Synchronized Box Pushing

Mark Milburn
Derick Monroy
Tanner Oakes
Chad Stephenson
Vincent Urias

May 4, 2007

Abstract

In this paper, we will discuss the implementation, design and evaluation of the Junior Design Project (EE382). This paper discusses the process that was developed to create a wireless system, composed of to two MicaZ motes, that capable of sensing a box and bounds of the lane, while transmitting information back to basestation all the while implementing an autonomous control mechanism using NecC and the TinyOS architecture.

Contents

List of Figures

List of Tables

1 Introduction

As our society becomes increasingly more digitized, there has been an increasing emphasis on communication systems and as future engineers, it is a necessity to understand, implement and analyze these systems in order to effectively compete in this global economy. A quickly emerging area in communications is wireless communications. With the advent of wireless communication, we see that it is a goal that many systems are trying to transition off the inherently limited wired world. For instance, in the area of cell phones, the whole idea rests on the fact that people can be mobile without having to worry about lugging around wires with them. Without the hindrance of wires, wireless communications is certain to be the future. With that in mind, research in the field of wireless communications has been increasing steadily. Every day new algorithms and applications to wireless devices are being discovered. One such application is this year's junior design project to test and evaluate the ability to coordinate wireless devices, nodes, to move an object, a box a set distance while staying in to confines of a give track.
The goal of this project concentrates on the extension and evaluation of a wireless system capable of sensing both the box and bounds of the lanes, transmission between the motes and the basestation and autonomous control using complex sensing devices using the wireless sensing nodes referred to here after as motes and the lightweight programming language NecC.
In this paper, we will discuss our team's implementation and design process in creating our robots that utilized the use of a machined tilt bar, optical sensors, a PS2 mouse, and the MicaZ.

2 Project Specifications

The goal of this project are to leverage the existing data processing and robust wireless communication abilities of the MicaZ motes using the existing TinyOs infrastructure to carefully design and evaluate potential designs by using our previous few years of electrical engineering to meet several key design elements. Our project falls within the wireless sensing and control domain. We were given the task to design a pair of robots that were to work autonomously to push a box from a designated start line to the finish line while remaining within the confines of black line bounded lane. The motes had to coordinate and communicate between each other and the basestation. The motes would not know their start position relative to the box, they would not know what side of the bar they would start on, and all the coordination code must reside locally on the motes. Finally, there has to be a GUI that provides a sensible display of all the commands that are sent between the robots and themselves, and between the basestation and the robots. The hard specifications of the project were as follows:

- Width of the lane will be 2 meters long

- Box width would be 1 meter long
- Robot size was a maximum of 6in x 6in x 6in

In order to realize a complete solution to this problem necessitated the following to be done:

- Understand how TinyOS worked and operated
- Design hardware
- Design software to integrate both hardware and create an autonomous control mechanism

3 TinyOS

TinyOS is a framework that enables the user to build programs and algorithms from components that can be linked together to produce a lightweight, robust program that can be uploaded on to a mote. The whole project is based on an open-source development environment, which is based on the programming language and model of NesC. There is a unique programming model that is used in order to produce an ideology of optimized power saving and extremely highly concurrency model which has its roots in the interrupt driven OS. This OS implementation is specifically event-driven who's primary goal is to reduce the costly (both in time and money) services such as complex memory management and allows for advanced interrupt system which condenses many of the performance reducing implementations such as polling and switching. Although there is reduced functionality such as complex math, support and inherent limitations to the expandability of the OS make this a special purpose OS which provides efficient wireless communications system and sensor collection.

4 Problem Description

At face value this problem seems very easy, two robots are placed somewhere in between a lane and then meet a box and push it a set distance. However, several key components that make this a complicated solution including wireless communication, as well sensor integration, complex PID control loops among other things.
The problem can be broken down into a several distinct steps that had to be integrated and working together:

- Box detection
- Moving the box
- Line detection

5 Coordinated Behavior

Some basic coordinate behavior principles were evaluated and implemented in this project. Coordinated behavior is used for a system of individual robots so that the robots can communicate with each other in order to perform a specified task, in our case there are three motes, two agents and one basestation that have the task of locating and moving a box.

5.1 Approaching the Centralized Model

Initially there was thought into creating a centralized model in which the basestation would calculate the speed, direction, etc for each of the motes. The basestation is controlling both of the robots and is designated as the center of communication and information for the entire system. This model would have been good because much of the complex calculation that was initially brainstormed could be done by the laptop, which would reduce the computational intensity on each of the local boxes. However, after some careful evaluation we realized that a decentralized model would be more appropriate because of several design issues. We thought it best that individuals agents in the system should be treated as autonomous as possible and should not be directly controlled by one agent. The first consideration was based on the fact that if we followed a centralized approach it would force a computationally more intensive solution to be developed because the system would have to handle the communication between each of the motes, command oversight, and information management so that the robots can operate optimally. It would also force a system that would have to propagate long-range communications through the network and can use relatively low power to communicate with the immediate neighbors.

5.2 Deciding on the Decentralized Model

Some of the net benefits that we realize out weighed the computational intensity reduction. In the alternate model, a decentralized approach, allows the each distinct robot to make local decisions and computations without the direct guidance of an individual controlling basestation. One huge benefit is there is a significant reduction in communication delays because it removes the clock cycles and communication lags that was required by the MicaZ to receive the communication and decide what its next course of action is. [5] Additionally it allows for a completely autonomous configuration that allows each mote to continue its course of action even if the other one were to fail which could ultimately jeopardize the entire task. The autonomous configuration makes the individuals responsible for only themselves.

6 Applying PI Loops

The speed and direction of the robot is controlled using a PI control loop. Feedback data is collected from the mouse and the tiltbar, and error values are

calculated from that data. The PI loop returns a PWM modifier value. The desired speed of the robot is set as a certain number of y counts over a set time interval, and the error is the difference between the measured y counts and the desired y counts. This error is fed into both proportional and integral control equations for the left and right wheel. The heading of the robot is set at the desired value relative to where the robot started facing; the current heading is maintained by adding the x count to current heading on each mouse read. The error is the difference between the desired heading and the current heading, and this error is run through a proportional control equation for each wheel. The robot always tries to keep the tiltbar from turning, when the robot starts up it defines the first tiltbar reading as the neutral position. When the tiltbar turns it takes the turn value and puts it into a proportional control equation, the desired speed is modified using the resulting value to slow down or speed up the robot depending on what side it is on. By tuning the control equations correctly, the robot can drive in a straight line while keeping the box from turning. The equations were tuned by adjusting the gain for each equation one at a time until the desired output was achieved.

7 Project Solution Narrative

In order to fulfill the project specifications above we created a solution based on careful evaluation of sensors and inter-mote interactions. In this situation, we see three key scenarios that dictate the operation of the motes: getting contact with the box, moving the box, and ensuring that the motes stay in the lane.

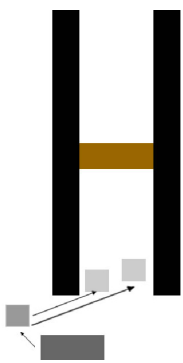

Figure 1: Initializing the motes

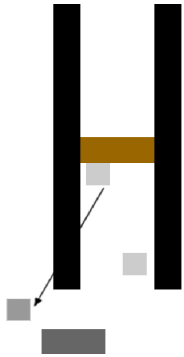

Figure 2: First mote hits

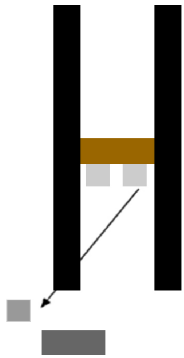

Figure 3: Second mote hits

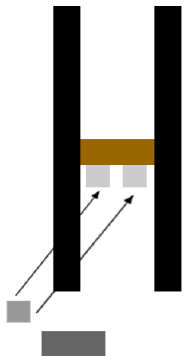

Figure 4: Basestation signals both to resume

7.1 Getting to the Box

The most fundamental portion of this project stemmed from the need for the motes to be able to detect when it had encountered the box. We needed a mechanism that was able to detect when the bar were exposed to the mote and report back to the basestation that it did. We used the wireless communication capabilities of the MicaZ to facilitate this. The first part of the process was built around addressing the project specifications we were unable to determine how far away or on what side of the track the motes would be placed on. First, we decided that the motes would receive their direction from a wireless signal initiated from the GUI. In order to ensure that the both were operational and would start at the same time we implemented a three way and shake that would prevent only one mote going toward the box and not having other one fail as illustrated in Figure 5. After passing the handshake, they would both approach

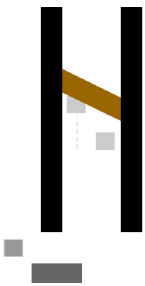

Figure 5: Moving the Box

the box. Once one mote encountered the box, it would signal the basestation, which will keep track of both of them and wait for the second mote to catch up with it. How this occurs is visualized in Figures 6 and 7. Once both robots hit the box the basestation, again send a go message that both motes will respond to as seen in Figure 7.

Figure 6: Continue Moving the Box

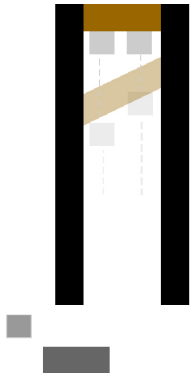

Figure 7: Basestation signals both to resume

7.2 Moving the Box

Once both motes were moving the box it was the responsibility of each individual mote to stay in contact with the box. In order to ensure this we created a tiltbar that would provide a mathematically trivial way to detect the overall movement of the mote. Based on the voltage difference of the potentiometer it would also be used to detect and correct the directional movement of the MicaZ. For example, we are not going to know which side either of the Micaz's will be on so we will just assume that one of the Micaz's will be on some side initially. After one mote hits the box, based on the feedback we can send back a message to the basestation, which would keep track which side they are on.
The direction can be determined to be left or right side and once it meets the box as it is move the tiltbar's voltage, either positive, right, or negative, left, would dictate what course of action to take. Based on the control loop we will be able to in a relatively smooth way allow speeding up and slowing down of each mote. For example, if after the robots were to make contact with the box and the left mote were to speed up off the line, then the right mote would increase its speed until it noticed that it made contact with the box again and so on and so forth. The two motes can push the boxes at a paced rate until the stop condition was sent by the basestation, which would be determined by the user designating that the end of the track was met.
This part of the solution clearly show how this implementation utilizes the decentralized coordinate behavior model between the two motes which removed the complex system of latency ridden wireless communication between the two motes to coordinate at a not so real time rate. This was especially important because TinyOS was not meant to account for extremely high speeds needed for the near real time reaction times we were seeking. The tiltbar is at the crux of the decentralized model because the control loop dictates how fast or slow the mote will be traveling based on the feedback that it receives independent of the second mote or even the basestation's input.

7.3 Black Line Detection

Here we evaluated several scenarios of worst-case situations that would prevent the motes from reaching its goal. One major obstacle stemmed from the risk that one or both motes crossing the black line, thus dictating that it was outside of the bounds of the track. What would happen, if one mote crossed the line would be an event would be triggered to stop the other mote until the first mote came back into contact with the box. Here we would turn both motes at the same rate back into the track, straighten out the motes and then continue heading down the track.

8 Implementation

When implementing both the hardware and software for this project several key design points were used as a guide for the majority of our decisions. Our goal was to create a low budget, innovative, self contained modular design that would showcase not only our skills as engineers but also our ability to create out of the box ideas and make them a reality.
We tried to compartmentalize our design into several key components. First, the collision detection whose purpose was to detect when a mote has encountered the box that lead us to creating the tilt bar. Second was the speed control, whose purpose was to detect how fast the MicaZ in order to provide a mechanism for coordinate behavior, which leads us to cannibalizing a P/S2 mouse. Third, the optical control whose purpose was needed to detect if the MicaZ had gone outside the bounds of the black tape, which lead us to optical sensors. Fourth, the chassis design to support each component mentioned above.

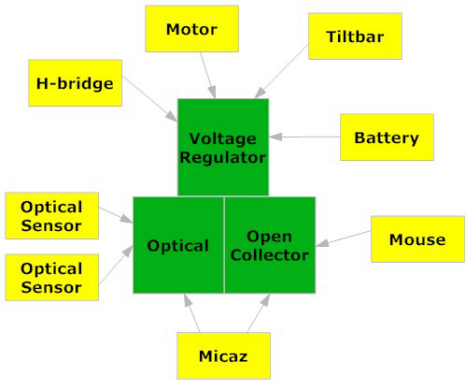

Figure 8: Component Overview

Finally, a wireless communication system was needed to support communication between the motes, GUI and basestation. What this modularization provided was an inherent model to not only integrate and test hardware but also software

and if by chance a system worked, it allowed the ability to swap out the system with limited impact on the entire design.

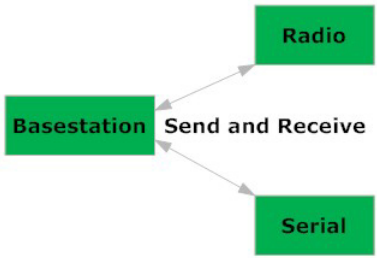

Figure 9: Basestation Component Architecture

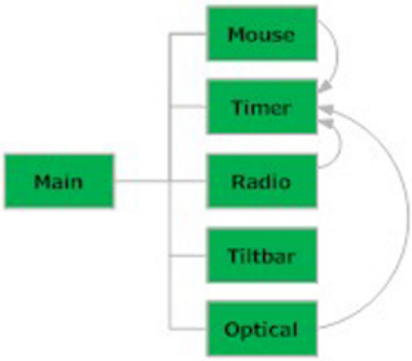

Figure 10: Mote Components

8.1 Optical Sensors

The initial design used photoresistors in combination with LEDs to detect boundary lines. Boundary lines present a unique problem in this project. Not only do the robots have to stay within some defined boundary, the box must also be kept within these lines. Without an effective boundary detection system, successful guidance of the box cannot be guaranteed. The group had several concerns about implementing this design. The sensitivity to different lighting environments, complexity of the input, time required for construction and the need for constant recalibration drove us to evaluate other design possibilities. Fairchild Semiconductor's QRB1134 infrared optical detectors fulfilled our need for a quick and easy-to-implement boundary detection system. The QRB1134 provides a simple digital output, has an easy to mount monolithic package, requires little supporting circuitry, and is inexpensive in comparison to other detection systems. Originally, the left and right photo output were fed into an

OR gate which was used to trigger the INT2 on the MicaZ. By feeding the left and right outputs into GPIO pins RD and WR on the MicaZ, we were able to recognize which input was fired. This became a problem because the MicaZ is a high impedance input device. We solved this problem by feeding the individual outputs of the photo sensors into OR gates. This provided our high impedance output to drive the GPIO pins. The original software design utilized an inter-

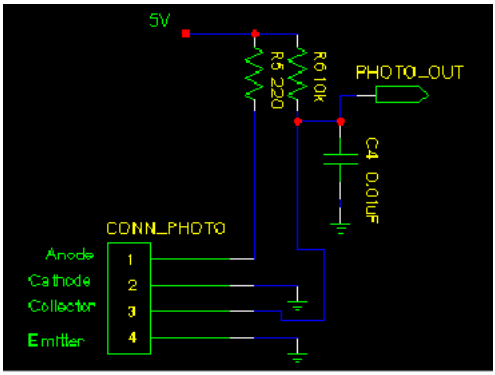

Figure 11: Optical Sensor Circuit

rupt service routine that triggered on the photo inputs. While this design was able to trigger on optical events, it proved to be troublesome. Triggering of the INT2 line on the MicaZ would often cause the unit to reset. We originally thought that this was caused by the optical output feeding 5V DC into the MicaZ that was operating at 3V DC. Implementing a voltage divider to provide a 3V output failed to solve the problem. The interrupt method was then abandoned in favor of using the TinyOS timer system to poll the state of the optical sensors. Interrupts would be ideal in a microcontroller scenario because processor time is spent checking and processing optical input only when input has been received. In contrast, polling simply checks in the input levels at some specified interval. If input is not present, the polling routine simply becomes wasted processor cycles.
Both polling and interrupts were tested by sweeping the sensors over a black line. The LEDs were used to indicate when the MicaZ had entered into the polling or interrupt routine, as well as which sensor was triggered. We found that reset events could be detected by using a boot-up routine that toggled the LEDs in a specific pattern before initializing interrupt or polling routines.

8.2 Directional Sensor

One sensor that the robots needed was a touch sensor. Instead of buying two touch sensors to put on the front (one on the left and one on the right) of the robot we decided to make our own. Having two sensors would let us know whether the robot lost contact on either side; letting us know that the robot was

no longer going in a straight line. For our one-touch sensors, we attached an aluminum bar to a 10Kohm potentiometer. We will take an initial reading of the voltage across the potentiometer and from this; we will be able to tell when the robot has engaged the box by the change in voltage across the potentiometer. In addition, if the robots stop going in a straight line the bar will rotate with the box, letting us know which way the robots are turning. We attached a spring to the left and right of the tiltbar to restore it to its equilibrium state when the bar rotates after engaging or when the robots are turning. We chose to make our own sensors to save on money and thought that it would be better than having digital touch sensors, which would only let us know whether it was touching, or not. From looking at the reading, we should be able to decide how far from straight the robot is in relation to the box.

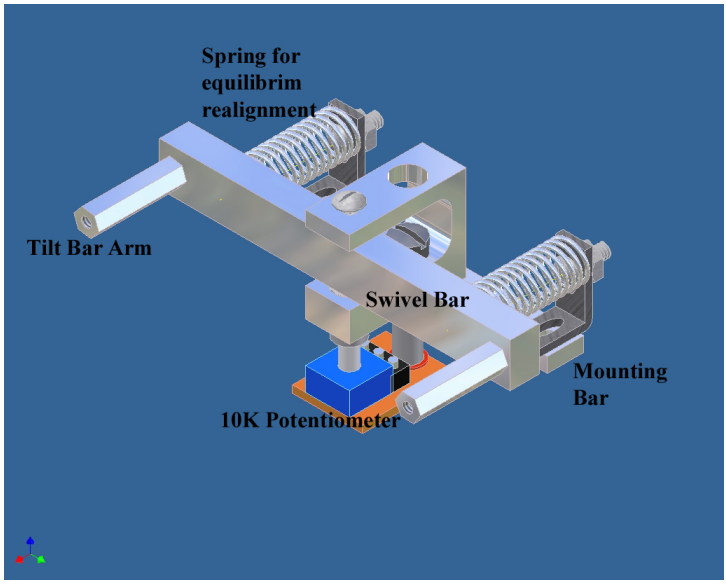

Figure 12: Tilt Bar

The tilt bar controls the entire system by letting us know whether the robot is veering in either the left or right direction. We correct the direction by reading the voltage coming off the tilt bar and slowing down the robot that is traveling faster than the other. This allows the other robot to catch up and align the tilt bar for both robots. Before gluing the potentiometer to the tilt bar, we calibrated the potentiometer to be approximately 5K Ohms, which is the middle value of the potentiometer. We thought about using ultrasonic sensors to detect where the box was but decided not to use them because of the complexity involved with setting up the timers to run them. We would need to have another timer to control the ultrasonic sensors, and that was unavailable.

8.2.1 Analysis of the Tilt Bar

The tiltbar is one aspect of the robot, which has a lot of influence on the overall control of the robot. The tiltbar connects to the MicaZ voltage output to make the following equations work. ADC_v is the analog to digital converter, which is what the MicaZ actually uses. V_{in} is the voltage going into the AD converter and V_{ref} is the reference voltage. V_{batt} is the battery voltage.

$$ADC_v = \frac{1024 \cdot V_{in}}{V_{ref}}$$

$$V_{in} = \frac{ADC_v \cdot V_{ref}}{1024}$$

$$V_{batt} = \frac{1024 \cdot V_{bg}}{ADC_{batt}}$$

By setting $V_{ref} = V_{batt}$ we obtain

$$V_{in} = \frac{ADC_v \cdot V_{batt}}{1024}$$

$$V_{in} = \frac{ADC_v \cdot V_{bg}}{ADC_{batt}}$$

V_{in} will change as the battery dies, however the ratio of V_{batt} to V_{in} will not change as the battery dies.

$$\frac{V_{batt}}{V_{in}} = \frac{1024}{ADC_v}$$

To make the resolution better than 0 to 1 we set our scale to be 0 to 1000 and we end up with

$$\frac{V_{batt}}{V_{in}} = \frac{1024000}{ADC_v}$$

8.3 Angle Calculations

The specifications given to us included that the robot would be placed behind the box at a distance no greater than two feet. While being placed here the robot will be pointed straight at the box, but this is not an exact measurement. We needed to know how much of an angle we could be off by. Since our robot is 3.5 inches wide in the front, we used simple trigonometry to see how great of an angle would cause us to drive straight and completely miss the box. Taking the inverse tangent of 3.5 divided by 24 gives us a maximum angle of 8.30 degrees that we could be off. Since this measurement is taken saying the robot will be aligned with the edge of the box, we determined that this would not be a problem.

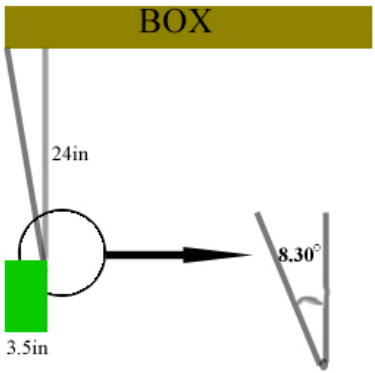

Figure 13: Angle of Error

8.4 Speed Sensor

The robots collect data on speed and direction by means of a PS/2 balled mouse. Mice have two optical encoders built in and provide the circuitry for counting the encoder pulses and transmitting them to a host (the MicaZ in this case). The y-axis optical encoder is used to collect forward motion, the mouse is polled at a constant time interval and the amount of forward movement is proportional to the forward speed of the robot. The x-axis optical encoder acts as a direction measurement, as the robot turns the x-axis changes relative to the direction of the robot. It follows an arc on a circle and is proportional to an angle offset from a reference point. Mice are designed to operate in either polling mode or interrupt mode, and the resolution and sample rate of the mouse can be changed as well. Mice start in polling mode with a resolution of four counts per mm with a sample rate of 100 samples per second. [2]

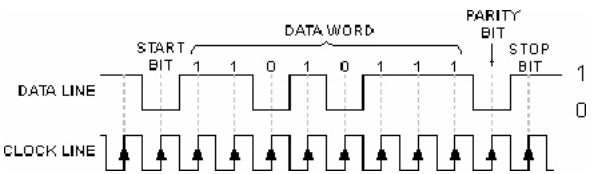

Figure 14: Mouse timing diagram

Communication with the mouse is done through the PS/2 protocol. There is a clock line and a data line between the mouse and the MicaZ, and an open collector circuit is used on each line for the MicaZ to read and set these lines.

To request data from the mouse the MicaZ pull the clock line low for about 100μs, then pulls the data line low and releases the clock line. The mouse then starts to generate a clock signal at 10 to 16.7 kHz and the MicaZ transmits the data request instruction on the falling edge of the clock. The mouse sends back an ACK on the data line to confirm that the message was received. Once a data request has been sent the mouse starts the clock again and sends back three bytes that are read on the rising edge of the clock by the MicaZ. [3] The first byte is the mouse status bytes, which includes x and y overflow, x and y negative, and mouse click information. The next two bytes are the number of x and y counts from the last read. [2] In order to keep the robot within the six-inch length requirement the front portion of the mouse was removed. This caused all the bits in the status byte to be shifted to the left by one, and caused the y-overflow bit to be shifted off so the speed has to be set such that the y count does not overflow as there is no way to check if it has now. Code for the mouse

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							

Figure 15: Structure of a mouse movement data packet

was adapted from Mouse Painter Pro, a graduate project by James Yu and Jared Clifton at Cornell University. Mouse Painter Pro is a microcontroller controlled TV paint program where the mouse is plugged into a microcontroller. [1] The code was modified to work on the MicaZ and to operate in polling mode instead of interrupt mode, this involved rewriting most of the sending code and change the signaling to the pins used on the MicaZ.

8.4.1 Testing PI

The control loop values were obtained through a systematic approach of adjusting the gains until the desired response is obtained. [4] First, all gains were set to their lowest values. The P gain was then increased until the robot started overreacting to the feedback data, then the P gain was lowered until the robot showed the desired response. Next the I gain was raised until the steady state error in the robot's speed went to zero. Because this is a discrete system, the minimum reaction of the robot can be too much small errors, so an error band was put into place to have the robot ignore errors this small. A similar approach was taken to find the correct adjust value for the x-axis bang-bang control. The adjust value was raised until the robot started displaying a very large reaction, and then the adjust was lowered until the robot's response was correct. The tiltbar feed back was also adjusted in this way.

8.5 Chassis Design

We originally considered a basic rectangular shaped design for the chassis. The idea was to have the robot's chassis be mounted only between the wheels of the motor. The dimension would consist of a width of the distance between the wheels and a length that would extend to the limiting dimension requirements, 6 inches. We selected aluminum as the material to construct the chassis because of its ability to withstand the loads that would be applied, the low density, durability and ease of machining.
Once the addition of a front-end mounted tiltbar sensor was finalized, the length of the chassis was shortened in the design to meet the allotted dimensions for the project.
The early design began by using Autodesk Inventor 11 (a 3-D modeling program) to produce a realistic model of the components that the robot we had in mind would resemble. The program utilizes parametric modeling that is composed of assemblies and components to build the 3-D model. Once we completed a design idea, we modeled it to predict and correct immediate issues within the assembly, such as alignment and clearance. Upon meeting these issues, the parts in need of machine work were transferred to a drawing via the CAD program. The hardware group and other machinists used these drawings to construct various prototype and final parts.
The components of the robot were designed with care. The capability of and relative ease of being machined were taken into account in order to alleviate unnecessary difficulties during the process.

8.5.1 Prototype 1

The first prototype chassis was produced in the R&ED instrument room using a mill machine. The shape was a solid rectangle and was six inches long, as the front sensor had not yet been confirmed. Four .0150 holes were drilled into the piece of aluminum to mount the motor and the h-bridge. The positions of the holes were placed in such a manner that the motor's wheel horizontal radius is located as close as possible to the end of the chassis and the h-bridge could utilize the same holes that the motor uses to mount. In addition to these four holes, two larger holes were placed towards the front of the square chassis to allow the wires to be run through the chassis. This chassis was good for initial operational testing of the MicaZ and the h-bridge.

8.5.2 Prototype 2

The second prototype was also produced in the R&ED instrument room using a mill machine. The shape was derived from the first prototype with the same location of the .150 inch thick mounting holes. The design of the robot became clearer after completion of the prototype tiltbar sensor. This caused the chassis to be shortened to a length of 4.5 inches in order to have enough room for the arms of the sensor to swivel. Instead of placing two larger holes in the chassis design, we removed as much material as possible. There was no need

for the material in the middle of the chassis; no mounting holes were needed in the center. We removed additional material from the sides of the chassis as well. The design only placed material where it was necessary and consequently became more frame-like. The design saves a few ounces in weight, as the robot had become heavy with the addition of machined sensors, and we wanted to free up energy for the racing aspect of the competition. We did not make any holes to mount the mouse or optical sensors, as the idea of mounting the mouse had yet to be addressed. Ultimately, the mouse was mounted by clamping and was capable of sliding along the chassis to the desired position and then fastened. The thickness of chassis was not considered important in this prototype.

8.5.3 Final Design

Two mechanical engineering students used a mill machine in the Mechanical Engineering Machine Shop to machine the final design. This was the best location because of the superior tools and resources available, and because the student machinists possessed a tremendous amount of more experience in using machine tools. The design is very similar to the second prototype. We included additional material and holes to accommodate the mounting of the mouse and optical sensors in the optimal position that was experimentally determined. The desired thickness was used in the final design. Altogether, the final model offers a compact design for the robot.

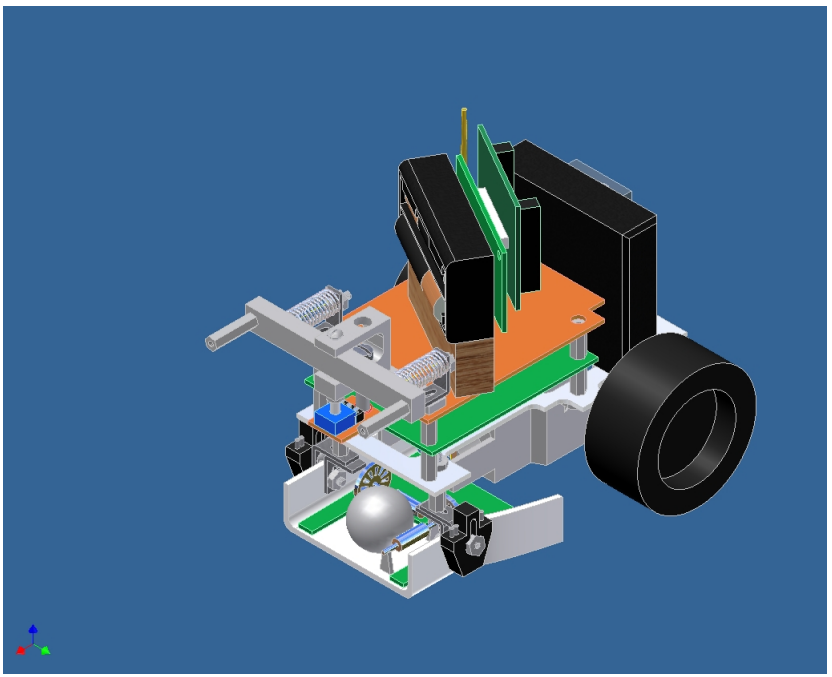

Figure 16: 3D Model of Robot

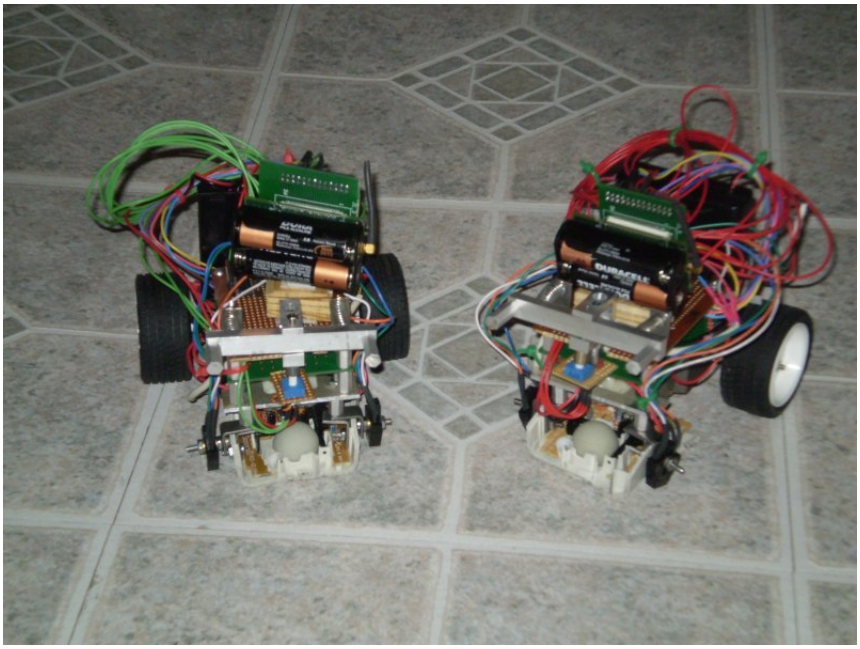

Figure 17: The two robots

8.6 Wirelss/Serial Communication

The entire wireless communication is based on a send / receive model that was implemented in the GenericComm interface that is provided by TinyOS. The only actions that the wireless uses is the broadcasts of serial data, the 'go' command from GUI and is received by the motes and then they respond to saying they got the message. The second message is based on the data that each mote is collecting, which is sent back to the GUI for processing of the data. Every time the basestation received the data, it would send it via the serial interface (UART) back to the laptop and where it would be processed to the screen.

9 Analysis

After completing construction of the robots we had to test each individual subsystem to make sure that it worked and then integrate all the subsystems to determine whether they worked together properly. The mouse was the only sensor that didn't give us that much trouble. The only difficulty with the two mice is that after being cut down to size and mounted they were cut and mounted slightly differently which in turn made the two robots drive slightly askew. This was easily corrected by adding an offset in the code. After doing some research

the code was implemented very quickly. The tiltbar was very difficult to construct and even after being constructed we broke it several times. The first two times we broke the potentiometer and the third time we broke the circuit board which the potentiometer attached to. The optical sensors we bought were easy to use but inserting the portion of code that they contributed turned out to be a much more difficult task than originally anticipated. Overall hardware construction took a couple of more weeks than we would liked. Also, the radio communication set us back two more weeks than we originally anticipated.

9.1 Limitations of TinyOS

TinyOS has some limitations with concurrent radio and serial I/O. In NesC their is a concept called fan-out. This is where there are multiple functions that have the same name, and are called at the same time, StdControl.start() is an example of one of these functions. A special function called a combiner is needed to combine the outputs of all the function calls together. Without the combiner the results of all but one of the functions are lost. The same issue exists when mixing radio and serial packets if the packet type is the same. NesC generates two functions with the same name and a combiner has to be provided by the user. If the combiner is not there then only one of the calls succeed. This is not a problem if a program only calls either the radio function or the serial function but not both in a calling function. To work around this different packet types were defined for radio and serial packets. Another issue that was found is that TinyOS can only support two concurrent I/O calls. This was found later in the development process and the only current work around is not to call more than two I/O functions at a time.

10 Conclusion

This project has given us valuable engineering experience. While the goals encouraged us to extend and utilize our collective knowledge of electrical engineering, it also challenged us to coordinate our goals and aspirations as a group. The stated design problem was to create a system whereby two autonomous robots could move a box within a specified bound from one end of a lane to the other. The real design problem was to learn to work together to realize this goal.

11 Appendix A: Budget

Part Ordered	*Cost*	*Quantity*	*Total Cost*
Phototransistor	0.50	1	0.50
Switching Regulator	2.25	1	2.25
Perf board(A)	1.00	1	1.00
Switch	0.50	2	1.00
Linear Regulator	0.75	3	2.25
BJT NPN	0.10	2	0.20
Perf board(B)	1.50	2	3.00
Female-Female wire kit	2.00	2	4.00
Female crimp pin	0.15	25	3.75
Batteries(9V)	6.25	4	25.00
Battery Case	1.25	1	1.25
Optical Sensors	4.39	5	21.95
		Total Cost	**66.15**

Table 1: Budget

12 Appendix B: Power Budget

Item	Voltage	Current	Capacity mAh	Power	Est Time	N
4xAA NiMH	-4.8V	1800mA	1800	-8.64W	1 hour	Moto
2xFA-130	4.8V	2x900mA	N/A	8.64W		M
9V NiMH	-9V	216mA	250	-1.95W	1hour 9min	Senso
LM317	9V	216mA	N/A	864mW		Linear
2xQRB1134	5V	2x40mA	N/A	400mW		Optica
Mouse	5V	100mA	N/A	500mW		
H-Bridge	5V	36mA	N/A	180mW		
2xAA Alkaline	-3V	24mA	2000	-72mW	3.5days	MicaZ
Micaz	3V	21mA	N/A	63mW		Estimated
Tiltbar	3V	3mA	N/A	9mW		Assumes

Table 2: Power Budget for Robot

Micaz	Duty	Operation	Current
Processor			
	50%	full load	8mA
	50%	sleep	8μ A
Radio			
	60%	Tx	12mA
	30%	Rx	8mA
	10%	Sleep	2μA
Logger			
	100%	sleep	2μA
Sensor			
	100%	full load	5mA

Table 3: Power Budget for Micaz

13 Appendix C: Robot Pictures

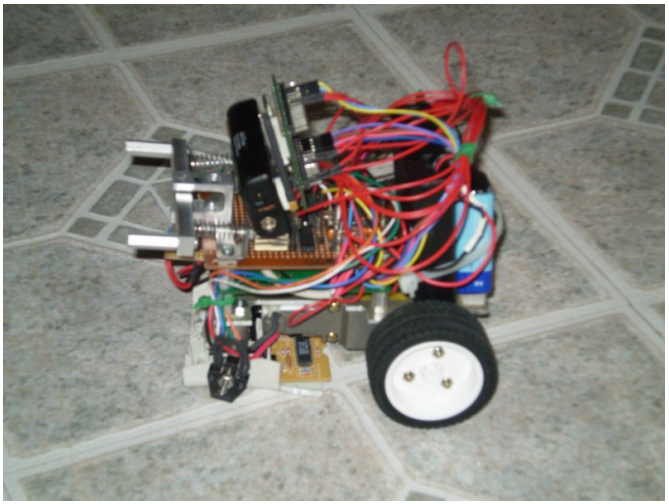

Figure 18: Side View

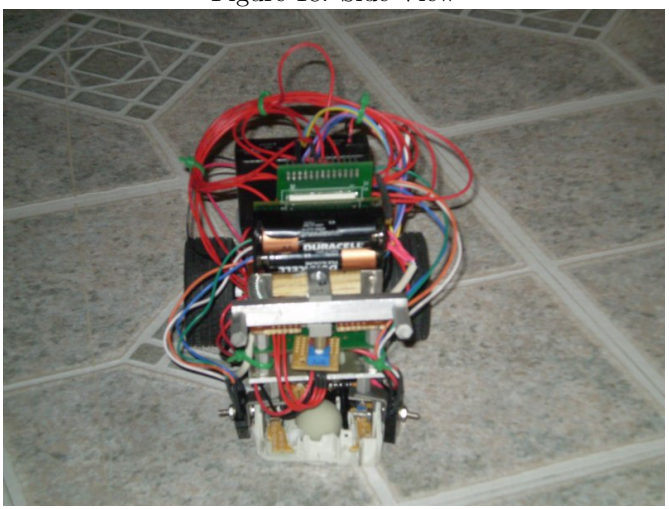

Figure 19: Front View

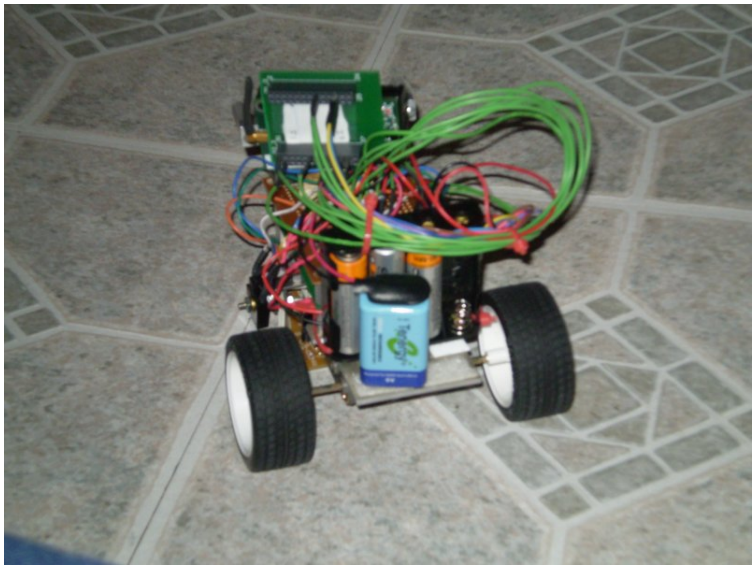

Figure 20: Back View

14 Appendix D: Design Pictures

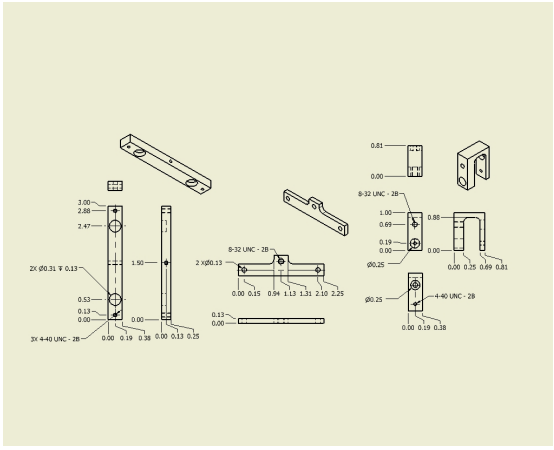

Figure 21: Tilt Bar

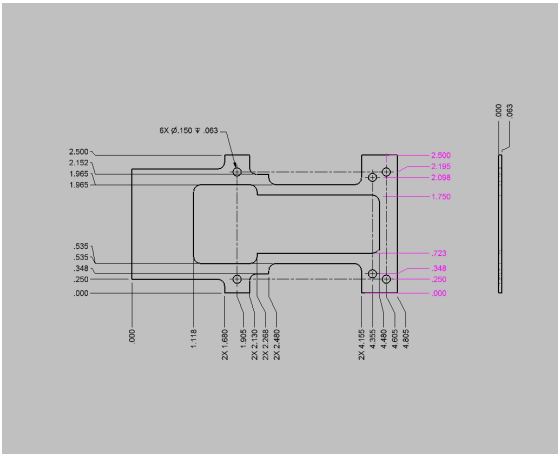

Figure 22: Chassis

References

[1] http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2004/jcc72/code.html.

[2] http://www.computer engineering.org/ps2mouse.

[3] http://www.computer engineering.org/ps2protocol.

[4] http://www.embedded.com/2000/0010/0010feat3.htm.

[5] Dr. H. Van Dyke Parunak, LCDR Michael Purcell, and Robert O'Connell, *Digital Pheromones for Autonomous Coordination of Swarming UAV's*, American Instiute of Aeronautics and Astronautics **3446** (2002).