University of Florida

Department of Electrical and Computer Engineering

EEL5666

Intelligent Machine Design Laboratory

Sensor Report

Aaron Tucker

March 17, 2005

Sensor Suite

Dusty uses a variety of sensors which it uses to gather information about its environment. Since an overly shy robot will be constrained to repeatedly cleaning the same areas – those most wide open –the design of this robot calls for an aggressive interaction with the environment. Dusty must be able to get up close and personal to clean the objects around it – but without causing environmental or self-inflicted damage.

Dusty must know if it has bumped objects in its environment. Bump sensors placed strategically on the robot can sense any collisions so Dusty can react accordingly. Hopefully, Dusty won't have too many collisions thanks to his forward looking sonar unit, which serves as a main set of eyes to detect upcoming obstacles. But it's not always moving forward – in spot cleaning mode he has to his side. Mounted facing right toward its rear, Dusty has an infrared sensor to detect objects beside it (on the right). This sensor is also used by Dusty to find walls to follow. Additionally Dusty has a ground sensing CDS cell to ensure that if he encounters the edge of the world, he won't fall off. A vacuum robot's environment is complex, and statically mounted sensors can only get Dusty so far. Sitting atop the unit is the eyes-behind-my-head unit, a rotating sensor array that lets Dusty look around the room. Mounted on the EBMHU are an additional sonar unit and infrared unit – redundant units that will help with complex decision making tasks.

Bump Sensors

Dusty's bump sensors are constructed using small momentary push switches connected to pull-up resistors. The final placement is to be determined after further research into Dusty's behaviors. It is expected that areas prone to collisions will be the extremities of the vacuum nozzle and the rear of the unit. Collision to the side of the unit are not expected based on the robot's behavior, however this remains to be proven. Triggering of these sensors is a medium-priority event: sometimes a bump might be a good thing for Dusty. It may mean a wall is in front of him, and he loves following walls.

Dusty is designed to be an aggressive robot, and is not shy from tight places. The range-finding sensors (SONAR and IR units) are focused at unit level purposefully – the environment above the robot is of little importance. Dusty as a special bump sensor used to detect height. To ensure clearance under low structures, Dusty uses an antenna connected to a pair of bump sensors. Mounted near the front, the antenna allows Dusty to peak his nozzle under things without getting himself stuck. Triggering of the antenna sensor is a high-priority event because it threatens Dusty's safety. If the antenna is pressed forward or backward, Dusty knows it must change direction.

CDS Cells – Ground Detection

Dusty is a fearless robot but he must keep his wheels on the ground. Two downward facing CDS cells mounted at wheel height sense the amount of light under the robot. Light shielding is placed around the cell so that, at ground level, very little light is detected. When he is reset Dusty must be placed on level ground so a baseline

calibration measurement is taken. If the ground should disappear from beneath him, perhaps at the edge of a stairwell, light enters from the bottom of the robot. When the level deviates more than a specified tolerance from the baseline measurement Dusty is programmed to turn around for safer ground. Triggering of the CDS cells is a high-level event since it threatens Dusty's safety.

Sonar

Dusty has two Devantech SRF04 Ultrasonic Range Finder units. One forward facing unit serves as Dusty's permanent forward vision. The second unit is mounted to the eyes-behind-my-head unit which rotates to extend Dusty's field of vision. The SRF04 is advertised to measure accurately from 3 cm to 3 meters, and may be purchased from Acroname.

Operation of the SRF04 is based on sonic principles. After sending out an ultrasonic pulse, the unit outputs a pulse when a return echo is detected. Timing the delay between the sending of the pulse and the receiving of the echo allows distance to be measured to the nearest object large enough to produce an echo. According to the SRF04 documentation the pulse travels at a rate of .9 ft/msec.

The SRF04 is interfaced with the Atmega128 using a timer. Timing begins when the pulse is sent. The controller polls the output echo lead until a signal is detected, upon which the timer is stopped. The elapsed time is a direct measurement of the distance to

the nearest measurable object. Although a 16-bit timer would allow Dusty to see farther, he is only concerned with his nearest surroundings, so an 8-bit timer can be used.

While converting this timer count value to common units of length measurement is useful to humans, it is meaningless to Dusty who is an expert at vacuuming but knows nothing of the metric system. Timer measurements (raw data) ranging from 0 (extremely close) to 255 (no object detected) are used by the robot for all its calculations. This is consistent with all other robot systems.

Initial testing of the sonar unit showed its output was reasonably steady for large, static objects. Variations occurred in output, and violent swings were observed from time to time. In dynamic, multi-object environments the sonar output quality was diminished. In these situations multiple echoes are detected. After the first sonar pulse, an echo is detected from the nearest object. The sonar pings again expecting another echo from this second ping. On the contrary, the echo it hears is from the first pulse, echoing from a distant feature. This situation is remedied by increasing the delay between subsequent sonar reads, allowing all the echoes to dissipate before restarting the process.

To test the sonar, a short program was written to take sonar readings in 250ms intervals, writing them to the LCD screen. A white card placed in front of the sonar was used to simulate an object. The distance from the card to the sonar was measured with a tape measure and compared to the output value. The results which can be seen in Appendix A

suggest a strong linear trend (with approximate slope of 3/8) for distances over 3 ft, which is the maximum range Dusty needs.

Two levels of filtering have been developed to minimize the effects of outlier data. The first filter level is a low-pass filter which in effect takes an average of a specified number of *n* measurements. The measurements are stored in an array and then averaged, and the average is reported.

To concentrate on only the nearest objects, and remove outlying measurements, the second level of filtering compares each of the *n* individual measurements to the calculated average. If a measurement deviates from the average by more than a specified value it is identified as an outlier. Outliers are replaced with the average, effectively removing it from the data set. At the conclusion of the process a new average is calculated and reported. A large deviation tolerance has yielded acceptable results (difference < 50) while smaller tolerances have limited the resolution to unacceptable levels. Appendix B has an example of filtering in effect.

Dusty is designed to use the SRF04 as its primary method of sensing its environment due to its reliability in different environments. Its performance is not affected by outside noise or light levels. It is not sensitive to the color of objects or their luminosity. Testing with a variety of materials indicates low absorption by all common living-room materials.

Infrared Sensors

Dusty has been given two Sharp GP2D120 Infrared Sensors. One unit is placed on the right, rear side of the robot. This position is an auxiliary view for wall following and spot cleaning. Wall following is always performed with the wall on the right side of the robot. In spot cleaning mode the robot moves forward and backwards while inching rightward across a room, mimicking human vacuuming motions. The auxiliary view will be valuable for these behaviors. An identical sensor is mounted to the eyes-behind-my-head unit.

The GP2D120 outputs a single analog signal with a range from 0 to 2.5V. The unit is operational once connected to a power supply. The D120 is unique for its special lenses which focus detection within a closer radius than other models. Dusty is only concerned with its immediate surroundings and therefore does not need a long vision range.

The output to the GP2D120 is connected to the analog-to-digital converter on Dusty's controller. Measurements a reported on a scale from 0 to 255, however the range of actual values is 48 (no object detected) to 192 (extremely close). Like the Sonar unit, Dusty makes no attempt to convert these values to common length measurements. Instead reported values are compared to the range of values, in this case 48-192, to get a sense of how close an object is.

Testing showed output that was more stable, less vulnerable to fluctuation, than the sonar unit. On the downside, the IR unit is sensitive to color – darker objects absorb more light and are measured slightly farther than white objects. This effect is minimized at short

distances, which is ideal since Dusty's aggressive programming only takes his immediate surroundings into account.

To test the IR, a short program was written to take IR readings in 250ms intervals, writing them to the LCD screen. A black cloth placed in front of the IR was used to simulate an object. The color and texture of this material was expected to be the most absorbent type of environmental object Dusty might encounter. The distance from the shirt to the IR canon was measured with a tape measure and compared to the output value. The results which can be seen in Appendix A suggest useful output from 0 inches to 15 inches. In this region a linear trend can safely be approximated.

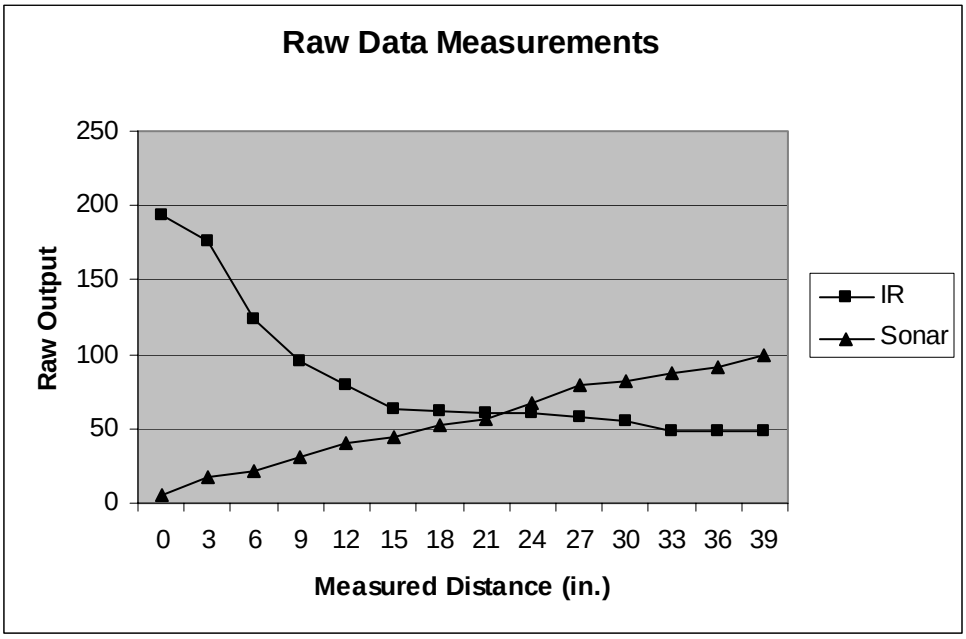
Two levels of filtering have been developed to minimize the effects of outlier data. The first filter level is a low-pass filter which in effect takes an average of a specified number of *n* measurements. The measurements are stored in an array and then averaged, and the average is reported.

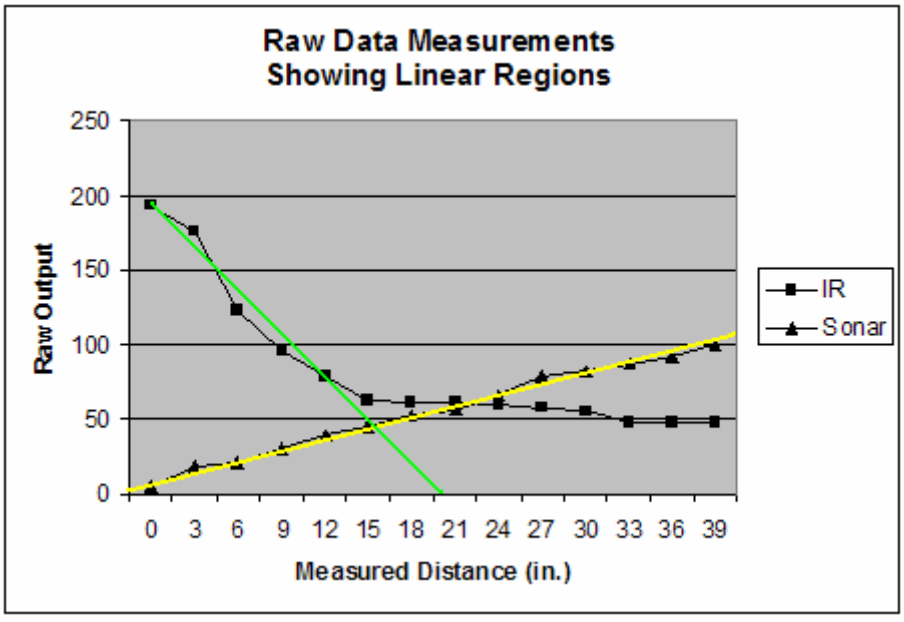
To concentrate on only the nearest objects, and remove outlying measurements, the second level of filtering compares each of the *n* individual measurements to the calculated average. If a measurement deviates from the average by more than a specified value it is identified as an outlier. Outliers are replaced with the average, effectively removing it from the data set. At the conclusion of the process a new average is calculated and reported. A large deviation tolerance has yielded acceptable results

(difference < 50) while smaller tolerances have limited the resolution to unacceptable levels. Appendix B has an example of filtering in effect.

Appendix A:
Experimental Results

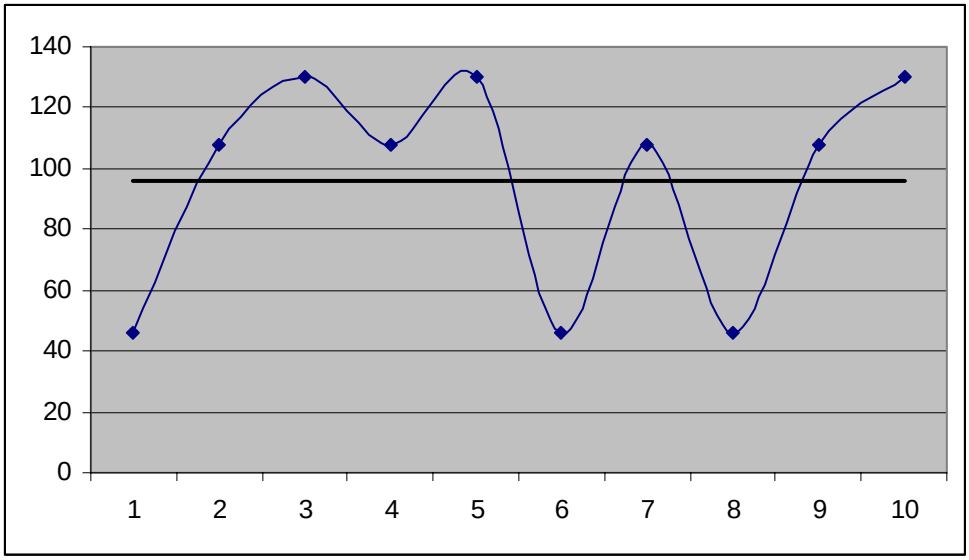
Measured Distance	IR	Sonar
0	193	5
3	176	18
6	123	21
9	96	31
12	79	40
15	63	45
18	62	53
21	61	57
24	60	67
27	58	79
30	55	82
33	48	87
36	48	92
39	48	100



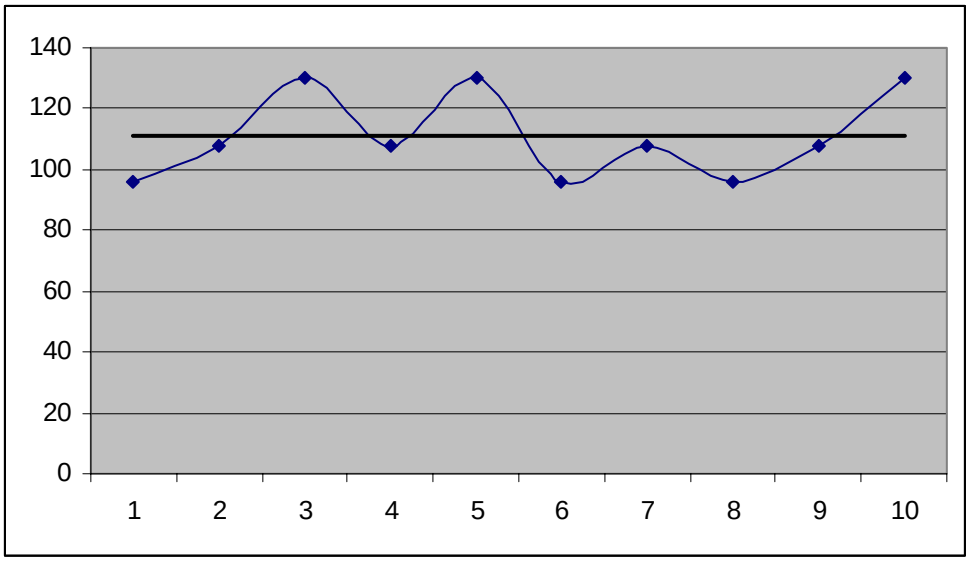


Appendix B:
Filtered Output Examples

Raw Data
46
108
130
108
130
46
108
46
108
130



Filtered Data
96
108
130
108
130
96
108
96
108
130



Appendix C:
Sample Source Code

```
void ir_test0()
{
        lcd_clear();
        lcd_putsf("Analog Testing Program");
        delay_ms(3000);

        while(1)
        {
                lcd_clear();
                lcd_int_write(ADCW);
                delay_ms(250);
        };
}
```

```
int sonar_test()
{
        //Requires Two Pins!  PortB(0:1)
        //PORTB0        Pulse Enable (DDR Output)
        //PORTB1        Echo         (DDR Input)

        unsigned int Count=0x00;          //Keeps track of overflows
        unsigned int Result=0x00;         //Final Timer Value

        TCCR0=0x00;         //Stop Timer
        TCNT0=0x00;         //Clear Timer
        PORTB|=0x01;        //Set Pulse Enable High
        delay_us(12);       //12 uSec Delay
        PORTB&=0xfe;        //Set Pulse Low
        TCCR0=0x07;         //Start Timer

        while (PINB.1==0)   //Poll Echo For Rising Edge
        {
                if (TCNT0==255)         //Check for Overflow
                {       Count+=256;     //Increment overflow counter
                        TIFR|=0x01;     //Clear Overflow Flag
                };
        };

        while (PINB.1==1)   //Poll Echo For Falling Edge
        {
                if (TCNT0==255)       //Check for Overflow
                {       Count+=256;        //Increment overflow counter
                        TIFR|=0x01;     //Clear Overflow Flag
                };
        };

        TCCR0=0x00;         //Stop Timer
        Result=TCNT0;       //Save Result
        delay_ms(10);       //Manditory Delay

        return (Count+Result);
 }
int sonar_filter()
```

```
{
        int test;
        int answer;
        int fanswer=0;

        test=0;
        answer=0;
        while(test<SONAR_NUMSAMPLES)
        {
                SonarValues[test]=sonar_test();
                answer+=SonarValues[test];
                test++;
        };

        answer = answer>>4;

        test=0;

        while(test<SONAR_NUMSAMPLES)
        {
                if ((SonarValues[test]-answer>Sonar_Smooth)||(answer-
SonarValues[test]>Sonar_Smooth))
                SonarValues[test]=answer;

                fanswer+=SonarValues[test];
                test++;
        };

       return fanswer;
}
```

```
int ir_filter()
{
        int test=0;
        int answer=0;
        int fanswer=0;

        while(test<IR_NUMSAMPLES)
        {
                IRValues[test]=ir_test0();
                answer+=SonarValues[test];
                test++;
        };

        answer = answer>>4;
        test=0;

        while(test<SONAR_NUMSAMPLES)
        {
                if ((IRValues[test]-answer>IR_Smooth)||(answer-
IRValues[test]>IR_Smooth))
                IRValues[test]=answer;

                fanswer+=IRValues[test];
                test++;
        };
```

```
       return fanswer;
}
```

```
void sonar_motor_test()
{

      //Combines object avoidance with speed control
      //Uses sonar to drive up to an object without hitting it
      //Uses a throttle to set a desired speed
      //Actual speed is controlled to avoid rapid changes

       int val1;
       Motor_Throttle_L = sonar_test();    //Set throttle to distance

       Motor_Throttle_L = Motor_Throttle_L * 2;

      //Software Control of Viewing Range
       if (Motor_Throttle_L < Motor_Shyness) Motor_Throttle_L = 2;
       if (Motor_Throttle_L > Motor_Aggressiveness)
            Motor_Throttle_L = 255;

       val1 = OCR2;

       //Set speed
       val1=(int) val1*Motor_Smooth+Motor_Throttle_L)/(Motor_Smooth+1);

       OCR2 = val1;

       lcd_gotoxy(0,0);
       lcd_putsf("Throttle: ");
       if (Motor_Throttle_L < 0x64)
         lcd_putsf(" ");

       lcd_int_write((int) Motor_Throttle_L);
       lcd_gotoxy(0,1);
       lcd_putsf("Speed: ");
       if (OCR2 < 0x64)
         lcd_putsf(" ");

       lcd_int_write((int) val1);
       delay_ms(10);
}
```