Lisa Anthony
Robot Building Lab
Graduate Project: Final Report
Due December 17, 2001

1 Task

I propose to investigate the accuracy of the sonar sensors we have been using in class; many groups received readings which did not make sense, given certain assumptions that were made about the operation of these sensors. To attain a more reliable sonar sensor would be both interesting and practical for future use of these sensors.

1.1 Confounding Variables

In the graphs of the elementary in-class experiments conducted with the sonar sensors, several problems or inconsistencies were apparent (see *plan.ps* and *design.ps* for details), implying that the situations in which either *sonar_closeup()* and *sonar_sample()* are most useful and accurate are not fully understood. I performed many experiments using both sonar sampling methods, trying to control for and isolate the following possible confounding variables:

- Distance of obstacle(s) from sonar/robot
- Timing interval between subsequent sonar readings
- Effect of using the two methods together instead of just one
- Effects of the servo motor motion on sonar accuracy
- Variability among individual sonars
- Blanking time intervals used in each method

2 Hardware

The only hardware used for this project was a sonar sensor mounted on a servo motor attached to a HandyBoard, in similar configurations to that used in class. No drive train or other sensors were needed for the experiments; although they were attached to the robot used, they were simply idle. The experiments were conducted in the same manner used in class: the sonar/servo/HandyBoard combination took readings of the world in a 180-degree sweep and sent them across a serial line to a PC.

3 Software

I was able to set up one control program which used easily changed global variables to set up each experiment I needed to run. The code (in three files, *main.c*, *readings.c*, *dump.c*) set up how the sonar would take the readings, obtained these readings and stored them. At the end of each experimental run, the results were transmitted to a PC via TeraTerm Pro; I imported them into Microsoft Excel to graph them.

The end result of this project was intended to be a "filter" function to add to the sonar library that would mask from its users the procedure used to obtain the most accurate sonar reading possible. However, it turned out that this function was not really necessary; our hypothesis had been that there were situations in which one method was much more effective than the other, but in reality, I was able to achieve equal performance from each method after my alterations to the *sonar.c* file (discussed in Section 4).

3.1 Interactive-C Libraries

Here is a list of the methods used from the default IC libraries:

- *sonar.c*—for sonar control
  - *sonar_init()*
  - *sonar_closeup()*
  - *sonar_sample()*
- *servo.c*—for servo motor control
  - *servo_on()*
  - *servo_deg()*
- *serialio.c*—for serial communication to a PC
  - *disable_pcode()*
  - *serial_putchar()*
- *printdec.c*—for printing decimal numbers via serial communication to a PC
  - *printdec()*

3.2 Control Program Modules

- **Serial Communication Module:** In previous labs, my team had built methods to use in taking collected sensor data and dumping it to a serial line, which I also used in this project.
- **Experimental Control Modules:** The software template for running sonar-based experiments and collecting data from them had already been implemented by our team in class, and was the skeleton to which I added the code to run the experiments I needed.
  - *main.c* - in charge of setting up current experiment by calling *readings()* with appropriate args; then dumps data to serial line using *dump()*
  - *readings.c* - takes *SONAR_SAMPLES* (defined in *main.c*) number of readings using both *sonar_closeup()* and *sonar_sample()* methods based on command line args
  - *dump.c* - dumps arrays of readings to the serial line; one reading per line, readings from *sonar_closeup()* followed by readings from *sonar_sample()*; a line with 3 asterisks marks the data separation

3.3 Alterations to sonar.c

Via experimental analysis, it was determined that the default blanking time used in *sonar_closeup()* in *sonar.c* was too short and was causing the method to often return funky data. I altered the default blanking time from 0.5ms to 0.7ms based on experimental discovery, and the method's performance immediately drastically improved. This is explained in more detail in Section 5.

4 Experimental Results

Each experiment set and its important characteristics, along with a brief description of the results, is enumerated below. In the discussions and tables below, *swait* is the amount of time to wait between immediately subsequent sonar readings, *vwait* is the amount of time to wait between a sonar reading and servo movement, *iwait* is the amount of time between readings which are to be averaged, and *cwait* is the amount of time each method "blanks" before allowing a return pulse to be recognized. Fields marked with a "*" are discussed in detail in the Results section of each experiment set section.

Description of Set	Determining amount of time to wait between sonar readings
# of Experiments Run	10
Obstacle Distance	45"
Sweeping?	Yes
Averaging?	No
Sampling Routine Used	sample() then closeup() interchanging
swait	*
vwait	1.0s
iwait	n/a
cwait	Default (2.38ms for sample(), 0.5ms for closeup())

Figure 1: Experiment Set 1.

4.1 Experiment Set 1: Determining Appropriate Amount of Time to Wait Between Sonar Readings

Results The *swait* values I tested were as follows: 1.0s, 0.5s, 0.25s, 0.2s, 0.15s, 0.1s, 0.05s, 0.025s, 0.02s, and 0s. Preliminary experiments were very promising; both methods were returning identical values for the same positions (unlike in class experiments, where there was often a parallel gap between the values return by each method). The *sonar_sample()* method returned good values for every *swait* used; however, the *sonar_closeup()* method's values began spiking too low at about *swait*=0.2s. See *swait.xls* for graphs and data.

4.2 Experiment Set 2: Determining Appropriate Amount of Time to Wait Between a Sonar Reading and Servo Movement

Description of Set	Determining amount of time to wait between a sonar reading and a servo movement
# of Experiments Run	6
Obstacle Distance	45"
Sweeping?	Yes
Averaging?	No
Sampling Routine Used	sample() then closeup() interchanging
swait	0.25s
vwait	*
iwait	n/a
cwait	Default

Figure 2: Experiment Set 2.

Results Because the last *swait* value which worked for both methods in Experiment Set 1 was 0.25s, I used that value as the default lowest one in most subsequent experiments, including this set. The *vwait* values I tested were as follows: 0.5s, 0.25s, 0.10s, 0.05s, and 0s. The *sonar_sample()* method returned good values for every *vwait* used; however, the *sonar_closeup()* method's values began spiking too low at about *vwait*=0.10s. See *vwait.xls* for graphs and data.

4.3 Experiment Set 3: Averaging Several Readings at One Position to Correct for Possible Random Error

Results At this point, we had a hypothesis that perhaps the low-spike values being returned by *sonar_closeup()* might be random errors which could be corrected by taking several readings at the same spot and averaging them together to cancel out any random erroneous readings. I decided to run averages of 3 readings at each spot in the sweep. The *iwait*

Description of Set	Averaging several readings at 1 position to correct for random errors
# of Experiments Run	3
Obstacle Distance	45"
Sweeping?	Yes
Averaging?	Yes
Sampling Routine Used	sample() then closeup() interchanging
swait	1.0s
vwait	1.0s
iwait	*
cwait	Default

Figure 3: Experiment Set 3.

values I tested were as follows: 0.05s for both methods, 0.05s for *sonar_sample()* and 0.10s for *sonar_closeup()*, and 0.10s for *sonar_sample()* and 0.20s for *sonar_closeup()*. The reason I started using different intervals for each method was that *sonar_sample()* was returning good values for any interval, but *sonar_closeup()* was acting up even worse than before. I thought increasing the *iwait* might help alleviate this problem, but it did not. See *averaging.xls* for graphs and data.

4.4 Experiment Set 4: Running sonar_sample() Independently

Description of Set	Running sonar_sample() independently
# of Experiments Run	10
Obstacle Distance	45"
Sweeping?	Yes
Averaging?	No
Sampling Routine Used	(just sonar_sample() used)
swait	n/a (0s)
vwait	*
iwait	n/a
cwait	Default

Figure 4: Experiment Set 4.

Results I had started to worry that my results might be affected by the fact that I was always running *sonar_sample()* before *sonar_closeup()* during my sweeps; I ran this set of experiments to try to isolate out any effect one method might have on the other. I ran a normal experiment with just the *sonar_sample()* method, using different waiting times before moving the servo to the next spot. The *vwait* values I tested were as follows: 0s, 0.25s, 0.10s, 0.05s, 0.025s, 0.01s, 0.005s, 0.004s, 0.002s, and 0.001s. The results showed that the *sonar_sample()* method performed very well with no spiking at all down to a *vwait*=0.025s, and very little spiking down to *vwait*=0.001s. See *sample.xls* for graphs and data.

4.5 Experiment Set 5: Running sonar_closeup() Independently

Results I ran this set of experiments to try to isolate out any effect one method might have on the other; I ran a normal experiment with just the *sonar_closeup()* method (see above), using different waiting times before moving the servo to the next spot. The *vwait* values I tested were as follows: 0.25s, 0.5s, 1.0s, and 5.0s. These were the most disheartening results, because even while waiting 5 seconds between servo movement and sonar reading *and* taking averages with an *iwait*=5.0s, the values returned were basically complete junk. See *closeup.xls* for graphs and data.

Description of Set	Running sonar_closeup() independently
# of Experiments Run	7
Obstacle Distance	45"
Sweeping?	Yes
Averaging?	Yes for some, No for others
Sampling Routine Used	(just sonar_closeup() used)
swait	n/a (0s)
vwait	*
iwait	0.10s for some, 5.0s for others
cwait	Default

Figure 5: Experiment Set 5.

Description of Set	Isolating the servo's influence
# of Experiments Run	6
Obstacle Distance	45"
Sweeping?	No
Averaging?	Yes for some, No for others
Sampling Routine Used	*
swait	1.0s for some, 0.25s for others
vwait	n/a
iwait	0.10s for sample()/0.20s for closeup()
cwait	Default

Figure 6: Experiment Set 6.

4.6 Experiment Set 6: Isolating the Servo's Influence (no sweeping)

Results Due to the very bad results the experiments in Set 5 yielded, it was clear that there were still too many variables that could be affecting the performance of *sonar_closeup()*. I ran this set of experiments to try to isolate out any remaining effect the servo might be having; I ran several experiments without moving the servo at all. The obstacles in these experiments were at 90 degrees in front of the sonar, 45 inches away. The sampling routine used in each experiment was different as well: I used *sonar_closeup()* followed by *sonar_sample()* for one, vice versa for another, and then ran all *sonar_closeup()* readings followed by all *sonar_sample()* readings, and then vice versa. While most of these experiments just took one reading at each position, I also ran a final experiment that averaged 3 readings for each position. The *swait* values I tested were as follows: 1.0s and 0.25s. The interchanging routines yielded the worst results, with many low spikes in the *sonar_closeup()* method's values. However, none of the runs had really good results for the *sonar_closeup()* method, so based on that, it seemed safe to assume that the servo movements were not the culprit in the bad results for Experiment Sets 4 and 5. See *nosweep.xls* for graphs and data.

4.7 Experiment Set 7: Adjusting sonar_closeup()'s Blanking Timing

Results After Experiment Set 6, it began to be apparent that whatever was causing *sonar_closeup()*'s performance to be so poor almost all the time, must be isolated within the code itself, as we had tested for every confounding variable we could think of. Closer inspection of the *sonar_closeup()* method's implementation revealed that, while the *sonar_sample()* method used blanking for 2.38ms, and the closeup() method was supposed to not use blanking, it actually did blank for 0.5ms. For this set of experiments, then, I adjusted the *sonar.c* code to increase the amount of blanking time. I ran two experiments without sweeping or averaging to re-create experiments from Experiment Set 6, with *cwait*=2.38ms, to ensure that the results emulated those of *sonar_sample()*. I then ran several experiments with varying values of *cwait*, during which I did use sweeping and averaging. That made these experiments similar to the ones run in Experiment Set 3, so that I could compare the results and be sure that my changes to the blanking interval were having an effect. The *cwait* values I tested were as follows: 2.38ms, 2.00ms, 1.00ms, 0.75ms, 0.70ms, 0.65ms, and 0.625ms. I gradually decreased the

Description of Set	Adjusting sonar_closeup()'s default blanking period
# of Experiments Run	9
Obstacle Distance	45"
Sweeping?	Yes for some, No for others
Averaging?	Yes for some, No for others
Sampling Routine Used	*
swait	1.0s
vwait	1.0s
iwait	0.10s
cwait	*

Figure 7: Experiment Set 7.

blanking interval until the results started to look goofy again. The lowest value for *cwait* that still yielded good results was 0.70ms. Based on this, I altered the *sonar.c* library to use this value within the *sonar_closeup()* method permanently. See *blanking.xls* for graphs and data.

4.8 Experiment Set 8: Different World Arrangements

Description of Set	Different world arrangements
# of Experiments Run	10
Obstacle Distance	*
Sweeping?	Yes
Averaging?	Yes
Sampling Routine Used	closeup() then sample() interchanging
swait	1.0s for some, 0.25s for others
vwait	1.0s for some, 0s for others
iwait	0.10s for some, 0.05s for others
cwait	2.38ms for sample(), 0.70ms for closeup()

Figure 8: Experiment Set 8.

Results Satisfied with the results from the blanking experiments, the last set I ran were designed to survey the performance of the *sonar_sample()* method and the new *sonar_closeup()* method in different world configurations. I tested the methods with obstacles at 12 inches, 5 inches, 4.75 inches, 4 inches and 1.75 inches. The blanking interval of 0.70ms now being used in *sonar_closeup()* should have allowed the minimum distance at which an obstacle could be detected accurately to be 4.75 inches, which was backed up in the experimental results. I then did cursory experiments with other obstacle shapes: an obstacle angled at +45 degrees (away), at -45 degrees (toward), a convex-cornered obstacle, and a concave-cornered obstacle. These experiments were merely cursory due to time constraints; further investigation needs to be done with these variables to be able to make intelligent assumptions about the behavior of each sampling method under these circumstances. See *obstacles.xls* for graphs and data.

4.9 Experimental Conditions

As stated in the project plan, it is necessary to keep careful records of the world map used in these experiments, for reproducibility's sake as well as to ensure that the obtained data can be later analyzed in the context of the experimental setting. See Figure 9 for a diagram of the world used in most experiments (except for Experiment Set 8).

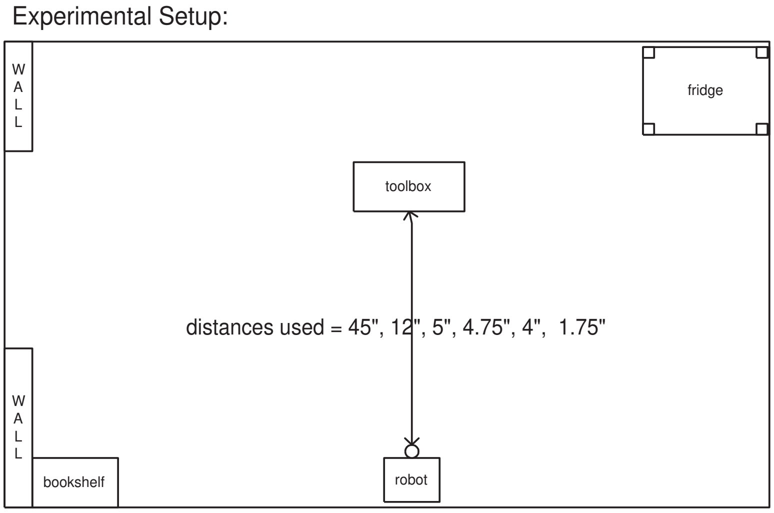

Figure 9: Layout of the World Map Used in Most Experiment Sets.

5 Analysis of Results

This project's progress has determined, through extensive empirical study, that the *sonar_sample()* method is really very accurate under most circumstances (regarding timing specifically) for obstacles above 12 inches away. The *sonar_closeup()* method, on the other hand, in its default form is rather useless. I altered the default blanking period used in that method from 0.5ms to 0.7ms and saw immense improvements. I recommend using the altered *sonar_closeup()* especially for obstacles under 12 inches (but according to my results, above 4.75 inches); *sonar_sample()* now is not any more useful, but can be used to support or average values obtained with *sonar_closeup()*.

6 Open Issues and Possible Future Improvements

Changes in Variables Studied In my original project plan, the confounding variables I had intended to study were level of battery power, movement of the robot itself using its DC motors, interference caused by taking sonar readings too close in time together, movement of the sonar atop the servo motor, and orientation or distance of the obstacle(s). Time constraints as well as intermediate results which suggested other avenues of exploration (i.e., the blanking time interval) might yield more interesting results causing me to abandon these other issues; in future, perhaps another graduate student's final project will be able to address them in more detail, if the sonar sensors continue to give people trouble.

Obstacle Orientations and Materials Because I was only able to run cursory experiments with other obstacle shapes, further investigation needs to be done with these variables to be able to make intelligent assumptions about the behavior of each sampling method under varying world conditions. It is probably also worthwhile to study the sound absorption properties of any future obstacles which may be used in the class; we generally used the toolboxes, which are heavily sound-reflective, but other materials will have different degrees of reflection. This could cause the results of sonar sampling to be completely different.

Delicacy of the Sonar Sensors I ran most of the experiments with one particular sonar sensor, but did run a few of them with other sonars (2 others). The results of these other experiments were convincing that variability among sonars is not as significant a confounding factor as was originally considered possible. However, a major problem with using other sonar sensors is that the sensors are so delicate that they must be handled with extreme care. The wires must not be pulled or stretched, and the sonar must be mounted in such a way on the servo motor that there is enough slack in the wire for the entire 180-degree sweep. Luckily, my main sonar for these experiments held up very well; however, I "broke" the other two I tried using after the first run. It should be emphasized to students that it's really not difficult to break these sensors and that any rough handling could skew your results without the cause being obvious.