Student Name: Jeff Panos
TA : Louis Brandy
William Dubel
Max Koessick
Instructor. A. A. Arroyo

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

Final Report

GIMP

A dog robot that will fetch a stick and bring it back to a designated place

Table of Contents

Executive Summary…………………………………………………………...3

Introduction……………………………………………………………….....4

Body Design……………………………………………………………….. 4

Integrated System………………………………………………………….. 5

Actuation…………………………………………………………………6

LCD……………………………………………………………………... 7

Receivers…………………………………………………………………8

Sensors…………………………………………………………………….. 9

Gimp’s Stick and Home Base……………………………………………… 10

Conclusions………………………………………………………………… 11

References………………………………………………………………….. 13

Source Code……………………………………………………………….... 14

Executive Summary:

Gimp uses the Atmel ATmega128 processor as his brain. He has three servos for actuation; two for romaing, one for his jaw. Gimp has ultrasound sensors to guide him from obstacles as well as to help him locate things that are necessary to locate. And an LCD screen and an LED are both used for debugging purposes.

He uses two beacons to fetch a stick and bring it back to home base. One beacon uses a 56.8 kHz signal, while the other uses a 40 kHz signal. Through the use of analog output, Gimp is able to use the strength of a signal to determine, generally, where he is in accordance to the beacon. If he is not sure, or loses signal strength he scans his location to find which direction is the best direction for him to take.

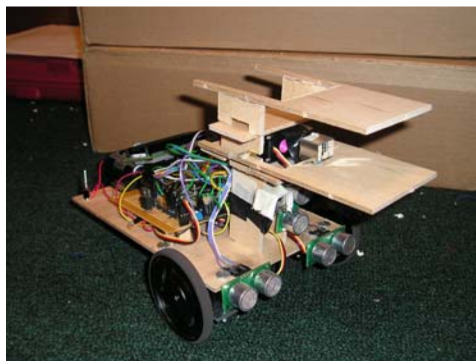
Once he finds his first destination (the stick), Gimp will then scan his location for home base. As he does when he finds the stick, he will determine the best way to get there. Once he sees it, he pings using ultrasound to determine the distance from home base. Given a specified distance, he stops and waits patiently with his stick to do it all over again.

Introduction:

In this paper, it will be described how Gimp fetches a stick and brings it back to a designated place, called "home base." He will demonstrate how he uses ultrasound sensors and beacons to guide his way through the use of his brain, the processor.

The idea of Gimp is to home in on a signal coming from it's "stick", stop in front of it, pick it up, and then proceed to home in on another signal, i.e. home base. He will avoid obstacles before he begins looking for his stick. Then go on to find his stick once he knows it's close by. Once he reaches home base, he waits around for another go just like any other dog.

Body Design:

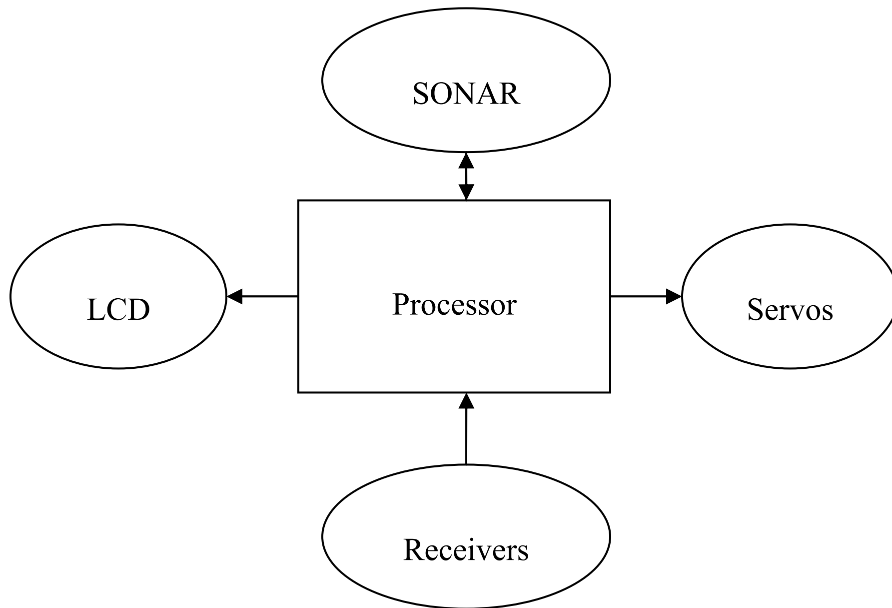


Gimp's platform is a simple one level design just for placement of items. The processor was offset from the main platform to use pins as needed from the bottom. His wheels are attached by a small metal plate 2 screws for each servo except the jaw. His notorious

backwheel is a ball-bearing caster wheel. This gives Gimp his great name. This also gives Gimp the best turns with the least bit of friction. It is not recommended to use a regular chair caster wheel.

Gimps head is attached to the platform with a level suited for the servo to control the jaw. It is not recommended to use my design for his jaw, however, it did work fairly well picking up a 9V battery with small additional weight. His upper jaw is securely connected to the main head so that the bottom part of the jaw can open and close.

Integrated System:



Arrows refer to whether the processor is receiving or sending information.

Gimp's Package:

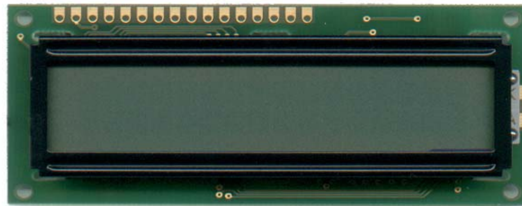
- ATmega128 Processor, chosen for its fast clock speed of 16 MHz, 8 Analog-to-Digital Conversion (ADC) channels, multiple interrupts, and Pulse Width Modulations (PWM)
- 16 X 2 Liquid Crystal Display (LCD) with LED Backlight
- Jameco 56.8 kHz Receiver
- Sharp 40 kHz Receiver
- 3 Devantech SRF04 Ultrasound Sensors
- 3 Hitec Standard Servos
- Powered by 8 NiMH Batteries

Actuation:

The HS-311 Hitec Standard Servos were used for actuation. These servos can operate between 4.8V – 6V and are all clockwise turning servos. They are extremely light, approximately 43g (1.5oz), but were not pre-hacked, which means they are not continuous servos. I had to hack 2 out of 3 of them so that they would spin without any problems. This involves using a Dremel to chip away at the gear pulleys where there is a notch that stops it from going around 360 degrees. Also, there is another notch to get rid of that centers the servos. Other than that, these servos worked great, but do not work

using the current from the processor. It is a must to use a voltage regulator because servos will draw more current than the processor can output.

LCD:



Gimp uses a 16 by 2 LCD with LED backlight by Hitachi. Can work in either 4-bit or 8-bit mode and uses a Hitachi HD44780U controller. The LCD has low power operation support from 2.7V to 5.5V. It has RAM on it that can be written to and read from. It allows the user to write one thing to it, then another, and still be able to recall what was previously written beforehand. I did not want to take the time to learn the use of that because it did not pertain to me, but it was nice to have the option if it were in fact necessary. The LCD was used for debugging purposes. It helped to determine where code was getting stuck in loops and such. Sometimes the problem was that it created a timing delay for other devices to work properly. This was found to be true in my case through much trial and error.

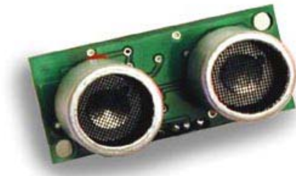
Receivers:

56.8 kHz 40 kHz

The receivers used for my beacons were the Jameco 56.8 kHz and the Sharp 40 kHz. Both were not previously hacked, however, it became a terrible problem to learn that Sharp does not sell the 40 kHz cans anymore. They are just integrated circuits with digital outputs from the receiver. I obtained this from another IMDL student who had these from a TJ robot created through MIL (Machine Intelligence Laboratory). This other student had already hacked it, which saved the problem with the digital output receiver. Gimp could not seem to scan and locate the receiver unless within 3 inches of it, no matter how slowly he scanned. The 40 kHz can I obtained had a great distance reading. See Special Sensor Report for more details.

The 56.8 kHz I hacked myself using another IMDL students document, see Documentations. This allowed Gimp to determine signal strength, in turn, giving him a vision on how close or how far away he might be from this transmitter. It worked very well and did allow him to play the game “Hot and Cold.” See Special Sensor Report for more details.

Sensors:



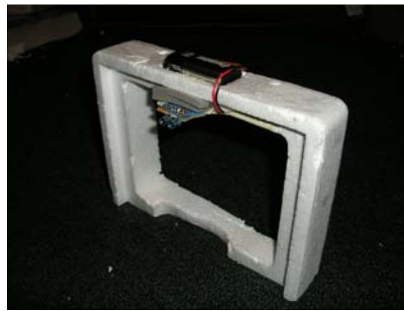
Devantech SRF04

Other than his special sensor combination of a 2 beacons (see Special Sensor Report), the main sensors that were used for Gimp were the 3 ultrasound sensors. These are the Devantech SRF04 Ultrasound Sensors. These determined distance for obstacles to avoid and also for something to move towards. These sensors are very specific to timing. There are 4 pins that are used on ultrasound sensors, power, ground, pulse trigger, and echo line. You toggle the pulse line High, hold for at least 10μs, then pull it Low to initiate a ping. Now the receiver circuitry is being enabled, and the timing for this to happen is about 200 or more microseconds (test to check). Next, the echo line is low and waiting for a ping to come back from hitting an object. The echo line goes High, when it receives a ping. You must time this ping to determine a distance. Basically, it can be adjusted that about every 50μs is about 9mm. This can vary, and it may not really be exact, but it gives you a general idea of a distance.

Two of the ultrasound sensors (SONAR) were used for obstacle avoidance and they work very well with a little bit of tweaking. The other SONAR distance to home base once

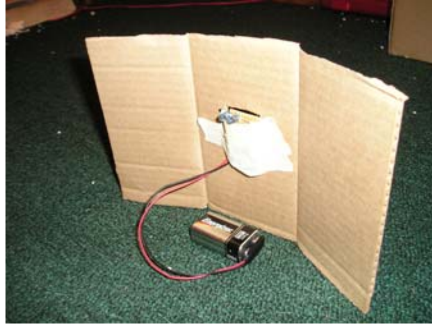
Gimp had his stick. This became extremely useful due to home base not having a great strength of signal from up close, but only from far away, unlike the stick's beacon. Pinging allowed to find the distance and determine what to do from there.was used to ping for the

Gimp's Stick and Home Base:



Gimp's Stick

This is Gimp's stick. It was designed to be at mouth level for him to easily grip it. Gimp proceeds towards this using the signal strength. Once he locates it, he will snap tightly on it and maintain this tight grip while scanning for Home Base.



Home Base

Gimp's Home Base was designed as one can see so that he could approach from different angles and still stop a specified distance away. When he gets too close he pings with the SONAR for a distance reading from Home Base. When that specified distance is met, he will stop while holding the stick and wait to do it all over again.

Conclusions:

Gimp's design was not well thought through. This is mostly due to the inexperience of completing a robot. Although it did not work perfectly, it did have many things went better than expected. The beacons got much better ranges than I ever they would. It was interesting to see how accurate they could really be. The other things that were successful without many problems was the design of Gimp's jaw. It was a horrible design mechanically, but that was no my main aspect of the project. The whole idea was to manipulate the code and the electronics to do as I proposed. In that sense, it was completed successfully.

Some problems that were encountered were loose solder wires and power problems. The processor seemed not to like the alkaline batteries that were used in case it was necessary to have back-up power. Gimp seemed to be very sensitive to power operation below 10.5V sometimes. Code would not run all the way through and he would get stuck in loops very often when this happened. Although this was very weird, it was good to find this out so that time was not spent on other areas debugging. Another problem that Gimp had was that he was very sensitive to Home Base. When he pings for it using ultrasound, he sees it so far away that he just stops. See Special Sensor Report for details, but basically the same digital number is seen from 0 to 6 feet using the Home Base beacon. Gimp knows this and stops when it gets within that range. However, he has been modified to come to Home Base and not stop, but be picked up by the owner because he is just that special of a dog bot.

There were several things that I would have done differently for Gimp. First, I would have designed a better stick for him to grip onto. The stick needed a different place for the 9V battery and even the transmitter. However, it did work out surprisingly well.

Second, I would have reconstructed Gimp's jaw differently. I would have consulted a mechanical engineer on a much more sturdy design. The idea was to come up with a pulley system or a way for the servo to constantly spin in a way that the jaw would open and close within a 360 degree turn.

References:

16x2 LCD Display with LED Backlight, Hitachi HD44780U LCD Controller Datasheet.
http://www.junun.org/MarkIII/datasheets/HD44780u.pdf

Atmel ATmega128 Microprocessor Datasheet. © Atmel Corporation 2003, Atmel® and combinations thereof, AVR®, and AVR Studio®,
http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

Devantech SRF04 UltraSonic Range Finder. © 1994-2003, Acroname Inc.,
http://www.acroname.com/robotics/parts/R93-SRF04.html

HiTec HS-311 Standard Hobby Servo Motor.
http://www.rentron.com/Products/ServoS.htm

Jameco 56.8 kHz Receiver Hack. Credited to Michael Hatterman, Spring 2002.
http://www.mil.ufl.edu/imdl/papers/IMDL_Report_Spring_02/michael_hatterman/hacked_ir.pdf

Sharp 40 kHz Receiver Hack.
http://www.mil.ufl.edu/5666/handouts/radio_shack_ir_hack.htm

Source Code for Gimp

```
// Jeff Panos
// EEL5666C - Intelligence Machine Design
// Gimp's Main "Package"

/***************************************/
/***************************************/
/************* MAIN PROGRAM ************/
/***************************************/
/***************************************/

#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>

void delay(int t);
void clr_display(void);                 //Clear LCD Display
void command(void);                     //Latching Command Data
void data(void);                                //Latching Character Data
void disp_control(void);                //Display Control
void entry_mode(void);                          //Entry Mode
void fct_set(void);                     //Setting Functions
void send_to_LCD(char lcd[32]);         //Sending characters to the LCD
void lcd_init(void);                    //LCD Initialization
void turn_left(void);
void slowly_turn_left(void);
void turn_right(void);
void slowly_turn_right(void);
void slowly_move_fwd(void);
void move_fwd(void);
void stop(void);
void back_up(void);
void delay_10us(int t);
void timer1_ab_init(void);
void avoidance(void);
void left_sonar_init(void);
void right_sonar_init(void);
void home_base_sonar_init(void);
void finding_home_base(void);
void adc_init(void);

int left_dist=0;
int right_dist=0;
```

```
int distance=0;
int count=0;

/* Use for debugging - displaying variable on LCD */
int def_putc(uint8_t x)                              //change to 'char' if want to
send character
{
        PORTC=x;                                       //sending each
character to LCD
        data();                                        //latching LCD data
        delay(1);                                      //delay between each
character sent
        return x;
}
uint8_t x = 0;

/*************************** Main Program***************************/

int main (void)
{
        DDRA = 0xFF;
        DDRC = 0xFF;
        DDRF = 0x00;
        fdevopen(def_putc, NULL, 0);                   // initialize stdio
        delay(1);
        lcd_init();                                   // LCD Initialization
        delay(1);
        send_to_LCD("Powered Up!");
        timer1_ab_init();
        ADMUX = 0x60;
        // MUX4:0=00000 activating ADC0 (channel 1)
        ADCSR = 0x80;
        // enable ADC
        sbi(ADCSR,ADSC);                              // start conversion
        sbi(ADCSR,ADFR);          // enable free-running mode, constantly update
        loop_until_bit_is_set(ADCSR,ADIF);        // checking for end of conversion

        while(1)        // ***Tease with stick before he looks for it***
        {
                sbi(ADCSR,ADSC);
                sbi(ADCSR,ADFR);
                loop_until_bit_is_set(ADCSR,ADIF);
                slowly_move_fwd();

                while(ADCH < 120 && ADCH > 88)
                {
```

```
			slowly_turn_right();
			loop_until_bit_is_set(ADCSR,ADIF);	// refreshing value
			while(ADCH < 117)
			{	slowly_turn_right();
				loop_until_bit_is_set(ADCSR,ADIF); }
		}

		loop_until_bit_is_set(ADCSR,ADIF);
		while(ADCH > 126 && ADCH <= 129)
		{
			slowly_turn_right();
			slowly_turn_right();
			loop_until_bit_is_set(ADCSR,ADIF);	}

		while(ADCH > 130 && count==0)		// reached beacon
		{
			slowly_turn_right();
			slowly_move_fwd();
			OCR1A=0;				// stop servos
			OCR1B=0;
			loop_until_bit_is_set(ADCSR,ADIF);
			delay(20);
			OCR1C=1450;				// open jaw

			delay(40);
			OCR1C=0;
			delay(80);

			OCR1C=1650;				// close jaw
			delay(50);

			count=1;				// to get out of loop
		}

		while(count==1)
		{
			OCR1C=1650;
			ADMUX = 0x61;			// switching to ADC1
			adc_init();		// Enabling ADC1, starting conversion
			finding_home_base();		// going for Home Base
		}

	}
}
```

```
/*************************************************************************/

/************************** Initializing ADC **************************/

void adc_init(void)
{
	//ADMUX = 0x60;
	// activating ADC Channel 0
	ADCSR = 0x80;
	// enable ADC
	sbi(ADCSR,ADSC);					// start conversion
	sbi(ADCSR,ADFR);		// enable free-running mode, constantly update
	loop_until_bit_is_set(ADCSR,ADIF);  // checking for end of conversion
}
/************************************************************/

void finding_home_base(void)

	{
		sbi(ADCSR,ADSC);
		sbi(ADCSR,ADFR);
		loop_until_bit_is_set(ADCSR,ADIF);
		slowly_move_fwd();
		clr_display();
		send_to_LCD("Looking for it...");
		while(ADCH > 87 && ADCH < 110)
		{
			slowly_turn_right();
			while(ADCH > 100 && ADCH < 110)
			{	slowly_move_fwd(); }
		}

		while(ADCH > 110 && ADCH < 126)
		{
			slowly_turn_right();
			while(ADCH > 120 && ADCH < 126)
			{	slowly_move_fwd();
				loop_until_bit_is_set(ADCSR,ADIF); }
		}

		loop_until_bit_is_set(ADCSR,ADIF);
		while(ADCH == 127)
		{
			loop_until_bit_is_set(ADCSR,ADIF);
			home_base_sonar_init();
```

```
			while(distance > 1 && distance < 3)

			{
				send_to_LCD("I see it!");
				stop();
				home_base_sonar_init();
				//count=0;
			}
		}
	}

/* General Delay Loop */

void delay(int t)
{
	int j = 0;
	while (t--)
		while (--j);
}


/************************ LCD Initialization ****************************/

void lcd_init (void)
{
	delay(1);
	clr_display();			// clearing display
	disp_control();		// turning display on, setting cursor to show up and blink
	fct_set();
	entry_mode();			// getting ready for entry
}

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

/************************** Move Forward ***************************/

void move_fwd(void)
{
	OCR1A=200;
	OCR1B=1912;
}

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

/************************* Slowly Move Forward ***********************/
```

```
void slowly_move_fwd(void)
{
        OCR1A=1420;
        OCR1B=1600;
}

/********************** Back Up **********************************/

void back_up(void)
{
        OCR1A=1912;
        OCR1B=200;
}

/******************** Left Turn ************************************/

void turn_left(void)
{
        OCR1A=1912;                                     // left servo, full speed bkwd
        OCR1B=1912;                                     // right servo, full speed fwd
}

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

/********************** Slowly Turn Left **********************************/

void slowly_turn_left(void)
{
        OCR1A=1600;
        OCR1B=1600;
}

/******************** Right Turn ************************************/

void turn_right(void)
{
        OCR1A=200;                              // left servo, full speed fwd
        OCR1B=200;                              // right servo, full speed bkwd
}


/******************** Slow Right Turn ************************************/

void slowly_turn_right(void)
{
        OCR1A=1450;
```

```
	OCR1B=1450;
}
/*****************************************************************/

/******************** Stop Servos *********************/

void stop(void)
{
	OCR1A=0;
	OCR1B=0;
}
/************** Timer 1 A and B Initialization ************************/

void timer1_ab_init(void)
{
	DDRB=0xE0;
	PORTB=0x00;

	TCCR1A=0xA8;			// Gets timer channels A,B,C ready for use
	TCCR1B=0x12;
	TCCR1C=0x02;
	TCNT1H=0x00;
	TCNT1L=0x00;
	ICR1H=0x4A;
	ICR1L=0xAA;
	OCR1AH=0x00;
	OCR1AL=0x00;
	OCR1BH=0x00;
	OCR1BL=0x00;
	OCR1CH=0x00;
	OCR1CL=0x00;
}
/*******************************************************************/

/* About a 8us delay */

void delay_10us(int t)
{
	int n = 800;
	while (t--)
	{
		n=800;
		while(n!=0)
		{	n--; }
	}
}
```

```
/*Initializing to clear display*/

void clr_display(void)
{       PORTC=0x01;
        delay(1);
        command();
        delay(1); }

/*Latching Command Data to Instruction Register*/

void command(void)
{       PORTA=0x20;                             //PortA = 001xxxxx,
RS=0,R/W=0,E=1
        PORTA=0x00;         }                   //PortA = 000xxxxx,
RS=0,R/W=0,E=0

/*Latching Character Data to Data Register*/

void data(void)
{       PORTA=0xA0;                             //PortC = 101xxxxx,
RS=1,R/W=0,E=1
        PORTA=0x80;         }                   //PortC = 100xxxxx,
RS=1,R/W=0,E=0

/*Entry Mode Set*/

void entry_mode(void)
{       PORTC=0x06;                             // enables entry of characters
        delay(1);
        command();
        delay(1); }

/*Display on/off control*/

void disp_control(void)     //Sets display on/off,cursor on/off,blinking of cursor on/off
{       PORTC=0x0f;
        delay(1);
        command();
        delay(1); }

/*Function Set*/

void fct_set(void)
{       PORTC=0x38;
        delay(1);
```

```
        command();
        delay(1); }             //sending data @ 8 bits, 2 lines, 5x10 dots (32 char)


/*Sending characters to the LCD*/

void send_to_LCD(char lcd[32])
{ int i=0;
        for(i=0; lcd[i]!='\0'; i++)
        {
                PORTC=lcd[i];                   //sending each character to LCD
                data();                         //latching LCD data
                delay(1);                       //delay between each character sent
        }
}

/************** Initiating SONAR and counting for distance *****************/

void left_sonar_init(void)
{

        PORTD=0x80;                                             // PA7 is PULSE Trigger
input
        delay_10us(4);
        PORTD=0x00;                                             // toggling PA7

        delay_10us(30);                                 // 200us delay before receiver
circuitry is enabled
        int timeout = 50;
        while((PIND&0x08)==0x00 && timeout)
        {
                timeout--;              // timeout makes sure it doesn't freeze if no response
                delay_10us(1);                  // i.e. putting hand in front of sonar
        }
                                // while ECHO is low, it is waiting for ECHO receive

        left_dist = 0;
        while((PIND&0x08)==0x08) // while ECHO is high, it is determining distance
        {
                left_dist++;
                delay_10us(6);
        }
}


void right_sonar_init(void)
```

```
{

        PORTD=0x40;                         // PA7 is PULSE Trigger input
        delay_10us(4);
        PORTD=0x00;                         // toggling PA7

        delay_10us(30);             // 200us delay before receiver circuitry is enabled
        int timeout = 50;
        while((PIND&0x04)==0x00 && timeout)
        {
                timeout--;       // timeout makes sure it doesn't freeze if no response
                delay_10us(1);          // i.e. putting hand in front of sonar
        }
        // while ECHO is low, it is waiting for ECHO receive

        right_dist = 0;
        while((PIND&0x04)==0x04) // while ECHO is high, it is determining distance
        {
                right_dist++;
                delay_10us(6);
        }
}

/********************* Initiating SONAR and counting for distance
************************/

void home_base_sonar_init(void)
{

        PORTD=0x20;                 // PORTD7 is PULSE Trigger input
        delay_10us(4);// min delay of 10us; extended to make sure long enough delay
        PORTD=0x00;         // toggling PA7

        delay_10us(30);         // 200us+ delay before receiver circuitry is enabled
                                        //(this is about 240us)
        // if you decrease the # in the delay_10us() it brings the distance closer
                        // to scale or the scale you want it to be
        int timeout = 50;
        while((PIND&0x02)==0x00 && timeout)    // PIND is the controllers way of
                                                //"reading" the port pin
        {
                                // I used PORTD3 for my input pin (ECHO)
                timeout--;              // timeout makes sure it doesn't freeze if no response
                delay_10us(1);        // i.e. putting hand in front of sonar may freeze it, but
                                        //not with timeout
```

```
        }
                              // while ECHO is low, it is waiting for ECHO receive

        distance = 0;         // making sure distance=0 at this point before it increments
        while((PIND&0x02)==0x02) // while ECHO is high, it is determining distance
        {

                distance++;
                delay_10us(6);              // about 50us delay, and everything 50us was
                                                    //about 9mm
        }               // i.e. if distance=1, something is 9mm away depending how you
}                             // scale the delay_10us() as I noted above
/*****************************************************************/
```