Final Project:
Wirelessly Controled Boe-Bot With MEMS Gyroscopic Sensing

Andres Fausto
Stewart Bewley
Philip Dawsey
Ki Baek Eom

ME 224 – Experimental Engineering
Professor Horacio Espinosa
Fall 2005

Northwestern University
Robert R. McCormick School of Engineering & Applied Science
Table of Contents

I. Background p. 3
II. Objective p. 3
III. Assembly p. 3
IV. Gyroscope Calibration p. 4
V. Servo Motors Calibration/Centering p. 5
VI. ADC p. 5
VII. Programming p. 7
    • LabVIEW
    • BASIC Stamp
VIII. Wireless Communication (Bluetooth) p. 8
IX. Feedback Control p. 9
X. Final Product – Results p. 10
XI. Future Work p. 10
XII. Conclusion p. 10
XIII. References p. 11

Appendices

• Appendix 1 p. 12
• Appendix 2 p. 14
• Appendix 3 p. 15
• Appendix 4 p. 17
• Appendix 5 p. 18
• Appendix 6 p. 23
• Appendix 7 p. 26
**Background:**

During this quarter of ME 224 – Experimental Engineering we learned through a number of labs about equipment and data acquisition methods. The equipment used in the labs includes multimeters, oscilloscopes, function generators, power supplies, breadboards, op-amps, transistors, resistors, capacitors, inductors, LEDs, LabVIEW software, and the DAQ card. More important is the application of all this equipment to create the data acquisition methods and tests to understand how electrical systems work and their relation to the mechanical domain. The data acquisition methods include the use of the DAQ card (ADC and DAC), LabVIEW acquisition (storing and displaying data). With this in mind, our group’s goal was to control a boe-bot wirelessly. The steps, problems, and results are described in the following report.

**Objective:**

The objective of the final project for ME 224 was to improve last year’s boe-bot robot design by implementing wireless communication. The boe-bot should follow any of four predetermined paths obeying wireless commands given to it from a computer. Analog signals from the gyroscope will be converted to digital signals by the Analog to Digital Converter (ADC) and will be sent back to the base computer through Bluetooth to be analyzed by LabVIEW based on the gyroscope calibration performed. After the error correction is calculated the signal will again be sent wirelessly through Bluetooth and the robot should correct its path. This feedback is done continuously.

**Assembly:**

Though the boe-bot came assembled, our group took it apart in order to understand the connections and the interaction between the ADC and the mechanical components. The following is a list of the most important elements in the boe-bot robot:

- Boe-Bot Robot Kit (Parallax #28132)
- MEMS Vibrating Mass Gyroscope (Analog Devices ADXRS150)
- Analog-To-Digital Converter or ADC (ADC0804LCN)
- 2 Servo Motors (Parallax Continuous Rotation Servo)
- BASIC Stamp, LabVIEW, and Bluetooth-enabled computer
**Gyroscope Calibration:**

To calibrate the gyroscope we ran the self-test and used LabVIEW to acquire data needed to understand the output voltages of the gyroscope in relation to the angular velocities.

Using the BASIC Stamp and LabVIEW programs shown in Appendix 1 we were able to show voltage outputs from the gyroscope at different angular velocities. Graph 1 below shows this relationship. The robot was programmed to spin counterclockwise for 250 pulses, pause 0.5 seconds, and then spin clockwise for 250 pulses. Note that when the robot pauses in the middle and at the end, the voltage is at nearly 2.5 V, which is the ideal value for no motion. We had to calculate voltage from D values output to Matlab by the DAQ using the formula: \( V = \frac{20D}{8192} \).

Graph 1 – Voltage vs. Time

We measured the voltage at different values of angular velocity \( \omega \) using an oscilloscope and LabVIEW, shown in Appendix 1. The oscilloscope had to be positioned so that the values did not go off the end; otherwise it would output erroneous values. We
had a problem with this the first time we ran tests. The data and graph obtained is in Appendix 2. We obtained the time data using a stopwatch and timing several rotations. To get the angular velocity we used:

\[ \omega = \frac{v}{r} = \frac{\pi d}{rt} \]

Where \( v \) is velocity, \( r \) is radius of spin, \( d \) is diameter of spin, and \( t \) is time for one rotation.

We used the trend line from the data to correlate an incoming voltage with an angular velocity. Integrating this value, we were able to determine the angle traveled by the bot. The data we were receiving had to go through an 8-bit ADC to be read by the microprocessor, so it was transformed from a voltage value between 0 and 5 volts to a digital number between 0 and 255 in order to increase the resolution of the output. This will be further explained in the ADC section.

**Servo Motor Calibration/Centering:**

To center the servo motors we used a BASIC Stamp input of 750 indicating no motion and used a screwdriver to zero the motors. For future reference we noted that below an input of 750, the right wheel (12) moves forward and the left wheel (13) moves backward.

**ADC:**

For our ADC to function, we had to connect the 8 outputs to pins 11-18. \( V_{CC} \) was the 5 V power supply. Pins 1, 7, 8, and 10 had to be connected to ground. Pins 2, 3, and 5 had to be connected to pins on the main board that could pulse them with high and low logic, 5 and 0 volts. The conversion process has several stages:

1. To trigger a new conversion, pin 5 (Interrupt) will start high. We must make pin 3 (Write) low and then return it to the high state. The conversion process starts when Write goes high.
2. When the conversion process is complete, pin 5 (Interrupt) will go low.
3. When we see pin 5 go low, we must make pin 2 (Read) low to load the new value into the outputs DB0 to DB7.

4. Next we read the values.

5. Finally, we return pin 2 (Read) to the high state. The next conversion can be started immediately.

To verify that the ADC was working, we had to test the outputs to see if they output logic high or low. To do this, we had to pulse high logic to the read port and low logic to the write port, then low logic to the read port and high logic to the write port. This allows the ADC to output voltages of 0 or 5 volts, and write a series of 0s and 1s to the microprocessor. We were able to do this by hand measuring the voltages of each port and obtaining different values of 0 or 5 volts for different pins with different incoming voltages. Appendix 3 shows a LabVIEW program used to analyze this.

The ADC was used to take a voltage from the gyroscope and output it to the microprocessor so that it could be read. The voltage is split into 8 bits containing a one or a zero. These can be read by the computer and changed back into a single value using the formula below:

\[ \text{Result} = IN0 + 2*IN1 + 4*IN2 + 8*IN3 + 16*IN4 + 32*IN5 + 64*IN6 + 128*IN7 \]
IN0 to 7 represent the 8 ports being read by the microprocessor in BASIC Stamp language. This formula is not necessary if you use a DEC3 command in BASIC Stamp. This command takes an 8 bit binary number and outputs it in decimal form as a number between 0 and 255, which corresponds to a voltage between 0 and 5. The larger range improves the resolution, thus making the feedback more accurate. The result can be converted to a voltage by multiplying by (5/255). The voltage can then be converted to an angular velocity using:

\[ V = 1.4336\omega + 3.6697 \]
\[ \omega = \frac{V - 3.6697}{1.4336} \]

Angular velocity can be converted to an angle by integrating over time. If each measurement is taken 20 ms after the previous one, we integrate over 20 ms per measurement.

This ideal approach did not work for us. We tried testing using the LabVIEW program in Appendix 3, the BASIC Stamp program in Appendix 4, and manual testing. We were able to get desired results at times with each method, but it was not consistent. Because we tried many different methods of testing, and many different high/low logic combinations at different ports, all revealing similar inconsistent results, we were forced to abandon the ADC and run the robot without feedback. However, the understanding behind the ADC and how feedback would have been incorporated are still valid.

*Programming:*

We used BASIC Stamp and LabVIEW programming in this project. The programs went through several iterations and all worked at different stages. We were able to read and write constantly using the program in Appendix 5, which stopped working after we changed computers. The outputs were not correct coming from the gyroscope, but we were receiving something. In the end, we were able to receive a number stored in the BASIC Stamp code, but not ADC data. The program is in Appendix 6. The programs are discussed in greater detail in the appendices.
As can be seen in the LabVIEW programs in the appendices, a UUID is needed. The intent of UUIDs is to enable distributed systems to uniquely identify information without significant central coordination. Thus, anyone can create a UUID and use it to identify something with reasonable confidence that the identifier will never be unintentionally used by anyone for anything else. In the first programs we used, the Bluetooth Discover and Bluetooth RFCOMM Service Discovery VIs were used to get the address and UUID. We found that this took too long, and improving the performance of our program was very important to us. As can be seen in Appendix 6, we removed the Bluetooth Discover and made the address a constant input. This saves enormous amounts of time and was very helpful for testing more quickly.

Another thing that was very helpful was removing the Bluetooth Read. The ADC only worked sometimes and keeping the Bluetooth Read meant having errors in our program and having to press “Continue” when the error message popped up. We were able to read constant values but not obtain feedback from the ADC.

**Wireless Communication:**

The wireless communication is done through Bluetooth, which uses a unique UUID with every connection to decrease interference. Data will be sent back and forth between the boe-bot and the computer using Bluetooth VI’s from LabVIEW, as shown in the programs in the appendices.

The first step in setting up the wireless communication was to install the Bluetooth device. When one of our teammates left he failed to inform us about how he had added the device on his computer. When we added the device it didn’t take long for it to disconnect. We had been selecting to use no passkey and eventually we figured out that a default passkey was needed. This kept the connection alive and solved the problem.

The next challenge was to make LabVIEW send and receive information from the wireless card on the boe-bot. We found that there existed Bluetooth VIs that helped with setting up the connection. We also found VIs such as Advanced Bluetooth Client and Advanced Bluetooth Server in LabVIEW Version 7.1 from which served as a base to develop our program.
After we made our program we had problems with the connection. This only happened in the lab, which caused us to believe the problem was interference with other Bluetooth devices in the lab from other group’s devices. We realized the address was different when we couldn’t connect to the one shown when we could establish communication with the Bluetooth. Therefore, we decided to keep the address constant. This, as mentioned in the Programming section, ultimately helped us in increasing the efficiency of our program since the time required to connect was dramatically decreased.

The biggest problem was not being able to receive correct data from the gyroscope. Our ADC seemed to not work most of the time and feedback control was impossible to perform. We therefore decided to create the best wireless controller possible so that we could adjust the path of the boe-bot in real time.

**Feedback Control:**

![Image 2 – Ideal Feedback Loop](image_url)
The feedback control corresponds to path corrections. The ADC will convert analog signals from the gyroscope into a digital signal that will be sent to the microcontroller. These signals will then be wirelessly sent to LabVIEW, which will find the angle at which the boe-bot is traveling. If the boe-bot is going too far off course then there will be wireless path-correcting instructions sent to the boe-bot. This feedback is done continuously.

The feedback we used is not ideal. We were not able to get the ADC to output correct signals, so we had to use feedback based on our eyes. If we saw that the bot was going off course we instructed the bot to turn right or left to correct the path.

**Final Product – Results:**

Our final product does not perform the exact application the project started out with. We can control the robot using push buttons through LabVIEW. The motion is smooth and seamlessly controlled. We can make the bot follow a similar path to that prescribed at the outset of the project. We added some obstacles to this path to prove the maneuverability of our robot.

**Future Work:**

The biggest problem was the ADC. If more time was available, using an ADC from Texas Instruments would be advantageous because it was already understood and had worked correctly the year before. Perhaps simply having more time to figure out how our own ADC works would be good as well. Clearly, however, using elements whose operation is already known would be easier.

A more enjoyable project would be a competition through an obstacle course using wireless control of the boe-bot, with the course gradually increasing in difficulty quarter after quarter. Also, having a lab during the quarter dealing with ADCs would be helpful. Another improvement could be to add adjustable speed for the bot.

**Conclusion:**

The concentration of our project was to wirelessly connect to the robot using LabVIEW, which was successful. Because we concentrated on this area, it was not
possible to focus on other areas such as the ADC. With a little more expertise and help with certain electronics, we would have been able to integrate feedback into our robot. If the ADC had returned data correctly we would have been able to use those numbers to understand the position of the bot. This, however, was not possible, but the outcome was still a success as we integrated smooth, wireless, real-time motion into the bot. We truly believe this project was a great way to finish ME 224. We learned many things throughout the quarter and this project is a good summary of these skills.

References:

Appendix 1

BASIC Stamp Program:

pulse_count VAR Word

LOW 12
LOW 13

main:
counterclockwise:
FOR pulse_count = 0 TO 250
  PULSOUT 12, 500
  PULSOUT 13, 500
  PAUSE 20
NEXT

PAUSE 500
clockwise:
FOR pulse_count = 0 TO 250
  PULSOUT 12, 1000
  PULSOUT 13, 1000
  PAUSE 20
NEXT

PAUSE 500

STOP
LabVIEW Program:
## Appendix 2

<table>
<thead>
<tr>
<th>Input</th>
<th>Time</th>
<th># Rotations</th>
<th>Voltage</th>
<th>Angular Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>5.7</td>
<td>3</td>
<td>0.21</td>
<td>-3.306842105</td>
</tr>
<tr>
<td>650</td>
<td>3.9</td>
<td>2</td>
<td>0.32</td>
<td>-3.222051282</td>
</tr>
<tr>
<td>700</td>
<td>6.3</td>
<td>3</td>
<td>0.404</td>
<td>-2.991904762</td>
</tr>
<tr>
<td>715</td>
<td>4.8</td>
<td>2</td>
<td>0.79</td>
<td>-2.617916667</td>
</tr>
<tr>
<td>730</td>
<td>8.2</td>
<td>2</td>
<td>1.51</td>
<td>-1.532439024</td>
</tr>
<tr>
<td>740</td>
<td>9.6</td>
<td>1</td>
<td>2.12</td>
<td>-0.654479167</td>
</tr>
<tr>
<td>750</td>
<td>N/A</td>
<td>0</td>
<td>2.6</td>
<td>0</td>
</tr>
<tr>
<td>760</td>
<td>9.7</td>
<td>1</td>
<td>3.04</td>
<td>0.647731959</td>
</tr>
<tr>
<td>770</td>
<td>8.3</td>
<td>2</td>
<td>3.62</td>
<td>1.513975904</td>
</tr>
<tr>
<td>790</td>
<td>7</td>
<td>3</td>
<td>4.41</td>
<td>2.692714286</td>
</tr>
<tr>
<td>850</td>
<td>5.9</td>
<td>3</td>
<td>4.79</td>
<td>3.194745763</td>
</tr>
<tr>
<td>900</td>
<td>5.8</td>
<td>3</td>
<td>4.82</td>
<td>3.249827586</td>
</tr>
<tr>
<td>1000</td>
<td>5.7</td>
<td>3</td>
<td>4.84</td>
<td>3.306842105</td>
</tr>
</tbody>
</table>

**Angular Velocity vs. Voltage**

$y = 1.4336x - 3.6697$

$R^2 = 0.9996$
Appendix 3

Using 8 ADC’s and connecting the 8 pins to the DAQ for testing using LabVIEW worked to a point. We changed the voltage and got different numbers, but sometimes the numbers were not close to 0 or 2048 (corresponds to 0 and 5 volts respectively). To pulse the ADC with logic at different pins, we manually inserted 5V and ground wires in a certain order described in the ADC section. We also tried to measure the voltages manually using the oscilloscope and multimeter. We could not take this as a positive sign that we knew how to pulse the ADC correctly or that it was working at all.
Appendix 4

Binary value output is a high or low logic from each of 8 pins. BASIC Stamp uses DEC3 command to output the same value in decimal form, a value between 0 and 255. Here we did not use the logic pulsing described above. We simply gave a pulse to pin 8 (Write) and values were displayed. When we changed the voltage, the numbers changed at times, but not at others.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

measurement VAR Byte

HIGH 8
LOW 8
HIGH 8
INPUT 8

DO

measurement = INL
DEBUG HOME, "Binary value:" , BIN8 measurement, CR,
   "Decimal value: ", DEC3 measurement
PAUSE 100

LOOP
```
Appendix 5

In the BASIC Stamp program below, the result is the output of the ADC from pins 0 to 7. CmdData is what is sent back to the microcontroller from the LabVIEW program after being analyzed.

`{$STAMP BS2}
LOW 12
LOW 13

LMotor CON 13
RMotor CON 12

LFwdFast CON 1000
LRevFast CON 500
RFwdFast CON 500
RRevFast CON 1000

CmdData VAR Byte
Result VAR Word

PAUSE 1000

WaitForConnection:
  IF IN5 = 0 THEN WaitForConnection

Main:
  HIGH 8
  LOW 9
  PAUSE 2
  LOW 8
  HIGH 9

Result = IN0 + 2*IN1 + 4*IN2 +8*IN3 + 16*IN4 + 32*IN5 + 64*IN6 + 128*IN7
'Result = IN15
SEROUT 1,84,[DEC Result]
SERIN 0,84,[DEC1 CmdData]
BRANCH CmdData,[Hold, Turn_Right, Turn_Left, Move_Fwd]
GOTO Main

Move_Fwd:
  PULSOUT LMotor,LFwdFast
  PULSOUT RMotor,RFwdFast
  GOTO Main

Turn_Right:
PULSOUT LMotor,LFwdFast
PULSOUT RMotor,RRevFast
GOTO Main

Turn_Left:
PULSOUT LMotor,LRevFast
PULSOUT RMotor,RFwdFast
GOTO Main

Hold:
GOTO Main

The formula node in the LabVIEW program can be altered to perform calculations on the data read from the ADC. The calculations are seen in the ADC section. In this program the connection is established, then it reads, then a time delay, then it analyzes and writes. We did receive values that the program above read, but they were not correct.
Appendix 6

In this final program we did not include Bluetooth Read because it only stalled the program. The ADC did not work either, so there was no real point in including it simply to read a constant value from the BASIC Stamp program.

'${STAMP BS2}
LOW 12
LOW 13

LMotor CON 13
RMotor CON 12

LFwdFast CON 800
LRevFast CON 700
RFwdFast CON 700
RRevFast CON 800

CmdData VAR Byte
Result VAR Word

PAUSE 1000
CmdData = 3

Main:
LOW 8
HIGH 8
INPUT 8

Result = INL 'INL reads all 7 output pins, so you don’t have to add
IN0+2*IN1+4*IN2…
‘Result = 103 Any random number could be used to send a result to LabVIEW

LOW 8

SEROUT 1,84,[DEC3 Result] 'DEC3 transforms Result from 8 bit format to
decimal format
SERIN 0,84,[DEC1 CmdData]
BRANCH CmdData,[Hold, Turn_Right, Turn_Left, Move_Fwd]
GOTO Main

Move_Fwd:
PULSOUT LMotor,LFwdFast
PULSOUT RMotor,RFwdFast
GOTO Main
Turn_Right:
PULSOUT LMotor,LFwdFast
PULSOUT RMotor,RRevFast
GOTO Main

Turn_Left:
PULSOUT LMotor,LRevFast
PULSOUT RMotor,RFwdFast
GOTO Main

Hold:
GOTO Main

Note that the string values in the true/false loops must be in the form ‘ 1 ’. There must be spaces before and after the 1 in order for the BASIC Stamp program to understand that the string number is starting and ending. The false side contains a string with nothing, not even a space. This allows the concatenated string to not contain extra values. We also abandoned the Find Address part in the beginning of the program. We input the address manually because the wireless kept finding other groups’ wireless cards and even someone’s cell phone. Doing this also increased the speed of the program.
By pressing a button in the motion control panel we are able to move the robot in that direction. The turn speed is 3 rad/s.