Lab Rat

A Maze Navigating Robot

Final Report

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

Name: Daniel Parker
Date: May 5, 2006
TAs: Adam Barnett
Sarah Keen
Instructor: A. Antonio Arroyo

Table of Contents

I. Abstract

Lab Rat is an autonomous robot capable of quickly and intelligently navigating a maze. This report shows how he uses his various sensors, especially IR, to find the end of the maze and also communicate wirelessly with a remote computer.

II. Executive Summary

Maze navigation may seem relatively easy to a human, but humans have excellent motor control and superior vision with built in image processing, pattern recognition and depth perception. Such control and perception is impossible for a robot to achieve with simple servos, basic IR sensors and microprocessor control. Even the simple behavior of going straight between two walls, easy enough in theory, is very tricky to implement in practice. However, Lab Rat does implement the behavior of going straight between two walls, as well as turning 90 degrees in a tight corridor.

This paper describes how Lab Rat is built, how each of the subsystems work, and how they work together to produce the behaviors that make him a maze navigating robot.

The first thing which is described is the body, which I designed in AutoCAD with each of the systems specifically in mind. The platform is small (5 cm radius), which forced me to carefully plan how I would fit all of the components into it.

The next part of this paper describes the actuation with servos. Servos are relatively easy to interface and allow precise control.

The bump sensors are standard on most robots, though my design does have a quirk or two.

The IR ranging sensors are critical to the operation of Lab Rat; they govern all of Lab Rat's movement within the maze.

Many people use photoreflectors to do line following, but I only use them to detect the end of the maze.

The transceiver boards were both tough and rewarding to work with. They sent data to MATLAB, which I was able to analyze with graphs and perfect my control over the IRs.

The appendices contain a picture of the platform design, as well as a circuit diagram/schematic of the wireless to serial module, and all of the MATLAB and C code which makes Lab Rat work.

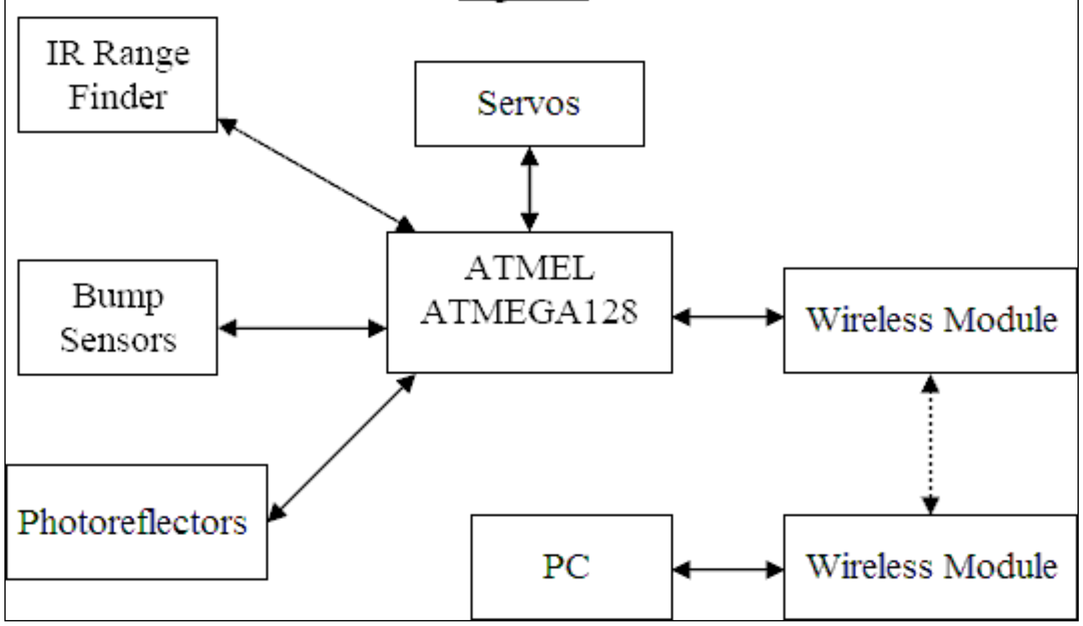
III. Introduction

Mazes have always fascinated me; I used to draw out mazes on graph paper for fun. So when I took IMDL, I decided to design a robot which which would be capable of finding its way through a maze. Lab Rat also demonstrates wireless communication, which was a powerful tool that I used to calibrate the sensors and which also allowed me to “see” what Lab Rat was seeing.

This paper describes the elements - the sensors, systems, and servos - which make up Lab Rat's body and mind. First I describe the overall system, then I describe the specifications and functions of: the platform, the servos, the IR range finders, the bump sensors, the photoreflectors, the wireless module, and the PC interface. I also describe the behaviors that Lab Rat demonstrates and how each of the modules helps to produce the behaviors.

IV. Integrated System

Figure 1



Lab Rat has all the requirements of a complete robotic system: continuous rotation servos for actuation, IR ranging sensors for detecting the walls of the maze, switches as backup sensors for bump detection, and a Hamamatsu Photoreflector to detect the end of the maze. An Atmel Mega103 microprocessor controls the entire system (the ATMega103 is similar to the ATMega 128, but with only one 16 bit timer, an Analog to Digital module which can measure from 0 to 5 V, but not 0 to 2.5 V, and a slower clock speed). Lab Rat also has a wireless transceiver which sends data to MATLAB on a remote PC for system analysis and debugging. See Figure 1 for a diagram which shows how the components are connected together.

V. Mobile Platform

The platform is made of 1/8 inch plywood; I designed it in AutoCAD, then milled it out on a T-Tech machine that was originally designed to mill PCBs but which an enterprising student discovered can be rigged to mill plywood in precise designs as well. The basic design similar to the TJ, which was a design originally by Scott Jannes, but I really only copied the bumper ring and hinged top, the rest is specific to Lab Rat's requirements. The platform is smaller, too, because it had to fit inside corridors which are 18 cm wide; the platform is a circle of radius 5 cm, with 4 cm clearance on both sides: this led to some creative wiring to fit everything inside. The axis of the driving wheels lies just behind the center line. Small balancing casters which rotate freely are positioned at the front and back of the platform. The servos are mounted inside the platform as are the 7.7 V battery which powers the board and sensors as well as the 5.2 V battery which powers the servos (the batteries are custom made of cells from a couple of 9 V batteries) , the photoreflectors are mounted underneath, the ATMega103 board is mounted on the back with the programming header sticking out to allow easy programming, and the front IR sensor is mounted just underneath the circular top and faces forwards. The circular top which is hinged to open at the back contains the bump sensors, power switch, and panic button. Two towers which rise a few inches above the platform mount the four side IR sensors and the transceiver board. Most of the components are glued on with hot glue; the IR sensors, the servos and the casters also use screws. Many pin connections doubled as structural connections, connecting modules on different pieces of the platform both electrically and physically.

If I were to do the project over again, I would make the top of Lab Rat a PCB, since the wiring on the top of the robot caused me so much trouble. I would also have more holes in the platform to allow more wires and various other bits to get through (although this was not really that much trouble since making additional holes in wood is not difficult). Originally, when I designed the platform, I forgot that the servo shaft is not in the center of the servo. Consequently, the wheels ended up behind the center of the robot rather than right at the center. However they are not very far behind the center and still allow Lab Rat to have a very tight turning radius. Furthermore the location of the servos is just right to allow the microprocessor to fit in the back and the battery to fit in the front. It was a mistake, originally, but if I were to do it again, I would do it the same way intentionally. Another change that I would make would be to make the bump ring thicker. Having a thin flexible bump ring has its advantages, but it also tends to slide up and off the switches at inopportune moments and it does not stick out quite far enough in some places. I would also make a PCB for the photoreflectors and design the bottom of the platform to stick down a little lower (at least in some places) to allow the photoreflectors to be closer to the ground (in the current version I use a bit of plastic glued underneath the bottom to get the photoreflectors at the right distance from the ground).

VI. Actuation

I decided to use servos, which are easy to control, because Lab Rat does not require the extra torque or rpms that a motor would provide. The servos I used are hacked to rotate continuously (they were already hacked when I bought them from Acroname.com). Lab Rat moves around by rotating two wheels with two independent servos, each of which has a torque of 3.4 kg-cm (47.2 oz-in) and 43.5 rpm, which allows a maximum speed of 15 cm/s when used with wheels which have a 7 cm diameter. Lab Rat can rotate in place by turning the two wheels in opposite directions. He can also turn slightly to either side by slightly reducing the turning speed of the servo on that side, which allows Lab Rat to travel smoothly down the center of a corridor without jerking around. The speed of the servos is determined by a PWM which occurs every 14 ms. The width of the PWM is slightly more than 1.5 ms for clockwise movement and slightly less than 1.5 ms for counter-clockwise movement - the variable used to control the servo speed has a range of +/- 60 (which is +/- .08 ms), so servo control is precise although Lab Rat never actually goes quite the maximum speed, because he only uses the servos in the range that small changes to the PWM width have an appreciable effect on the speed of the servos. To move forward, the right servo must rotate clockwise and the left servo counter-clockwise.

The code for the servos does not change speeds immediately, but rather uses exponential decay formula (actually an average of the previous value and the current value, weighted towards the previous value) to gradually come to the new speed. The delay is hardly perceptible to a human, but it reduces the strain on the servos and decreases the extra current which is normally drawn by a servo changing speeds. Besides being more power efficient, this also makes it less likely that the servos will spike the ground plane and cause the microcontroller to reset. The servos are also supplied by a separate battery, with only the ground connected to the battery which supplies the microcontroller and the other sensors; together these precautions prevented the servos from ever resetting the microcontroller (except possibly when the battery was extremely low). A more complete solution would be to completely isolate the ground planes as well by using optoisolators to decouple the input PWMs, but this proved unnecessary in practice. Together, the servos draw about 150 mA of current when running at full speed, and maybe twice that when quickly changing direction.

Calibrating the servos was a little bit bothersome. They seem to have a relatively constant speed range, but over a period of days, the center of that speed range shifts and the I have to recalibrate the software again. Sometimes I would have weird errors that turned out to be caused by the servo centers not being calibrated properly. I ended up writing a function which tests different values around the supposed center of each servo to figure out where the true center actually lies. Then I modified a global constant at the top of the code to remember the change.

Early on, I could not get the servos to work: they would spin only in one direction at a constant speed no matter what PWM I gave them. I discovered that the problem was that the grounds needed to be connected.

VII. Behaviors and Sensors

Lab Rat has the following behaviors:

1. Travel straight down a corridor while staying as far away from the walls as possible

2. Detect side passages

3. Turn ninety degrees into a side passage and end up close to straight and in the middle of the side passage

4. Detect the black strip of tape that signifies the end of the maze

5. Wirelessly transmit sensor data to the computer

Lab Rat navigates the maze from the start to the finish. He stays at the center of each corridor and as he comes to an intersection he turns left. When he detects that the floor is no longer white, he stops. While he is traversing the corridor, Lab Rat sends information about the IRs and the direction of the servos wirelessly to a laptop, where it is analyzed by MATLAB and displayed in graphs.

A. Bump Sensors

Lab Rat has six bump sensors: three in the front and three in the back (although two of these are more on the sides than the back). These bump sensors are actually just switches connected to ground on one end and to interrupt pins on the other. The interrupt pins are internally pulled high, so the switches act as active low switches. If Lab Rat bumps into the wall of the maze with the bumper ring which encircles him, one of the switches is pressed and Lab Rat initiates a maneuver to get back onto course. Only three interrupt pins are used to power all six bump sensors -- the bump sensors are wired up similarly to the way that a keypad is wired up, with three interrupts (left, right, center) determining where the interrupt is and with two outputs deciding which of the pair (rear or front) triggers the interrupt.

The bump sensors do not actually implement any of the behaviors, however they do act as a backup system in case the IR sensors fail temporarily.

B. IR Range Finders

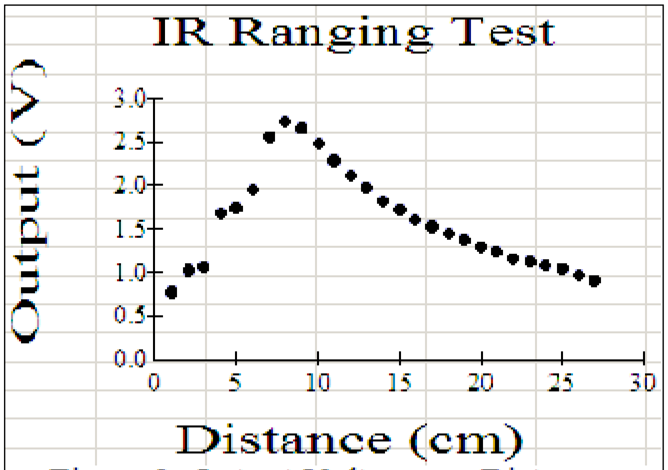


Figure 2: Output Voltage vs. Distance

The most important sensors on Lab Rat are the Sharp IR proximity sensors. Each of the IR sensors contains an infrared LED and an infrared Photodetector which constantly measures the intensity of the light reflected off of objects from the LED; the IR sensor outputs an analog signal in the range of .8 to 3 V. The sensors which I originally ordered from Junon Robotics never arrived, nor did I ever receive an answer to my emails, so I ended up having to order some IRs from Parallax.com relatively late in the semester. The sensors are model GP2D12 and have a working range from 8 cm to 40 cm; the closer the obstacle is, the larger the value output to the A/D port will be (see Figure 2). Lab Rat actually uses the range from 1 cm to 8 cm as well, since the walls of the maze are so close. When Lab Rat is first turned on, he uses his initial readings as calibration for each of the IRs. In general, as Lab Rat gets closer to a wall, the value of the IRs decrease slightly from their original values (generally from 0 to -150); and if an opening is perceived, the values decrease far below their original values (generally around -200). The software used to determine whether a negative number means that the wall is really close or really far is tricky: occasionally, some value could mean either that the wall is extremely close, or that there is an opening. If I were to do the project over again, I would definitely use the Sharp GP2D120 IR sensors as my rear IR sensors: they are supposed to have a range from 4 cm to 30 cm and would make it easier for the software to determine whether a wall or an opening is being perceived. However, I might still use the 8 cm to 40 cm IR sensors on the front, since they have a large variation of voltage with small changes in distance and allow Lab Rat to travel extremely straight.

Lab Rat tries to maintain approximately the same distance to each wall; he needs to keep as far away as possible from either wall and also not turn too much at any given time. This behavior is the extremely tricky to implement, but it is also the quintessential behavior that Lab Rat must have because if the IR sensors get too close to the walls, they confuse the walls with corridors because Lab Rat uses both sides of the IR ranging spectrum. The back sensors follow the same patterns as the front sensors, but are delayed in time, so only the front sensors are actually useful. Additionally, only the IR on the same side as the closer wall is relevant when determining which direction to turn and by how much. If the reading on one of the IR sensors on the side gets slightly smaller than the other (meaning that it is getting close to that wall), then the servo on that side slightly increases speed proportional to the difference between the currently perceived value and the initial value (i.e. the distance to the wall), and the servo on the other side slightly decreases speed so that Lab Rat moves towards the center again. The actual value that the servos change one of the most salient features. If it is too high, then Lab Rat swerves violently back and forth. If it is too low, Lab Rat has some trouble getting back to the middle. I tried implementing a differential velocity control based off of previous IR readings compared to current IR readings, but in the end it did not really seem to have an appreciable benefit on how straight Lab Rat went nor on how far away it stayed from the walls. Perhaps I did not keep track of enough past positions, perhaps I did not affect the movement enough, or perhaps the movement was smooth enough as it was.

Lab Rat uses all of the side sensors to detect side passages. One of the main problems I faced was differentiating between side passages and gaps between wall sections. If one of the front sensors suddenly reads a much smaller value than it had been reading, then Lab Rat assumes that there is an opening of some sort on that side. While it is waiting for the rear sensor to come past the opening, the software remembers the last value that the front sensor had returned and uses that value to determine how to vary the servo values. If both the front and the rear sensors on one side perceive an opening (and continue perceiving an opening for a short while to eliminate possible false readings) then Lab Rat assumes that the opening is a passage rather than a gap between sections and initiates a turn into the opening.

The first part of a turn involves letting the far servo continue to rotate at maximum speed, while turning the close servo backwards slightly. This is continued for a little over half a second, after which the microcontroller starts polling the back IR on the side of the turn to check whether it sees a wall or not. When the back sensor stops seeing a wall, the front sensors should be approximately at the center of the next corridor. When the turn is complete, Lab Rat continues forward for a little more than a quarter of a second, then returns to the main code, where he continues to sample the front IRs to get back on course. After a turn, Lab Rat may not be quite straight; he needs at least a full section of the maze (preferably two) to get going perfectly straight again. Theoretically, a turn which ends up (relatively) straight and in the center of the next passage should be at least as difficult, if not more so than going straight between walls, but in practice I did not have nearly as much trouble with this behavior. Perhaps I just chose the best method and the best delays by accident, or maybe I learned enough from trying to go straight that I was prepared for the challenge of turning. Whatever the reason, Lab Rat can turn well inside a maze (as long as he is going straight approaching the intersection).

Lab Rat sends a command to stop when it reads that the front IR sensor is less than a certain threshold. This prevents him from running into the end of a corridor.

As a side note, the IRs draw quite a bit of current, ~30-40 mA per IR. If a power supply is used which is close to its maximum output current, the output voltage will fluctuate periodically. Batteries don't have this problem, but they do tend to get depleted rather quickly with the large current draw. This also caused a problem with the voltage regulator, since the voltage regulator that came with the ATMega103 board can only handle 100 mA. To fix this problem, I added another 5 volt regulator which can handle 1 A. All of the external devices get their power from the 1 A regulator. This keeps the microcontroller board and the transceiver board from resetting.

C. Photoreflectors

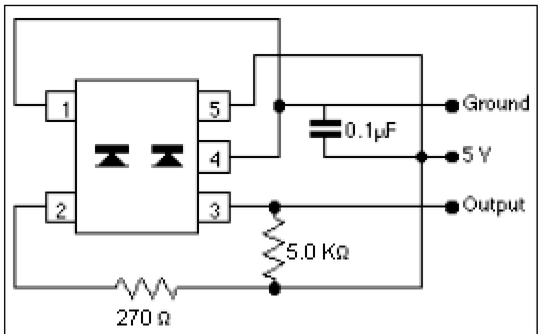


Figure 3: Photoreflector Schematic

The Hamamatsu photoreflectors (available from Acroname.com) are relatively simple, but I spent a while figuring out how to reduce their current draw and discovered that they still work with 270 ohm and 5 kohm resistors (as shown in Figure 3), and only draw about 10 mA of current each (as opposed to upwards of 100 mA with other configurations). They do consistently spike the ground plane (albeit only with a small voltage spike), but a 100 uF capacitor from VCC to GND near each photoreflector tends to even this out and fix the problem. The photoreflectors output 5 V if they see white and 0 V otherwise. The outputs of the two photoreflectors are compared; if only one photoreflector is outputting 0 V, then the output may be faulty. However if both photoreflectors output 0 V for 8 consecutive measurements, then Lab Rat assumes that he has reached the end of the maze. He then sets a variable which stops the servos. This also serves to stop Lab Rat if he is picked up off of the white tiles for any reason.

D. Transceivers

Lab Rat also uses a wireless nRF2401A transceiver chip on the TRF-2.4G board made by Nordic Semiconductors (available at Sparkfun.com) to transmit data wirelessly to a PC which displays detailed sensory results (i.e. IR output and servo values) in a MATLAB graph. In theory, the transceiver chips are easy to operate, and they do have great documentation; in practice, however, I was often frustrated when the transceivers failed to work for some reason or another. The reasons the transceivers failed to work tended to be simple things like coding typos or lack of power or lack of initialization (although I had two boards burn out as well), but the errors were hard to track down because the transceivers either worked or failed to work, generally with no indication of what the problem might be. Towards the end I started recognizing the common errors that kept the transceivers from working properly and had them working almost all the time. At that point they were very useful, but before I really mastered them, they were something of a pain to work with.

Lab Rat clocks in ten bytes of data one bit at a time (most significant bit first) into the transmitter, which uses the ShockBurst mode to transmit data at 256 kbps and 2.442 GHz, at a power of 10 dBm (the receiver has -90 dBm sensitivity). The transmitter transmits a header, then the address 42 (an address agreed upon by the receiver), then the 10 bytes of the message, then the 16 bit CRC (The transceiver can be setup for a 8 bit CRC, but in order to not pick up a whole lot of spurious data, 16 bits is needed). The receiver receives the data, automatically checks the CRC, then dumps the header, address and CRC. The data is then read by the microprocessor, so the chip makes DR1 high, and outputs the data one bit at a time as the clock pin is toggled.

The TRF-2.4G board claims to run off of a maximum of 3.6 V (since it is designed for low power applications), but it actually does seem to run fine at 5 V. The transceiver board requires only 12 uA of current when not transmitting, and only 13 mA when transmitting, or 18 mA when receiving. When compared to the servos, the IRs or the photoreflectors, the transceiver is a very low power device. On the other hand, it tends to easily reset if there is too much noise on the VCC and GND pins. To remedy this, Lab Rat periodically reinitializes the transmitter to make sure that it continues to transmit. I also added external bypass capacitors to further reduce noise. Note that the 1 A voltage regulator also helps by eliminating noise from the power supply at least.

E. PC Module and MATLAB

To receive the data from Lab Rat on my laptop, I designed a printed circuit board in a program called WinQcad. WinQcad has a free evaluation, although it is limited to 100 pin designs; the interface is rather different from PROTEL, though it performs the same functions. It is not quite as powerful nor perhaps as user friendly as PROTEL, but some of the controls were more intuitive.

The PCB I designed was simple, merely connecting a transceiver board, an RS232 chip and a DB-9 serial connector to an ATMega103 board. In order to use the serial communication on the ATMega103, I had to connect pins E0 and E1, which are also used to program the board (since the ISCP programming also uses serial communication to program the board). The end result is that I have to remove the ATMega103 board every time I want to program it. I could have used a jumper which is removed every time the board is programmed, I suppose, but it is easier to remember to put the board back than to put a jumper back, so perhaps this way is better.

The board receives data from the transceiver and outputs the data serially, with an LF character between each 10 byte section of data. This serial communication occurs at 9600 baud, which is relatively slow compared to the transceiver data rate. As a result of this, a delay of about 15 ms is needed between each time the transceiver transmits data. MATLAB reads in the data a line at a time using a COM port toolbox. Some data manipulation is necessary since the data is read in as character bytes, when in fact the IR data is 10 bits to each measurement and some of the measurements are negative (and have to be sign-extended).

Once the data is interpreted, MATLAB stores it and gets some more data. Once it has ten values for each data element, MATLAB creates a new graph with the all of the values on it. The axes of the graph are from 1 to 10 and -100 to 100 so that different graphs can be compared visually. MATLAB also spits out statistical data such as the variance, mean, minimum, and maximum values for each element. When no more data is forthcoming for 3 seconds, MATLAB exits the function and the graphs can be studied and interpreted.

Wireless communication, along with MATLAB's graphs and analysis of the data were so important to figuring out exactly what the IR sensors were seeing that I do not think that I could have succeeded in getting Lab Rat to navigate the maze without them.

VIII. Experimental Layout and Results

The IRs work great when there is no sunlight on them; I did extensive tests on them at night with the robot completely still in the exact middle of two walls. The first thing that I varied was the prescaler for the A to D converter. There was significant improvement in the standard deviation of the IR values when I changed from a prescaler of 32 to a prescaler of 64 (with a clock speed of 6 MHz). There was also a slight improvement when I changed from 64 to 128, but it may be insignificant. However the extra time taken is inconsequential since it only takes 319 us per reading (32768 readings took 10457 ms) using the 128 clock divider. Using a 128 clock divider, I found that the standard deviation of 50 values is 16 at worst and less than 4 at best. An average of the currently measured value and the previous average, weighted towards the previous average (this algorithm results in exponential decay) reduces the worst case standard deviation to 9, while a straight average of the last 32 values only reduces the worst case standard deviation to 13 with a much higher computational cost. I conclude that a 128 prescaler with an average of only the current and previous values, weighted towards the previous value yields the best results. In any case, though, any significant change in the distance of a measured object results in far greater change in the A to D value than is given by random variation. Unless you have sunlight. Then IR sucks. If you have only a little bit of sunlight, you might get away with heavily weighting the average towards the old value, but for any significant sunlight, precision maneuvering is out of the question, and Lab Rat needs precision maneuvering.

The IRs take 2 to 3 seconds after being powered on to output a reliable value. Therefore I inserted a 3 second delay at the beginning of the code to wait for the data to stabilize. Additionally, it seems to take 5 to 10 readings to go from 0 to 200 with a moderately heavy weighting on the average. In order to have a very quick reaction time, with reasonable damping, a medium or light weighting seems to be sufficient.

The final test that Lab Rat undertook was to go straight for 5 sections, then turn left, go straight for another section, turn left again, straight again, then left for a final time and stop at a black piece of tape in the middle of the spiral. Lab Rat was able to navigate the first turn almost every time. Sometimes he was able to navigate all three perfectly. But usually he would end up running into a wall on the second or third turn. The bump sensors generally got him back on the right direction after that, though. I conclude that he did not always have enough time to establish a straight path after the first and second turns. A longer corridor would solve the problem. Slightly more optimized code might as well.

IX. Conclusion

Lab Rat can navigate a simple maze by turning left at every intersection. However, he tends to get confused if the turns are too close together or if the walls are not perfectly straight. This limitation is partly because GP2D12 IR sensors were used rather than GP2D120 IR sensors. A compass may have helped in turning and going straight, but then again, it would have to be a highly precise compass with quick feedback. The IR sensors are almost as good, if not better. The wireless module, though tough to work with originally, was easy to work with by the end and was extremely helpful and reliable. As a platform for demonstrating the capabilities of wireless communication, Lab Rat excelled.

Future work on Lab Rat would include improving his turning ability (or giving him longer corridors to maneuver in), making him turn around in a dead end (which would probably be easy to do) and making him turn right (the mirror image of left, this would be a piece of cake). Additionally, once the turning is perfect, an algorithm for solving the maze would need to be developed. A* would probably not be too hard to implement, and figuring out where in the maze he is can be done already by counting the gaps in the walls. The maze would have to be larger than the current 5x6 rectangle in order to demonstrate the effectiveness of the A* algorithm, though.

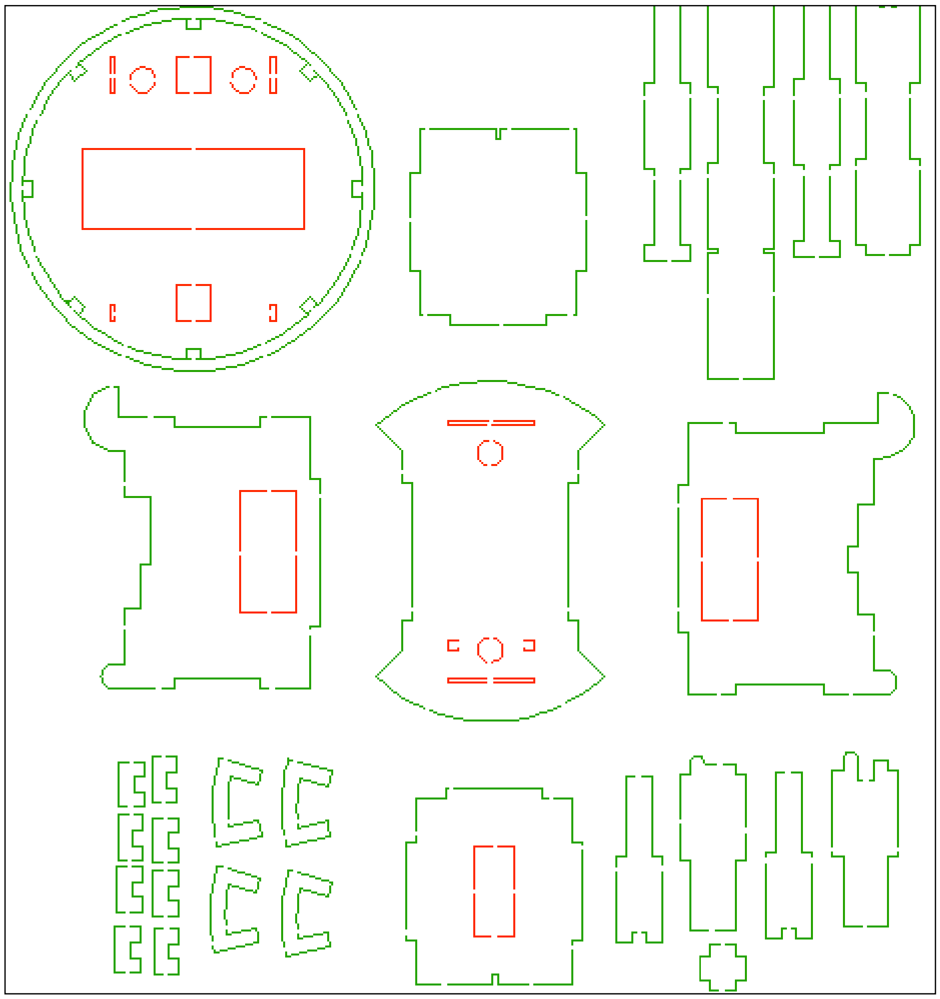
Lab Rat was a project that was far harder to design and build than I had originally anticipated. I am glad to be done and thrilled that I got it to work. I learned a whole lot this semester about robots and the practical application of various electronics, especially about the IR ranging sensors and the transceiver boards. I also learned the practical value of bypass capacitors in reducing noise. My ability to code, especially in MATLAB, also increased, as well as my ability to detect and deal with errors. I learned the value of hard work and the necessity of working on projects early (even though I did start early, and did a lot early, I failed to do *enough* early).

X. Acknowledgments

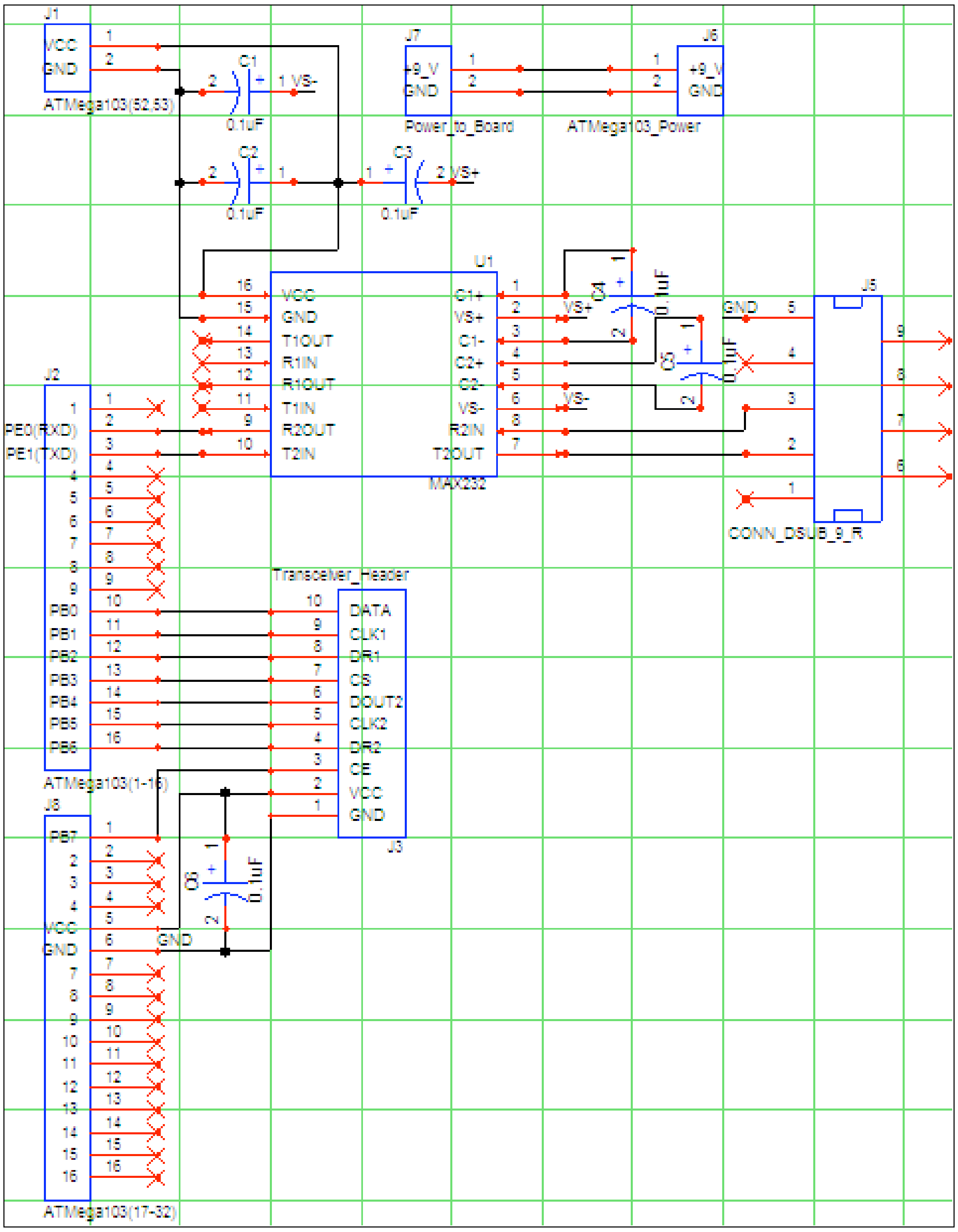
I used Chris Shepherd's final report for “Snackbot” and Paul Dutka's final report for “MILee”, both from IMDL summer 2006 as guidelines for how to write my report. I also used the TJ model by Scott Jannes as a basis for designing the top of my robot platform. I used Karl Dockendorf's circuit diagram from StalkerBot (IMDL Spring 2006) to figure out how to wire up my photoreflectors.

XI. Appendices

A. AutoCAD Diagram of the Lab Rat Platform



B. Circuit Diagram/Schematic of Wireless to Serial Module



C. MATLAB Code

```
function M = sample10c(void)

close all;

minimum_IR=0;
maximum_variance=0;

fig=1;
%character=zeros(1,255);
lf_IR=zeros(1,10);
rf_IR=zeros(1,10);
lr_IR=zeros(1,10);
rr_IR=zeros(1,10);
f_IR=zeros(1,10);
position=zeros(1,10);
velocity=zeros(1,10);
desired_right_servo_value=zeros(1,10);
desired_left_servo_value=zeros(1,10);
x=1:10;
%total_packets_read=0;
%total_packets_used=0;
delay=timer('TimerFcn',' ');%disp("The graph should be displayed by now.")');

%configures com ports--all I needed to configure for my com port was 'StopBits',
%but other things may need to be configured for the actual robot.
%NOTE: IF CPORTCLOSE IS NOT CALLED AFTER THE PREVIOUS CALL, THIS WILL NOT
WORK.
COM2=cportopen('com2');
status1=cportconfig(COM2,'StopBits',1);
status2=cportpurge(COM2);
if(status2 ~= 1)
 output='An error occurred while opening the COM2 port - it is probably already in use'
 cportclose(COM2);
 return
end

output='Waiting for data from serial port.'

input=1;
input_size=1;
while(isempty(input) ~= 1 && input_size ~= 11)
 input=cportgetline(COM2,11,char(10),10);
 %total_packets_read=total_packets_read+1;
 [err input_size]=size(input);
 %for ii = 1:input_size,
 % if(ii ~= 11)
 %  character(input(ii))=character(input(ii))+1;
 % end
 %end
end
if(isempty(input))
 output='No data on the COM2 port for 10 seconds'
 cportclose(COM2);
 return
end
%total_packets_used=total_packets_used+1;

lf_initial=input(1)*4 + input(2)/64;
rf_initial=double(bitand(uint8(input(2)),63))*16 + input(3)/16;
lr_initial=double(bitand(uint8(input(3)),15))*64 + input(4)/4;
rr_initial=double(bitand(uint8(input(4)),3))*256 + input(5)/1;
 f_initial=input(6)*4 + input(7)/64;
position_initial=double(bitand(uint8(input(7)),63))*16 + input(8)/16;
if(position_initial>511)
 position_initial=position_initial-1024;
end
rsv_initial=input(9);
lsv_initial=input(10);

output='Initial values: lf,rf,lr,rr,f,pos,rsv,lsv'
output=[lf_initial,rf_initial,lr_initial,rr_initial,f_initial,position_initial,double(rsv_initial),double(lsv_initial)
]

counter=1;

%endless loop
while(1==1)
 input=1;
 input_size=1;
 while(isempty(input) ~= 1 && input_size ~= 11)
  input=cportgetline(COM2,11,char(10),3);
%  total_packets_read=total_packets_read+1;
  [err input_size]=size(input);
  %for ii = 1:input_size,
  % if(ii ~= 11)
  %  character(input(ii))=character(input(ii))+1;
  % end
  %end
 end
 if(isempty(input))
  output='No data on the COM2 port for 3 seconds'
  cportclose(COM2);
  figure(fig);

plot(x(1:counter),lr_IR(1:counter)/4,'r',x(1:counter),rf_IR(1:counter)/4,'g',x(1:counter),f_IR(1:counter)/4,'b',
x(1:counter),position(1:counter)/8,'k*-',x(1:counter),desired_right_servo_value(1:counter),'c.--
',x(1:counter),desired_left_servo_value(1:counter),'m.--')
  axis([1 counter -100 100]);
  legend('left front IR','left_rear IR','front IR','position','desired right servo value','desired left servo value');
  output=[minimum_IR maximum_variance]
%  plot(1:255,character);
%  output=[character(10),character(13),character(32)]
%  output=[total_packets_read total_packets_used]
  return
 end
% total_packets_used=total_packets_used+1;

 lf_IR(counter)=double(input(1)*4 + input(2)/64)-lf_initial;
 rf_IR(counter)=double(bitand(uint8(input(2)),63))*16 + input(3)/16-rf_initial;
 lr_IR(counter)=double(bitand(uint8(input(3)),15))*64 + input(4)/4-lr_initial;
 rr_IR(counter)=double(bitand(uint8(input(4)),3))*256 + input(5)/1-rr_initial;
  f_IR(counter)=double(input(6)*4 + input(7)/64-f_initial);
 position(counter)=double(bitand(uint8(input(7)),63))*16 + input(8)/16;
 if(velocity(counter)>511)
  velocity(counter)=velocity(counter)-1024;
 end
 position(counter)=position(counter)-position_initial;
 desired_right_servo_value(counter)=double(rsv_initial-input(9));
 desired_left_servo_value(counter)=double(input(10)-lsv_initial);

% output=strcat('left_front_IR: ',int2str(lf_IR(counter)),'  right_front_IR: ',int2str(rf_IR(counter)),'
left_rear_IR: ',int2str(lr_IR(counter)),'  right_rear_IR: ',int2str(rr_IR(counter)),'  front_IR:
',int2str(f_IR(counter)))
% output=strcat('position: ',int2str(position(counter)),'  desired_right_servo_value:
',int2str(desired_right_servo_value(counter)),'  desired_left_servo_value: ',
int2str(desired_left_servo_value(counter)))
 counter=counter+1;

 if(counter>10)
  counter=1;

  status=cportpurge(COM2);
  if(status ~= 1)
   cportclose(COM2);
   return
  end

  figure(fig);
  fig=fig+1;
  plot(x,lf_IR/4,'r',x,lr_IR/4,'g',x,f_IR/4,'b',x,position/4,'k*-',x,desired_right_servo_value,'c.--
',x,desired_left_servo_value,'m.--')
  axis([1 10 -100 100]);
  legend('left front IR','left rear IR','front IR','position','desired right servo value','desired left servo value');

%  ms100=1/24/60/60/10;
%  startat(delay,now+ms100);
%  wait(delay);

  if(minimum_IR > min(lf_IR))
     minimum_IR=min(lf_IR);
  end
  if(minimum_IR > min(rf_IR))
     minimum_IR=min(rf_IR);
  end
  if(maximum_variance < std(lf_IR))
     maximum_variance=std(lf_IR);
  end
  if(maximum_variance < std(rf_IR))
     maximum_variance=std(rf_IR);
  end
  output='left front IR, right front IR, left rear IR, right rear IR, front IR'
  standard_deviation=[std(lf_IR),std(rf_IR),std(lr_IR),std(rr_IR),std(f_IR)]
```

```
mean_value=[round(mean(lf_IR)),round(mean(rf_IR)),round(mean(lr_IR)),round(mean(rr_IR)),round(mea
n(f_IR))]
  minimum=[min(lf_IR),min(rf_IR),min(lr_IR),min(rr_IR),min(f_IR)]
  maximum=[max(lf_IR),max(rf_IR),max(lr_IR),max(rr_IR),max(f_IR)]
 end
end
```

D. ATMega103 AVR C code

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

#define ONE_MS 188 // 188 / (6,000,000 / 32) = 1 ms

#define PHOTOREFLECTOR_PORT PORTE
#define PHOTOREFLECTOR_DDR  DDRE
#define PHOTOREFLECTOR_PIN  PINE
#define PHOTOREFLECTOR_FRONT 7
#define PHOTOREFLECTOR_BACK  6

#define FRONT_IR       7
#define LEFT_FRONT_IR  4
#define RIGHT_FRONT_IR 3
#define LEFT_REAR_IR   2
#define RIGHT_REAR_IR  1

#define TRW_PORT PORTB
#define TRW_PIN  PINB
#define TRW_DDR  DDRB
#define TRW_CE   0
#define TRW_CS   1
#define TRW_DR1  2
#define TRW_CLK1 3
#define TRW_DATA 4

#define SERVO_PORT PORTB
#define SERVO_DDR  DDRB
#define RIGHT_SERVO 5
#define LEFT_SERVO  7
#define RIGHT_SERVO_CENTER 149
#define LEFT_SERVO_CENTER  145
#define FULL_SPEED         60

#define BUMPER_PORT PORTD
#define BUMPER_PIN  PIND
#define BUMPER_DDR  DDRD
#define BUMPER_FRONT  4
#define BUMPER_RIGHT  3
#define BUMPER_MIDDLE 2
#define BUMPER_LEFT   1
```

```
#define BUMPER_PANIC  0
#define RIGHT_BUMPER_PRESSED  SIG_INTERRUPT3
#define MIDDLE_BUMPER_PRESSED SIG_INTERRUPT2
#define LEFT_BUMPER_PRESSED   SIG_INTERRUPT1
#define PANIC                 SIG_INTERRUPT0

#define NONE             0
#define REVERSE          1
#define REVERSE_RIGHT      2
#define REVERSE_LEFT       3
#define REVERSE_LEFT_RIGHT 4  //i.e. an S curve to get away from a wall on the left
#define REVERSE_RIGHT_LEFT 5  //i.e. an S curve to get away from a wall on the right
#define FORWARD          6
#define STOP             7

volatile uint8_t bumper_command;
volatile uint16_t right_servo_value;
volatile uint16_t left_servo_value;
volatile uint16_t desired_right_servo_value;
volatile uint16_t desired_left_servo_value;
volatile uint16_t ms_count;
volatile uint8_t data_to_transmit[10];
volatile int16_t initial[8],IR[8],intersection[8];
volatile uint8_t perceived_end_of_maze=0;
volatile uint8_t turning=0;
volatile int16_t position;

void transmit_data(void);
void configure_transmitter(void);
void delay_ms(uint16_t ms);
void delay_40us(void);
void initialize_IR(void);
void read_IR(uint8_t IR);
void delay(int time);
void arbitrator(void);
void calculate_servo_values(void);
void do_bumper_maneuver(void);
void initialize_bumpers(void);
void initialize_timers(void);
void initialize_servos(void);
int16_t min(int16_t a, int16_t b);
int16_t abs(int16_t a);
void veer_right(int8_t value);
void veer_left(int8_t value);
void test_servos(void);
void turn_left(void);


/*
 *  the arbitrator decides which of the servo commands generated by the sensors to implement
 */
void arbitrator(void)
 {
 uint16_t counter=0;
 //Full Forward
 //BUMPER_RIGHT servo CW, BUMPER_LEFT servo CCW
 desired_right_servo_value = RIGHT_SERVO_CENTER - FULL_SPEED;
 desired_left_servo_value  = LEFT_SERVO_CENTER  + FULL_SPEED;

 while(1==1)
  {
  if(bumper_command == NONE)
   {
   counter++;
   read_IR(FRONT_IR);
/*   if(IR[FRONT_IR] - initial[FRONT_IR] > 256)
    bumper_command=STOP;*/
   read_IR(LEFT_FRONT_IR);
   read_IR(RIGHT_FRONT_IR);
   read_IR(LEFT_REAR_IR);
   read_IR(RIGHT_REAR_IR);

   //if we are at an intersection
   if(intersection[LEFT_REAR_IR]       >       16       &&       intersection[LEFT_FRONT_IR]       >
intersection[LEFT_REAR_IR])
    {
            turn_left();
//          bumper_command=STOP;
            }

   //if the ground appears to be black

if(bit_is_clear(PHOTOREFLECTOR_PIN,PHOTOREFLECTOR_FRONT)&&bit_is_clear(PHOTOREFL
ECTOR_PIN,PHOTOREFLECTOR_BACK))
    {
            if(perceived_end_of_maze>=8)
             bumper_command=STOP;
            perceived_end_of_maze++;
    }
   else if(perceived_end_of_maze > 0)
            perceived_end_of_maze=0;

   //positive value is closer to the right wall
   if(IR[LEFT_FRONT_IR]      -      initial[LEFT_FRONT_IR]      <      IR[RIGHT_FRONT_IR]      -
initial[RIGHT_FRONT_IR])
    position = IR[LEFT_FRONT_IR] - initial[LEFT_FRONT_IR];
   else
    position = -(IR[RIGHT_FRONT_IR] - initial[RIGHT_FRONT_IR]);

   desired_right_servo_value = RIGHT_SERVO_CENTER - FULL_SPEED - position/6;
   desired_left_servo_value = LEFT_SERVO_CENTER + FULL_SPEED - position/6;

   if(counter % 4 == 0)
            {
            data_to_transmit[9] = (uint8_t)                (IR[LEFT_FRONT_IR]  >> 2) ;//first 8 bits of
lf_IR
            data_to_transmit[8]  =  (uint8_t)  ((IR[LEFT_FRONT_IR]   <<  6)  |  (IR[RIGHT_FRONT_IR]  >>
4));//last 2 bits of lf_IR and first 6 bits of rf_IR
            data_to_transmit[7]  =  (uint8_t)  ((IR[RIGHT_FRONT_IR]  <<  4)  |  (IR[LEFT_REAR_IR]    >>
6));//last 4 bits of rf_IR and first 4 bits of lr_IR
          data_to_transmit[6] = (uint8_t) ((IR[LEFT_REAR_IR]     << 2) | (IR[RIGHT_REAR_IR]   >>
8));//last 6 bits of lr_IR and first 2 bits of rr_IR
          data_to_transmit[5] = (uint8_t)  (IR[RIGHT_REAR_IR]  << 0)                    ;//last 8 bits of
rr_IR
          data_to_transmit[4] = (uint8_t)                (IR[FRONT_IR]      >> 2) ;//first 8 bits of f_IR
   data_to_transmit[3] = (uint8_t) ((IR[FRONT_IR]       << 6) | (position           >> 4));//last 2 bits of f_IR
and first 6 bits of velocity
   data_to_transmit[2] = (uint8_t) ((position             << 4))                        ;//last 4 bits of velocity.
photoreflector status will go here, too
   data_to_transmit[1] = (uint8_t) desired_right_servo_value;
   data_to_transmit[0] = (uint8_t) desired_left_servo_value;


          //periodically reconfigure the transmitter (since it sometimes appears to get reset)
   if(counter%1024==0)
           configure_transmitter();

   transmit_data();
          }
  }
 else
  do_bumper_maneuver();
 }
}


void turn_left(void)
 {
 turning=1;
 delay_ms(100);
 //possibly change this to two while loops, one which waits for the maximum value instead of the delay
routine we have now.
 desired_left_servo_value = LEFT_SERVO_CENTER - FULL_SPEED/8;
 desired_right_servo_value = RIGHT_SERVO_CENTER - FULL_SPEED;
 delay_ms(800);
 read_IR(LEFT_REAR_IR);
 read_IR(LEFT_REAR_IR);
 read_IR(LEFT_REAR_IR);
 read_IR(LEFT_REAR_IR);
 while(/*IR[LEFT_FRONT_IR]  >  initial[LEFT_FRONT_IR]  -  160  ||  */IR[LEFT_REAR_IR]  >
initial[LEFT_REAR_IR] - 160)
  {
  read_IR(LEFT_REAR_IR);
  }

 desired_left_servo_value = LEFT_SERVO_CENTER + FULL_SPEED;
 desired_right_servo_value = RIGHT_SERVO_CENTER - FULL_SPEED;
 delay_ms(400);
 read_IR(LEFT_FRONT_IR);
 read_IR(RIGHT_FRONT_IR);
 read_IR(LEFT_FRONT_IR);
 read_IR(RIGHT_FRONT_IR);
 read_IR(LEFT_FRONT_IR);
 read_IR(RIGHT_FRONT_IR);
 read_IR(LEFT_FRONT_IR);
 read_IR(RIGHT_FRONT_IR);

 turning = 0;
 intersection[LEFT_FRONT_IR]=0;
 intersection[LEFT_REAR_IR]=0;
 }


int main(void)
 {
 initialize_timers();

 configure_transmitter();

 initialize_bumpers();

 initialize_servos();

 //makes sure that photoreflectors are set as inputs
 //pulled high to VCC
 cbi(PHOTOREFLECTOR_DDR,PHOTOREFLECTOR_FRONT);
 cbi(PHOTOREFLECTOR_DDR,PHOTOREFLECTOR_BACK);
 sbi(PHOTOREFLECTOR_PORT,PHOTOREFLECTOR_FRONT);
 sbi(PHOTOREFLECTOR_PORT,PHOTOREFLECTOR_BACK);

 sei();

 //test_servos();

/*          data_to_transmit[9] = 0x39;
            data_to_transmit[8] = 0x38;
            data_to_transmit[7] = 0x37;
            data_to_transmit[6] = 0x36;
            data_to_transmit[5] = 0x35;
            data_to_transmit[4] = 0x34;
            data_to_transmit[3] = 0x33;
            data_to_transmit[2] = 0x32;
            data_to_transmit[1] = 0x31;
            data_to_transmit[0] = 0x30;

            while(1==1)
             {
    transmit_data();
             delay_ms(15);
             }*/

 //note that this MUST come after sei()
 initialize_IR();

 arbitrator();
 }


/*
```

```
   This sends out the data stored in the data_to_transmit
   data_to_transmit must be setup before calling this function
 */
void transmit_data(void)
 {
 static uint8_t ii, jj, temp, rf_address;

 sbi(TRW_PORT,TRW_CE);

 //Clock in address
 rf_address = 42;  //the address that I set
 for(jj = 0 ; jj < 8 ; jj++)
  {
  //TRW_DATA = rf_address.7;
  if(bit_is_set(rf_address,7))
   sbi(TRW_PORT,TRW_DATA);
  else
   cbi(TRW_PORT,TRW_DATA);

  //toggle clock
//delay(100);
  sbi(TRW_PORT,TRW_CLK1);
//delay(100);
  cbi(TRW_PORT,TRW_CLK1);
//delay(100);

  rf_address <<= 1;
  }

 //Clock in the data_array
 for(ii = 0 ; ii < 10 ; ii++) //10 bytes
  {
  temp = data_to_transmit[ii];

  for(jj = 0 ; jj < 8 ; jj++) //8 bits each
   {
   //TRW_DATA = temp.7
   if(bit_is_set(temp,7))
    sbi(TRW_PORT,TRW_DATA);
   else
    cbi(TRW_PORT,TRW_DATA);

   //toggle clock
//delay(100);
   sbi(TRW_PORT,TRW_CLK1);
//delay(100);
   cbi(TRW_PORT,TRW_CLK1);
//delay(100);

   temp <<= 1;
   }
  }

 //Start Transmission
 cbi(TRW_PORT,TRW_CE);
```

```
//delay(1000);
 }


void test_servos(void)
 {
/* desired_right_servo_value = RIGHT_SERVO_CENTER+3;
 desired_left_servo_value = LEFT_SERVO_CENTER+FULL_SPEED;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER+2;
 desired_left_servo_value = LEFT_SERVO_CENTER-FULL_SPEED;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER+1;
 desired_left_servo_value = LEFT_SERVO_CENTER+FULL_SPEED;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER;
 desired_left_servo_value = LEFT_SERVO_CENTER-FULL_SPEED;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER-1;
 desired_left_servo_value = LEFT_SERVO_CENTER+FULL_SPEED;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER-2;
 desired_left_servo_value = LEFT_SERVO_CENTER-FULL_SPEED;
 delay_ms(3600);

 desired_right_servo_value=RIGHT_SERVO_CENTER-3;
 desired_left_servo_value=LEFT_SERVO_CENTER+FULL_SPEED;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER+FULL_SPEED;
 desired_left_servo_value = LEFT_SERVO_CENTER+3;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER-FULL_SPEED;
 desired_left_servo_value = LEFT_SERVO_CENTER+2;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER+FULL_SPEED;
 desired_left_servo_value = LEFT_SERVO_CENTER+1;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER-FULL_SPEED;
 desired_left_servo_value = LEFT_SERVO_CENTER;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER+FULL_SPEED;
 desired_left_servo_value = LEFT_SERVO_CENTER-1;
 delay_ms(3600);

 desired_right_servo_value = RIGHT_SERVO_CENTER-FULL_SPEED;
 desired_left_servo_value = LEFT_SERVO_CENTER-2;
 delay_ms(3600);

 desired_right_servo_value=RIGHT_SERVO_CENTER+FULL_SPEED;
 desired_left_servo_value=LEFT_SERVO_CENTER-3;
 delay_ms(3600);


 desired_right_servo_value=RIGHT_SERVO_CENTER-FULL_SPEED;
 desired_left_servo_value=LEFT_SERVO_CENTER+FULL_SPEED;
 while(1==1);*/
 }


void initialize_servos(void)
 {
 // servo port set as output
 sbi(SERVO_DDR,RIGHT_SERVO);
 sbi(SERVO_DDR,LEFT_SERVO);

 right_servo_value = RIGHT_SERVO_CENTER;
 left_servo_value = LEFT_SERVO_CENTER;
 desired_right_servo_value = RIGHT_SERVO_CENTER;
 desired_left_servo_value = LEFT_SERVO_CENTER;
 bumper_command = NONE;
 }


void initialize_timers(void)
 {
 TCCR0  = _BV(CTC0)|_BV(CS01)|_BV(CS00); // clear timer on compare, prescale = 32
 sbi(TIMSK,OCIE0); // Output compare interrupt enabled
 OCR0 = ONE_MS;
 }


void initialize_bumpers(void)
 {
 //enable external low-level triggered interrupts for the bump switches and red panic button
 sbi(EIMSK,BUMPER_RIGHT);
 sbi(EIMSK,BUMPER_MIDDLE);
 sbi(EIMSK,BUMPER_LEFT);
 sbi(EIMSK,BUMPER_PANIC);

 //enable pull-up resistors for bump switches and red panic button
 sbi(BUMPER_PORT,BUMPER_RIGHT);
 sbi(BUMPER_PORT,BUMPER_MIDDLE);
 sbi(BUMPER_PORT,BUMPER_LEFT);
 sbi(BUMPER_PORT,BUMPER_PANIC);

 //turn on front bumpers
 sbi(BUMPER_DDR,BUMPER_FRONT);
 cbi(BUMPER_PORT,BUMPER_FRONT);
 }
```

```
/*
 *  comes to the desired servo value gradually (changes slightly more than 1/8 of the way each ms)
 */
void calculate_servo_values(void)
 {
  uint16_t temp;

  if(right_servo_value<desired_right_servo_value)
   right_servo_value++;
  if(right_servo_value>desired_right_servo_value)
   right_servo_value--;
  if(left_servo_value<desired_left_servo_value)
   left_servo_value++;
  if(left_servo_value>desired_left_servo_value)
   left_servo_value--;

  temp = (right_servo_value + desired_right_servo_value) >> 1;
  temp = (right_servo_value + temp) >> 1;
  right_servo_value = (right_servo_value + temp) >> 1;

  temp = (left_servo_value + desired_left_servo_value) >> 1;
  temp = (left_servo_value + temp) >> 1;
  left_servo_value = (left_servo_value + temp) >> 1;
 }


/*
 *  This function delays for 1 ms / 1500 a short amount of time.
 */
void delay(int time)
{
int n,m;
for(n=0;n<time;n++)
 {
 m=time%2;
 }
}


void initialize_IR(void)
 {
 uint8_t ii;

 //initializes A to D at 6 MHz / 32, interrupt disabled
 ADCSR = _BV(ADEN) | _BV(ADSC) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0);

 //delay to allow IR to power up and place robot after initial servo glitch
 delay_ms(2000);

 //calibrate the IRs from the initial position
 //note that you must start Lab Rat in between two walls with no walls in the first 50 cm in front of it
 //also note that the IRs gradually power on to the actual values
 for(ii=0;ii<128;ii++)
  {
```

```
  read_IR(FRONT_IR);
  read_IR(LEFT_FRONT_IR);
  read_IR(RIGHT_FRONT_IR);
  read_IR(LEFT_REAR_IR);
  read_IR(RIGHT_REAR_IR);
  }
 initial[FRONT_IR]       = IR[FRONT_IR];
 initial[LEFT_FRONT_IR]  = IR[LEFT_FRONT_IR];
 initial[RIGHT_FRONT_IR] = IR[RIGHT_FRONT_IR];
 initial[LEFT_REAR_IR]   = IR[LEFT_REAR_IR];
 initial[RIGHT_REAR_IR]  = IR[RIGHT_REAR_IR];
 }


/*
 *  checks the analog value of the left front IR
 */
void read_IR(uint8_t IR_being_read)
 {
 static int16_t result=0;

 //start AD - it will finish in the interrupt
 ADMUX=IR_being_read;
 sbi(ADCSR,ADSC);

 //waits until conversion is complete
 while(bit_is_clear(ADCSR,ADIF));

 //clear flag
 sbi(ADCSR,ADIF);

 //read in new IR values
 result = ADCW;

 if(result + 200 <= initial[IR_being_read] && turning==0)
  {
  //this is either an outlier or an intersection of walls
  intersection[IR_being_read]++;
  return;
  }
 else if(intersection[IR_being_read]>0)
  intersection[IR_being_read]--;

 //average the values (new value affects the result by 1/16 of the difference)
 result = (result + IR[IR_being_read]) >> 1;
 result = (result + IR[IR_being_read]) >> 1;

 IR[IR_being_read] = result;
 }


/*
 *  millisecond counter interrupt vector (actually not quite 1 ms when the servos are running)
 */
SIGNAL(SIG_OUTPUT_COMPARE0)
```

```
 {
 ms_count++;
 if(ms_count % 16==1)
  {
  OCR0=ONE_MS;//delay to a point somewhat short of the first servo

  calculate_servo_values();

  //turn on the pulse to drive the servos unless the pulse isn't enough to turn the servos at all
  if(right_servo_value < RIGHT_SERVO_CENTER - 3 || right_servo_value > RIGHT_SERVO_CENTER
+ 3)
   sbi(SERVO_PORT,RIGHT_SERVO);
  if( left_servo_value <  LEFT_SERVO_CENTER - 3 ||  left_servo_value >  LEFT_SERVO_CENTER + 3)
   sbi(SERVO_PORT,LEFT_SERVO);
  }
 else if(ms_count % 16==2)
  {
  TCCR0  = _BV(CTC0)|_BV(CS01); // clear timer on compare, prescale = 8
  OCR0 = 250;
  }
 else if(ms_count % 16==3)
  {
  if(right_servo_value<left_servo_value)
   OCR0 = right_servo_value;
  else
   OCR0 = left_servo_value;
  }
 else if(ms_count % 16==4)
  {
  if(right_servo_value<left_servo_value)
   {
   cbi(SERVO_PORT,RIGHT_SERVO);

   OCR0=left_servo_value-right_servo_value;
   if(left_servo_value-right_servo_value < 5)
    cbi(SERVO_PORT,LEFT_SERVO);
   }
  else
   {
   cbi(SERVO_PORT,LEFT_SERVO);

   OCR0=right_servo_value-left_servo_value;
   if(right_servo_value-left_servo_value < 5)
    cbi(SERVO_PORT,RIGHT_SERVO);
   }
  }
 else if(ms_count %16==5)
  {
  cbi(SERVO_PORT,LEFT_SERVO);
  cbi(SERVO_PORT,RIGHT_SERVO);
  TCCR0  = _BV(CTC0)|_BV(CS01)|_BV(CS00); // clear timer on compare, prescale = 32
  OCR0=ONE_MS;
  }
 }
```

```
/*
   2.4G Configuration - Transmitter
   This sets up one RF-24G for shockburst transmission
 */
void configure_transmitter(void)
 {
 static uint8_t ii,jj,temp;
 static uint8_t config_setup[14];

 sbi(TRW_DDR,TRW_CE);
 sbi(TRW_DDR,TRW_CS);
 sbi(TRW_DDR,TRW_CLK1);
 sbi(TRW_DDR,TRW_DATA);

 //Config Mode
 cbi(TRW_PORT,TRW_CE);
 sbi(TRW_PORT,TRW_CS);

 //Note that there must be at least 5us delay from CS to Data
 delay(10);

   //Setup configuration array
   //==================================================================
   //Data bits 111-104 Max data width on channel 1 (excluding CRC and adrs) is 80 (10 bytes)
   config_setup[0] = 80;

   //Data bits 103-64 Channel 2 address - we don't care, set it to 37
   config_setup[1] = 0;
   config_setup[2] = 0;
   config_setup[3] = 0;
   config_setup[4] = 0;
   config_setup[5] = 37;

   //Data bits 63-24 Channel 1 address - set it to 42
   config_setup[6] = 0;
   config_setup[7] = 0;
   config_setup[8] = 0;
   config_setup[9] = 0;
   config_setup[10] = 42;

   //Data bits 23-16 16 bit CRC enabled, Address width 8 bits
   config_setup[11] = 0x23;

   //Data bits 15-8: Maximum ouput power, 16 MHz clock, 250 kbps (the 9600 baud is much slower than
this, anyway), Shockburst Mode, One Channel Receive Modee
   config_setup[12] = 0x4f;

   //Data bits 7-0: Transmitter enabled at 2.442 GHz
   config_setup[13] = 42 << 1;
   //==================================================================

 //Clock in configuration data
 for(ii = 0 ; ii < 14 ; ii++)
  {
```

```
  temp = config_setup[ii];

  for(jj = 0 ; jj < 8 ; jj++)
   {
   if(bit_is_set(temp,7))
    sbi(TRW_PORT,TRW_DATA);
   else
    cbi(TRW_PORT,TRW_DATA);

   sbi(TRW_PORT,TRW_CLK1);
   cbi(TRW_PORT,TRW_CLK1);

   temp <<= 1;
   }
  }

 //Configuration is actived on falling edge of CS (page 10)
 cbi(TRW_PORT,TRW_CE);
 cbi(TRW_PORT,TRW_CS);
 }

/*
 *  Delays the specified number of milliseconds
 */
void delay_ms(uint16_t ms)
 {
 //makes sure that ms_count+number_of_ms_to_delay does not overflow 16 bits
 //The servos use % 16, so changing ms_count to ms_count % 16 does not affect their operation
 ms_count = ms_count % 16;
 ms = ms_count + ms;
 while(ms_count != ms);
 }

/*
 *  enters an endless loop when the red panic button is pressed
 */
SIGNAL(PANIC)
 {
 //This is reached when the red panic button is pressed.
 //Everything needs to stop and wait for a reset or for the red button to be pressed again
 uint16_t ii;

 //delay to debounce switch
 for(ii=0;ii<1000;ii++)
  delay(100);

 //wait until switch is not pressed any more
 while(bit_is_clear(BUMPER_PIN,BUMPER_PANIC));

 //delay to debounce switch
 for(ii=0;ii<1000;ii++)
  delay(100);

 //wait until switch is pressed again
 while(bit_is_set(BUMPER_PIN,BUMPER_PANIC));
 //delay to debounce switch
 for(ii=0;ii<1000;ii++)
  delay(100);

 //returns to operation when the switch is released (since the switch needs to be debounced)
 while(bit_is_clear(BUMPER_PIN,BUMPER_PANIC));

 //delay to debounce switch
 for(ii=0;ii<1000;ii++)
  delay(100);
 }

SIGNAL(RIGHT_BUMPER_PRESSED)
 {
 //disable bumper interrupt until bumper maneuver is completed
 cbi(EIMSK,BUMPER_RIGHT);

 cbi(BUMPER_DDR,BUMPER_FRONT);  //disable low output to front bumpers to check whether the
 sbi(BUMPER_PORT,BUMPER_FRONT); //interrupt was generated from the back (uses a pull-up
resistor)
 delay(3);                                              //delay till front bumpers actually turn off
 if(bit_is_clear(BUMPER_PIN,BUMPER_RIGHT))
  {
  //right bumper is pressed
  bumper_command = REVERSE_RIGHT_LEFT;
  }
 else
  {
  //front right bumper is pressed
  bumper_command = REVERSE_RIGHT;
  }

 //re enable low output to front bumpers
 cbi(BUMPER_PORT,BUMPER_FRONT);
 sbi(BUMPER_DDR,BUMPER_FRONT);
 }

SIGNAL(MIDDLE_BUMPER_PRESSED)
 {
PORTE=11;
 //disable bumper interrupt until bumper maneuver is completed
 cbi(EIMSK,BUMPER_MIDDLE);

 cbi(BUMPER_DDR,BUMPER_FRONT);  //disable low output to front bumpers to check whether the
 sbi(BUMPER_PORT,BUMPER_FRONT); //interrupt was generated from the back (uses a pull-up
resistor)
 delay(3);                                              //delay till front bumpers actually turn off
 if(bit_is_clear(BUMPER_PIN,BUMPER_MIDDLE))
  {
  //back bumper is pressed
  bumper_command = FORWARD;
  }
 else
  {
```

```
  //front bumper is pressed
  bumper_command = REVERSE_RIGHT;
  }

 //re enable low output to front bumpers
 cbi(BUMPER_PORT,BUMPER_FRONT);
 sbi(BUMPER_DDR,BUMPER_FRONT);
 }

SIGNAL(LEFT_BUMPER_PRESSED)
 {
 //disable bumper interrupt until bumper maneuver is completed
 cbi(EIMSK,BUMPER_LEFT);

 cbi(BUMPER_DDR,BUMPER_FRONT);  //disable low output to front bumpers to check whether the
 sbi(BUMPER_PORT,BUMPER_FRONT); //interrupt was generated from the back (uses a pull-up
resistor)
 delay(3);//delay till front bumpers actually turn off
 if(bit_is_clear(BUMPER_PIN,BUMPER_LEFT))
  {
  //left bumper is pressed
  bumper_command = REVERSE_LEFT_RIGHT;
  }
 else
  {
  //front left bumper is pressed
  bumper_command = REVERSE_LEFT;
  }

 //re enable low output to front bumpers
 cbi(BUMPER_PORT,BUMPER_FRONT);
 sbi(BUMPER_DDR,BUMPER_FRONT);
 }

void do_bumper_maneuver(void)
 {
  if(bumper_command == REVERSE)
   {
   //Full Reverse
   //BUMPER_RIGHT servo CCW, BUMPER_LEFT servo CW for 1 s
   desired_right_servo_value = RIGHT_SERVO_CENTER + FULL_SPEED;
   desired_left_servo_value  = LEFT_SERVO_CENTER  - FULL_SPEED;
   delay_ms(1000);
   }
  else if(bumper_command == REVERSE_RIGHT)
   {
   //Reverse with a turn away from the right wall
   //BUMPER_RIGHT servo 1/4 CCW, BUMPER_LEFT servo CW for 1 s
   desired_right_servo_value = RIGHT_SERVO_CENTER + FULL_SPEED/4;
   desired_left_servo_value  = LEFT_SERVO_CENTER  - FULL_SPEED;
   delay_ms(1000);
   }
  else if(bumper_command == REVERSE_LEFT)
   {
   //Reverse with a turn away from the left wall
   //BUMPER_RIGHT servo CCW, BUMPER_LEFT servo 1/4 CW for 1 s
   desired_right_servo_value = RIGHT_SERVO_CENTER + FULL_SPEED;
   desired_left_servo_value  = LEFT_SERVO_CENTER  - FULL_SPEED/4;
   delay_ms(1000);
   }
  else if(bumper_command == REVERSE_LEFT_RIGHT)
   {
   //Reverse S curve away from left wall
   //BUMPER_LEFT then BUMPER_RIGHT for 1 s
   desired_right_servo_value = RIGHT_SERVO_CENTER + FULL_SPEED/4;
   desired_left_servo_value  = LEFT_SERVO_CENTER  - FULL_SPEED;
   delay_ms(500);
   desired_right_servo_value = RIGHT_SERVO_CENTER + FULL_SPEED;
   desired_left_servo_value  = LEFT_SERVO_CENTER  - FULL_SPEED/4;
   delay_ms(750);
   }
  else if(bumper_command == REVERSE_RIGHT_LEFT)
   {
   //Reverse S curve away from right walll
   //BUMPER_RIGHT then BUMPER_LEFT for 1 s
   desired_right_servo_value = RIGHT_SERVO_CENTER + FULL_SPEED;
   desired_left_servo_value  = LEFT_SERVO_CENTER  - FULL_SPEED/4;
   delay_ms(500);
   desired_right_servo_value = RIGHT_SERVO_CENTER + FULL_SPEED/4;
   desired_left_servo_value  = LEFT_SERVO_CENTER  - FULL_SPEED;
   delay_ms(750);
   }
  else if(bumper_command == FORWARD)
   {
   // Full Forward with a small bias to the right to prevent the robot from becoming stuck
   // (i.e. BUMPER_RIGHT servo CW, BUMPER_LEFT servo 3/4 * CCW) for .5 s
   desired_right_servo_value = RIGHT_SERVO_CENTER - FULL_SPEED;
   desired_left_servo_value  = LEFT_SERVO_CENTER  + (FULL_SPEED * 3/4);
   delay_ms(500);
   }
  else if(bumper_command == STOP)
   {
   // it can be used by a function to manually stop
   desired_right_servo_value = RIGHT_SERVO_CENTER;
   desired_left_servo_value  = LEFT_SERVO_CENTER;
   while(1==1);
   }
  //end of bumper maneuver
  bumper_command = NONE;

  //Full Forward
  //BUMPER_RIGHT servo CW, BUMPER_LEFT servo CCW
  desired_right_servo_value = RIGHT_SERVO_CENTER - FULL_SPEED;
  desired_left_servo_value  = LEFT_SERVO_CENTER  + FULL_SPEED;

  //re enable bumper interrupts
  sbi(EIMSK,BUMPER_RIGHT);
  sbi(EIMSK,BUMPER_MIDDLE);
  sbi(EIMSK,BUMPER_LEFT);
  }
```