacle detection | front |
| 4 | 40 kHz transducer | 12 ft. | Pet localization | top |

Actuation: Servo motors on each wheel
Behaviors: Obstacle detection and avoidance. Determine pet's location, pet interaction Function: Pet Buddy's function is to be an autonomous pet toy that will interact with the pet and respond to its actions.

The basic software methodology used is as follows. The software was written using ICC for the TJPro. The RTI interrupt was used to read all of the sensors. Also the logic to control the Polaroid 6500 is contained in the RTI interrupt. The main body of the software first calls an initialization function, then goes into a while loop that contains the rest of the logic. First the maximum safe speed is found using the data from the Polaroid 6500[1]. Then, a routine is run to determine if the IR sensors detect a wall. If they don't, the pet-sonar information is used to determine the final speed for each wheel. The code is attached in the appendix.

**Mobile Platform:**

The major objective of the platform is that it will serve as a structure to support the electronics and sensors. The platform consists of a central cavity to store and protect the micro-controller board and other circuitry needed to interface the sensors. There is also a cover to support and maintain the alignment of the sensors. It is critical that the alignment of the 4 sonar receivers be maintained in order to get reliable data regarding the pet's location.

**Actuation:**

[1] Because I was unable to use the desired DC motors and had to instead use the servos, which are very slow. The Polaroid 6500 was not needed and thus taken out to conserve battery power. In the software, code pertaining to the Polaroid 6500 was commented out.

Two servo motors were used to move the robot. The data from each sensor is considered before the final speed for the motor is determined. The speed is controlled using the motorp() command in ICC. This command generates a PWM signal that is sent to the servo. The direction is encoded in the PWM signal. Electronics on the servo interpret the PWM signal and directly interface the motor.

I had originally intended to use DC motors capable of higher speeds. This would have allowed the robot to keep up with the pet and provide it with a competitive stimulus. Unfortunately, I had problems with the motor driver board and was unable to use the DC motors. Therefore, I had to use the standard servos available from Mekatronix. Since these are only capable of slow speeds, the maximum speed was kept at 100%.

**Sensors:**[2]

Polaroid 6500:

The Polaroid 6500 sonar module will be the primary means of obstacle avoidance. The objective is to see the obstacles far away and to avoid them. In addition, since the robot will be traveling relatively fast, it will be necessary to detect obstacles far away. One of these sensors was used and it detects the distance to the closest obstacle directly in front of the robot.

The Polaroid 6500 has two signals of interest: init and echo. Port D, bit 2 was connected to the init pin, as well as Output Compare 1. The echo signal was connected to Output Compare 2. To start the process, Port D-2 was set high. This triggered the Polaroid 6500 to send out a sonar pulse. The time this happened was captured in OC1. When the Polaroid 6500 hears the return echo, it sets the echo line high. The time of this event was capture in OC2. The difference between these two times is the time it took the sound to travel from the robot to the object and back. This time is measure in .5 microseconds. To roughly convert this time to feet, the difference was divided by 4. The data collected is attached in the appendix and figure 1 shows a plot of this data.

[2] This discussion assumes the use of DC motors when talking about the purpose and use of each sensor.

There were several modifications to the Polaroid 6500 that were required for its proper operation. First, a 4.7k pull-up resistor is required on the echo line. Second small bypass capacitors (about 10 uF) are needed directly on the board to reduce noise, which can cause errors on the echo line. I used two 10 uF and one 47 uF. Finally the Polaroid 6500 draws an extreme amount of current when firing, on the order of amps. A 1 mF bypass capacitor was required to avoid resetting the board!

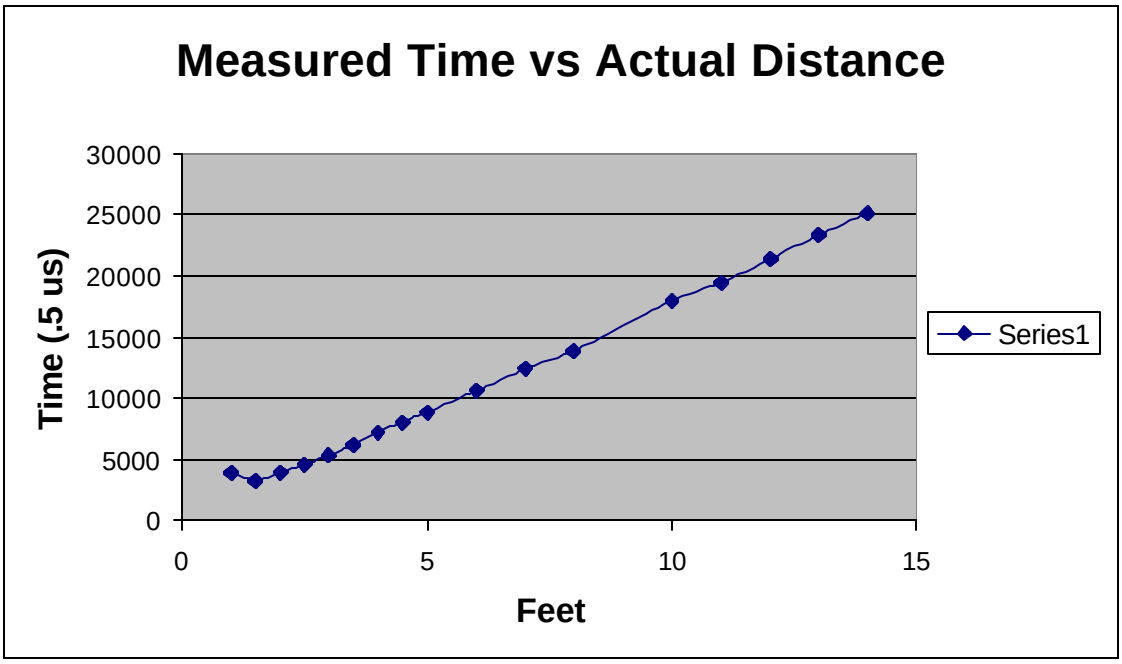


Figure 1: Experimental data for the Polaroid 6500 sonar module.

IR Detectors:

The IR detectors will be used for close range obstacle detection because the sonar can't detect objects closer than about a foot. The IR detectors were hacked to produce an analog signal proportional to the amount of 40 kHz IR received. IR emitters were placed near the detectors pointed away from the robot. The closer an obstacle is, the more IR that would be reflected back to the IR detector. The voltage was read using the analog ports.

When an object was about 10 inches away, the IR would read about 105. This was the first threshold used in the IR avoidance routine. When this threshold was reached, the robot would turn slightly from the obstacle. When an object was about 5 inches from the robot, the IR would read 115. When this threshold was reached the robot would abruptly turn away from the object. The code for this behavior is attached in the appendix.

Pet Location System:

The system to detect the pet's location is by far the most complex sensor in the robot. The system contains the following components:

1. 4 sonar receivers
2. logic to control event timing
3. Board to be placed on pets collar which emits sonar when triggered
4. analog circuits to convert sonar presence to digital values
5. logic to calculate the pets distance and direction

A flow chart of the basic operation of the system is shown below. The robot triggers the pets collar board to emit sonar. The robot waits to hear a response. Based on the response time at each of the 4 sensors, the pet's distance and angle are calculated. The distance and angle are stored in registers and are read by the CPU.

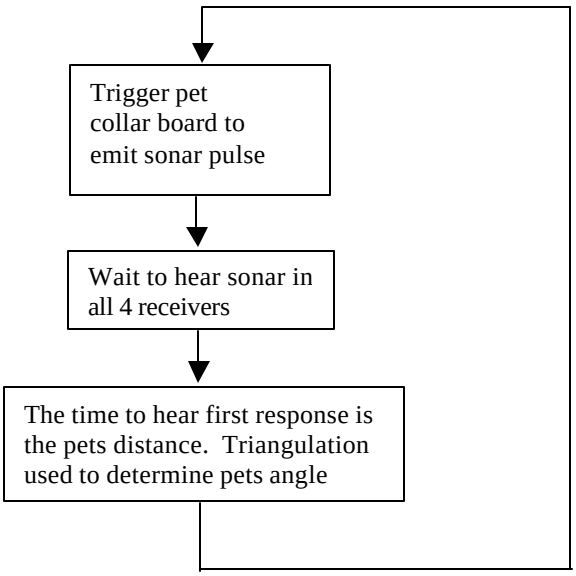


Figure 2: Basic operation of pet location system.

The pet collar board contains the following parts:

1. 40 kHz oscillator
2. RF receiver
3. Solid state relay
4. Amplifier and buffer
5. 40 kHz sonar transmitter
6. Power IC's

A block diagram of the pet color board is shown below in figure 3. Figure 4 is a circuit schematic of the oscillator and output stage.

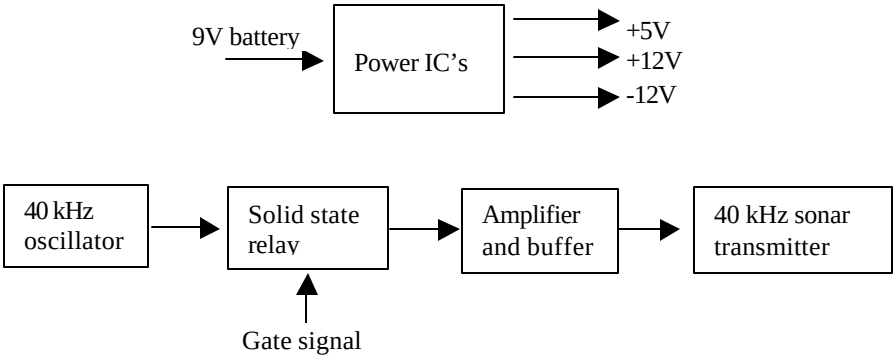


Figure 3: Block diagram of pet collar board.

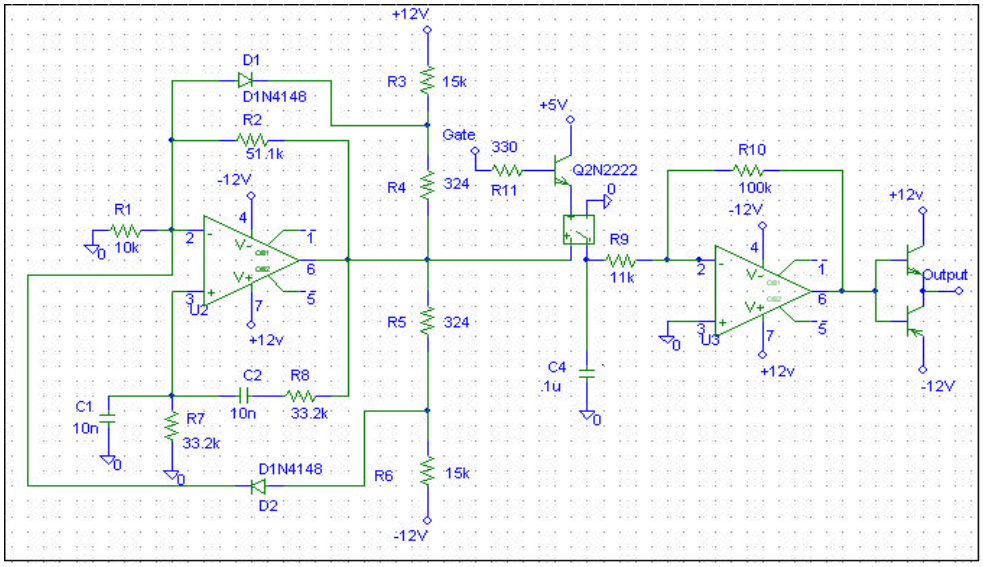


Figure 4: Schematic of oscillator, switch, and output buffer.

The oscillator was set up so that it generates a constant 40 kHz sine wave regardless of the status of the gate signal. This was done because the oscillator takes time to stabilize; thus a constant sine wave is desirable. The RF receiver has a digital output that is the same as the digital input to the RF transmitter on the robot. This output is the gate signal, and when equal to 5V, closes the switch. This sends the sine wave to the sonar transmitter. A buffer made from discrete transistors was required because an op-amp cannot supply enough current to the transducer. A capacitor was required on the output of the switch because when the switch was open, voltage spikes would still reach the output, causing unwanted sonar output. A capacitor to ground at this pin filtered out this spikes.

Below is a parts list for the pet transmitter board.

| Item # | Desciption | Source | Part # |
|---|---|---|---|
| 1 | 5V regulator | National Semi | LM1083 IT-5 |
| 2 | 5V to +_15V converter | Texas Instruments | DCP01B15DBP |
| 3 | +12V regulator | Digi-Key | NJM78L12A |
| 4 | -12V regulator | Digi-Key | NJM79L12A |
| 5 | High speed dual op-amp | National Semi | LM7372 |
| 6 | Various resistors and caps | Digi-Key | |
| 7 | RF reciever | Glolab | RM1V |
| 8 | NPN transistor | | 2N2222 |
| 9 | Power NPN transistor | Radio Shack | MJE30 |
| 10 | Power PNP transistor | Radio Shack | TIP42 |
| 11 | Sonar transmitter | Digi-Key | P9895 |
| 12 | Solid State Relay | Digi-key | PVA3324 |
| 13 | Diode | Digi-Key | 1N4148WSTR |

Figure 5: Parts list for pet collar board.

A circuit was also designed to convert sonar received to a digital output. This consists of a high gain, two-stage amplifier and a comparator. Shown on the next page is a schematic diagram of the circuit used. All of the op-amps used are high speed. A high slew rate is required for large signal operations at 40 kHz. There are four of these circuits, one for each sonar receiver.

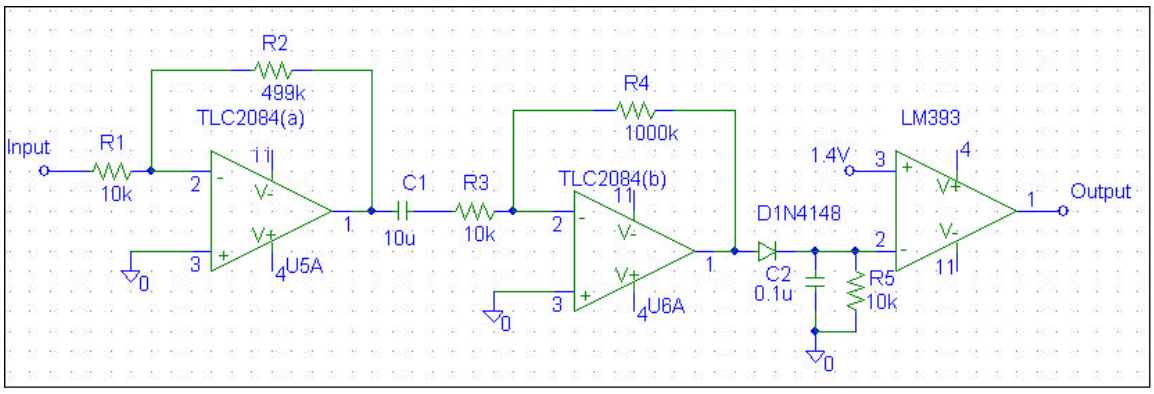

Figure 6: Schematic of sonar receiver circuit

A digital circuit was designed to take these four inputs and compute the location of the pet. This was done using Max-Plus II and a 7128S CPLD. The other inputs are the e-clock, Y1 and Y3 to read the registers. The outputs are an error signal and the gate signal. The hierarchy of the digital circuit is shown below.

- Sonar System
  - Timer
  - Controller
  - Channel Array
    - 2 Subtract modules
    - Channel_x
      - Channel Controller
      - Channel Registers

The timer is a 17-bit timer. It counts at rate of 1 MHz, thus it overflows after 131 ms. The timer is set up so that it doesn't count until triggered, once counting it will continue to count until it overflows and reaches zero again. Its inputs are a 1 MHz clk and an init_timer signal. Its outputs are the timer, a signal that controls how long the signal gate is high (when gate is high, the pet is emitting sonar), and an overflow signal that signals the timer has overflowed.

The controller is a finite state machine that controls the timing for the overall system. Below is its ASM diagram. The state machine enters a state and sets Gate = 1. It waits for a response on any of the channels or for the end_gate signal to go true and then it sets Gate = 0. If all channels are heard, it then loads the data into the registers. If an error condition exists, then the error flag is set. The state machine then waits for the timer to overflow and starts over. It waits for the overflow to space out the sonar pulses so the analog circuitry will not become overwhelmed.

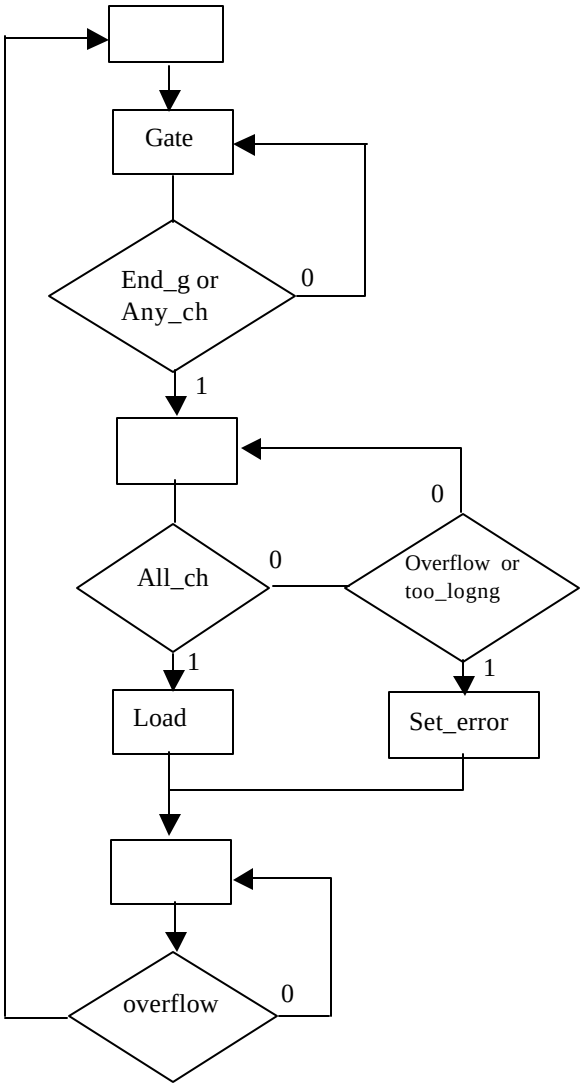


Figure 7: ASM diagram of main controller

Each channel has a state machine; this state machine monitors the input from the comparator. When this input goes high, the 10 least significant bits of the timer are copied into a register and stored to save that channels time. Also, a flag is set to indicate that the channel has heard the sonar. The ASM diagram is shown below.

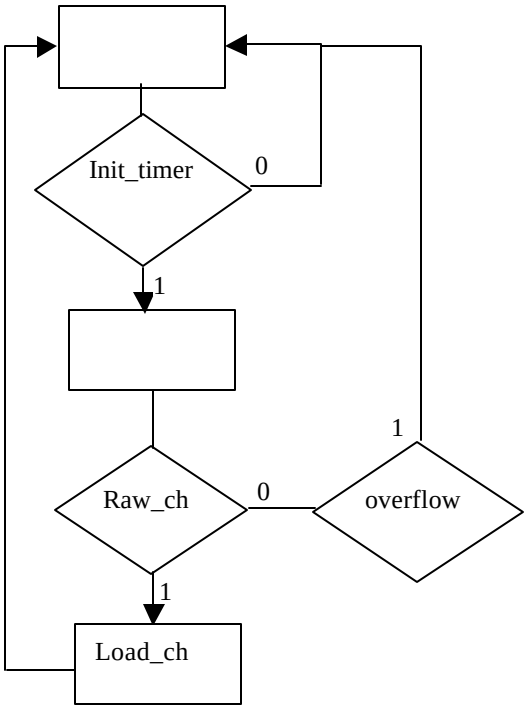


Figure 8: ASM diagram for channel controller

When the first channel hears the sonar, bits 15 through 9 of the timer are copied into the distance register. This gives the sonar a system a maximum range of about 64 feet. The values of channel 1 and channel 2 (left and right ) are fed into a subtraction module. The values of channel 3 and channel 4 (front and rear) are fed into a comparator, the result of which is 0 if channel 3 is less than channel 4 (which is the case if the pet is in front of the robot). The difference and the front/rear bit form the address to an EPROM. The data that is read out is the angle that the pet is relative to the robot. This angle is then stored in a register.

This system was working completely but failed before I could take experimental data.

**Behaviors:**

There are two behaviors programmed. One is obstacle avoidance and the other follows the pet at about 5 feet. I tested the obstacle avoidance behavior and it worked satisfactorily. I was never able to test the pet following behavior. The code for these behaviors is attached in the appendix.

**Experimental Layout and Results:**

I wanted to perform the following experiments:

- Characterization of Polaroid modules, including their effective range
- Characterization of the pet location system
- Determine the what speed will be produced by each duty cycle
- Characterization of the IR detectors

However, I only conducted tests on the Polaroid module and the IR detectors. The data for the Polaroid module is shown in the appendix and a graph is shown in figure 1. The results of the IR testing are described in the Sensor section.

Because the motor driver board failed and the servos are slow, it was not necessary to plot motor speed versus duty cycle. The sonar board failed before I could setup and conduct an experiment. But initial testing showed that it could calculate the angle and distance with consistency. And the maximum distance detected was about 12 feet.

## Conclusion:

This was a very challenging project, especially given the time constraints of one semester. I did manage to get everything on the robot working, just now at the same time, so I was never able to demonstrate the finished product. However, I was successful in designing a system to locate the pet. That was by far the most challenging part of this project. Even though I did not meet all of my initial specifications, I am satisfied with my accomplishments this semester. I gained valuable experience in printed circuit board design, analog and digital circuit design, and project management skills. I am confident that if I had the sonar board professionally made, it would be reliable and the entire robot would work. It would have been beneficial to have had two semesters to work on this project. That would allow for time to prototype, optimize, and then produce a final product.

## Documentation


Fred Martin, *The 6.270 Robot Builder's Guide* 2nd edition, MIT Media Lab, Cambridge, MA, 1992.

Joseph Jones, Bruce Seiger and Anita Flynn, *Mobile Robots: Inspiration to implementation* 2nd edition, A.K. Peters Publishers, Nattick, MA, 1998

Keith L. Doty. TJ-PRO Assembly Manual; Mekatronix 1999.

IMDL class: Instruction from Dr. Arroyo, Dr. Schwartz, Rand Chandler, and Scott Nortman

IMDL Web Page: http://mil.ufl.edu/imdl/handouts.


## Appendices

Please see attached code and sonar data

| Distance | Time of Flight |
|---|---|
| 1 | 3934.5 |
| 1 | 3914 |
| 1 | 3923.5 |
| 1 | 3902.5 |
| 1 | 3903.5 |
| 1 | 3887.5 |
| 1 | 3896.5 |
| Avg | 3908.86 |
| Std Dev | 16.21 |

| Distance | Time of Flight |
|---|---|
| 1.5 | 3229.5 |
| 1.5 | 3251.5 |
| 1.5 | 3234 |
| 1.5 | 3234 |
| 1.5 | 3252 |
| 1.5 | 3246.5 |
| 1.5 | 3247 |
| Avg | 3242.07 |
| Std Dev | 9.31 |

| Distance | Time of Flight |
|---|---|
| 2 | 3939.5 |
| 2 | 3955.5 |
| 2 | 3956.5 |
| 2 | 3966 |
| 2 | 3972 |
| 2 | 3939.5 |
| 2 | 3940 |
| Avg | 3952.71 |
| Std Dev | 13.43 |

| Distance | Time of Flight |
|---|---|
| 2.5 | 4572 |
| 2.5 | 4568 |
| 2.5 | 4557 |
| 2.5 | 4573 |
| 2.5 | 4566.5 |
| 2.5 | 4569.5 |
| 2.5 | 4574 |
| Avg | 4568.57 |
| Std Dev | 5.78 |

| Distance | Time of Flight |
|---|---|
| 3 | 5365.5 |
| 3 | 5365 |
| 3 | 5367 |
| 3 | 5370 |
| 3 | 5369.5 |
| 3 | 5364 |
| 3 | 5364 |
| Avg | 5366.43 |
| Std Dev | 2.49 |

| Distance | Time of Flight |
|---|---|
| 3.5 | 6204.5 |
| 3.5 | 6200.5 |
| 3.5 | 6210 |
| 3.5 | 6208.5 |
| 3.5 | 6199 |
| 3.5 | 6200 |
| 3.5 | 6214.5 |
| Avg | 6205.29 |
| Std Dev | 5.89 |

| Distance | Time of Flight |
|---|---|
| 4 | 7212 |
| 4 | 7204.5 |
| 4 | 7207 |
| 4 | 7210 |
| 4 | 7209 |
| 4 | 7209 |
| 4 | 7212.5 |
| Avg | 7209.14 |
| Std Dev | 2.78 |

| Distance | Time of Flight |
|---|---|
| 4.5 | 8039.5 |
| 4.5 | 8038 |
| 4.5 | 8034 |
| 4.5 | 8043.5 |
| 4.5 | 8044 |
| 4.5 | 8033.5 |
| 4.5 | 8036 |
| Avg | 8038.36 |
| Std Dev | 4.24 |

| Distance | Time of Flight |
|---|---|
| 5 | 8862.5 |
| 5 | 8851.5 |
| 5 | 8859 |
| 5 | 8863 |
| 5 | 8862.5 |
| 5 | 8860.5 |
| 5 | 8859 |
| Avg | 8859.71 |
| Std Dev | 3.99 |

| Distance | Time of Flight |
|---|---|
| 6 | 10664 |
| 6 | 10648 |
| 6 | 10646 |
| 6 | 10654 |
| 6 | 10649.5 |
| 6 | 10648 |
| 6 | 10645 |
| Avg | 10650.64 |
| Std Dev | 6.56 |

| Distance | Time of Flight |
|---|---|
| 7 | 12471.5 |
| 7 | 12461.5 |
| 7 | 12467.5 |
| 7 | 12474 |
| 7 | 12459.5 |
| 7 | 12463.5 |
| 7 | 12464.5 |
| Avg | 12466.00 |
| Std Dev | 5.28 |

| Distance | Time of Flight |
|---|---|
| 8 | 13855.5 |
| 8 | 13869 |
| 8 | 13841.5 |
| 8 | 13843.5 |
| 8 | 13848.5 |
| 8 | 13853 |
| 8 | 13864 |
| Avg | 13853.57 |
| Std Dev | 10.20 |

| Distance | Time of Flight |
|---|---|
| 10 | 17977 |
| 10 | 17956 |
| 10 | 17958.5 |
| 10 | 17960.5 |
| 10 | 17969 |
| 10 | 17971 |
| 10 | 17963 |
| Avg | 17965.00 |
| Std Dev | 7.57 |

| Distance | Time of Flight |
|---|---|
| 11 | 19459.5 |
| 11 | 19368.5 |
| 11 | 19417 |
| 11 | 19416 |
| 11 | 19440.5 |
| 11 | 19414 |
| 11 | 19413.5 |
| Avg | 19418.43 |
| Std Dev | 28.04 |

| Distance | Time of Flight |
|---|---|
| 12 | 21340.5 |
| 12 | 21320.5 |
| 12 | 21302 |
| 12 | 21339 |
| 12 | 21395.5 |
| 12 | 21383 |
| 12 | 21403.5 |
| Avg | 21354.86 |
| Std Dev | 39.25 |

| Distance | Time of Flight |
|---|---|
| 13 | 23299.5 |
| 13 | 23285.5 |
| 13 | 23334 |
| 13 | 23345 |
| 13 | 23345.5 |
| 13 | 23371.5 |
| 13 | 23337 |
| Avg | 23331.14 |
| Std Dev | 29.31 |

| Distance | Time of Flight |
|---|---|
| 14 | 25106.5 |
| 14 | 25103.5 |
| 14 | 25099.5 |
| 14 | 25104 |
| 14 | 25103.5 |
| 14 | 25103.5 |
| 14 | 25109 |
| Avg | 25104.21 |
| Std Dev | 2.94 |

```
/*********************************************************
* Title:            pet.c
* Programmer:       David Martin
* Date:             April 5, 2001
* Version:                1.0
*
* Description:
*********************************************************/
/********************** Includes ************************/
#include <analog.h>
#include <hc11.h>
#include <clocktjp.h>
#include <vectors.h>
#include <stdio.h>
#include <tjpbase.h>
#include <math.h>
/*********************************************************/

/********************** Variables ***********************/
unsigned char left_IR, right_IR, rear_IR;
unsigned char bumper, dist, pet_error;

char L_IR_obs_c, R_IR_obs_c, L_IR_obs_f, R_IR_obs_f, IRA_cnt;
char ir_motor_l, ir_motor_r, pet_motor_l, pet_motor_r;

unsigned int sonar;

char angle;
```

```
char sonar_cnt, sonar_error, sonar_cnt2, sonar_enable;

char s_max, IR;

#define raw_angle           *(unsigned char volatile *)(0x4000)
#define raw_dist  *(char volatile *)(0x5000)
#define IR_reg              *(char volatile *)(0x7000)
/*******************************************************/

/********************* Prototypes *********************/
#pragma interrupt_handler RTI_isr;
void initialize(void);
void find_s_max(void);
void turn(void);
void IR_avoid(void);
void pet_follow(void);

/*********************** Main *************************/

void main(void) {

        initialize();
        while(1)
        {
//              find_s_max();
                s_max = 100;
                IR_avoid();
                pet_follow();
                if (IR == 1)
                {
                        motorp(0, 0.88* ir_motor_l);
                        motorp(1, ir_motor_r);
                }
                else
                {
                        motorp(0, 0.88 * pet_motor_l);
                        motorp(1, pet_motor_r);
                }
                wait(50);

        }
}
/*******************************************************/

/********************** initialize() ******************/
void initialize()
{
// RTI interrupt used for gathering sensor data
        SET_BIT(TMSK2, 0x41);   // enable RTI interrupt
                                // timer overflows every 131 ms
        SET_BIT(PACTL, 0x01);   // interrupt every 8 ms
        CLEAR_BIT(PACTL, 0x02);
        SET_BIT(DDRD, 0x18);    // set data direction for Port D
        CLEAR_BIT(DDRD, 0x20);
```

```
// Input captures 1 and 2 are used for sonar ranging
//                      they do not trigger a hardware interrupt
        SET_BIT(TCTL2, 0x14);     // set TIC1 and TIC2 to capture on the
        CLEAR_BIT(TCTL2, 0x28);           // rising edge
        CLEAR_BIT(PORTD, 0x04);           // clear sonar_pulse bit
        SET_BIT(DDRD, 0x04);      // set port D, bit 2 as output

        sonar_cnt = 0;                    // counter used in the RTI interrupt
        sonar_error = 0;          // initialize sonar error flag
        sonar_cnt2 = 0;
        sonar_enable = 0;
        pet_error = 1;
        IR_reg = 0x07;
        init_analog();
        init_motortjp();
        asm("cli");
}
//*********************************************************
void RTI_isr(void)
{
        CLEAR_FLAG(TFLG2,0x40);

        left_IR = analog(3);
        right_IR = analog(1);
        rear_IR = analog(4);
        bumper = analog(0);
        dist = raw_dist;
        angle = raw_angle;
        pet_error = (PORTD & 0x20);

        if (sonar_cnt2 > 0)               //counter to pause between sonar firings
        {
                sonar_cnt2--;
        }
        else
        {
                if ((sonar_cnt == 0) && (sonar_enable == 1))
                {
                        CLEAR_FLAG(TFLG1, 0x02);        // clear the "sonar recieved flag"
                        SET_BIT(PORTD, 0x04);           // trigger sonar ranging
                        sonar_cnt++;                                    // increment counter
                }
                else if (sonar_cnt != 0);
                {
                        if( !(TFLG1 & 0x02) && (sonar_cnt <= 8) )               // if pulse not recieved
                                                                                //  and it isn't too long
                        {
                                sonar_cnt++;                    // increment counter
                        }
                        else if( TFLG1 & 0x02 )                 // pulse recieved
                        {
                                sonar = (( ((TIC2 ) - (TIC1)) / 4));      // compute range
                                sonar_cnt = 0;                  // reset counter
                                sonar_error = 0;    // clear error flag
                                CLEAR_BIT(PORTD, 0x04);         // clear trigger flag once recieved
```

```
                                        sonar_cnt2 = 15;
                                }
                                else
                                {
                                        sonar_error = 1;    // set sonar error flag
                                        sonar_cnt = 0;                  // reset counter
                                        CLEAR_BIT(PORTD, 0x04);
                                        sonar_cnt2 = 15;
                                }
                        }
                }

}/*end of end of RTI_ISR*/
/*****************************************************/

//***********************************************
void turn(void)

{
 int i;
 unsigned rand;

 rand = TCNT;

 if (rand & 0x0001)
 /*turn left*/
 {
   motorp(1, s_max);
   motorp(0, -s_max);
 }
 else
 /*turn right*/
 {
   motorp(0, -s_max);
   motorp(1, s_max);
 }
 i=(rand % 1024);
 if(i>250) wait(i); else wait1(250);
 ir_motor_l = s_max;
 ir_motor_r = s_max;
}

//***********************************************

void find_s_max(void);
{
        if (pet_error == 0)             //pet has been located
        {
                if (sonar < 2000) // obstacle less than 4.5 feet away
                {
                        s_max = 50;
                }
                else if (sonar < 3000) // obstacle less than 7.5 feet
                {
                        s_max = 60;
                }
```

```
                else if (sonar < 4000) // obstacle less than 9 feet
                {
                        s_max = 75;
                }
                else if (sonar < 5000) // obstacle less than 12
                {
                        s_max = 85;
                }
                else if (sonar < 6000) // obstacle greater than 15 feet
                {
                        s_max = 100;
                }
        }
        else
        {
                s_max = 50;
        }
}
//*********************************************************************
void IR_avoid(void)
{
        if ( left_IR > 115)
        {
                L_IR_obs_c = 1;   // set flag to indicate a close obstacle on the left
                IR = 1;
        }
        else if (left_IR > 105)
        {
                L_IR_obs_c = 0; // set flags to indicate far obstacle on the left
                L_IR_obs_f = 1;
                IR = 1;
        }
        else
        {
                L_IR_obs_c = 0; // set flags to indicate no obstacle on the left
                L_IR_obs_f = 0;
                IR = 0;
        }

        if ( right_IR > 115)
        {
                R_IR_obs_c = 1;  // set flag to indicate a close obstacle on the right
                IR = 1;
        }
        else if (right_IR > 105)
        {
                R_IR_obs_c = 0; // set flags to indicate far obstacle on the right
                R_IR_obs_f = 1;
                IR = 1;
        }
        else
        {
                R_IR_obs_c = 0; // set flags to indicate no obstacle on the right
                R_IR_obs_f = 0;
                IR = 0;
        }
```

```
	if ( (L_IR_obs_c == 1) || (R_IR_obs_c == 1))		// if close obstacle detected
	{
		if (L_IR_obs_c == 0)		// obstacle on the right
		{
			ir_motor_l = -s_max;		// avoid obstacle by turning left
			ir_motor_r = s_max;
		}
		else if (R_IR_obs_c == 0)	// obstacle on the left
		{
			ir_motor_l = s_max;		// avoid obstacle by turning right
			ir_motor_r = -s_max;
		}
		else					// obstacle straight ahead
		{
			IRA_cnt = 0;
			while ((rear_IR <120) && (IRA_cnt < 50))
			{
				motorp(0, -s_max);
				motorp(1, -s_max);
				IRA_cnt++;
				wait(10);
			}
			turn();				// turn in a random direction
		}
	}
	else if ( (L_IR_obs_f == 1) || (R_IR_obs_f == 1))		// if far obstacle detected
	{
		if (L_IR_obs_f == 0)		// obstacle on the right
		{
			ir_motor_l = 0;
			ir_motor_r = s_max;
		}
		else if (R_IR_obs_f == 0)	// obstacle on the left
		{
			ir_motor_l = s_max;
			ir_motor_r = 0;
		}
		else
		{
			turn();				// turn in a random direction
		}
	}
	else
	{
		ir_motor_l = s_max;
		ir_motor_r = s_max;
	}
}

//*********************************************************************

void pet_follow(void)
{
	if (pet_error == 0)
	{
```

```
                if (angle <= 0 && angle >= -64 ) // pet is in front and to the right
                {
                        if (dist < 6)
                        {
                                pet_motor_l = s_max;
                        }
                        else
                        {
                                pet_motor_l = (dist - 10) * s_max * 10 ;
                        }
                        pet_motor_r = pet_motor_l + (2 * angle);
                }
                else if (angle > 0 && angle <= 63) // pet is right  and front
                {
                        if (dist > 6)
                        {
                                pet_motor_r = s_max;
                        }
                        else
                        {
                                pet_motor_r = (dist - 3) * s_max * 33 ;
                        }
                        pet_motor_l = pet_motor_r - (2 * angle);
                }
                else if (angle <= -65 && angle >= -128) // pet is behind and left
                {
                        pet_motor_r = s_max;
                        pet_motor_l = -s_max;
                }
                else
                {
                        pet_motor_r = -s_max;
                        pet_motor_l = s_max;
                }
        }
        else
        {
                pet_motor_r = s_max;
                pet_motor_l = s_max;
        }
}
```