Final Report

Walid Mnif, Lim Seang, and Tamer Shadid
ECE 4006B
Summer 2003

July 31, 2003

Georgia Institute of Technology
College of Engineering
School of Electrical and Computer Engineering
Executive summary

The demand for wireless services are increasing every day as the scope of its application is expanding into our every day lives. Bluetooth is the latest arrival in this field; it is an emerging protocol with the premise of being the next big thing in short distance communication applications. Bluetooth protocol offers a reliable and secure connection between any group of devices, without the need for a supporting backbone, and with low power consumption. It has already found a place among every-day electronics such as phones and headsets, and is quickly taking over sensors’ data communications in the manufacturing industry. Among the slew newly available Bluetooth enabled products is the handheld PC manufactured by Compaq known as the iPAQ. The iPAQ supports a Windows CE operating system and thus supports a lot of the basic Windows functionality that everyone is familiar with. It has the ability to wirelessly connect to any other Bluetooth enabled peripheral. For our project, we decided to design a new robot with Bluetooth capability, which will be wirelessly controlled by the iPAQ through the Bluetooth link. The robot we used is the Amigobot, a multi-functional robot with added features such as sonar sensors. A combination of two PIC18F452 micro-controllers and a UART Bluetooth converter were used in order to establish the link and control the robot. We were able to establish a full duplex communication between the iPAQ and the robot. A graphical user interface was created on the iPAQ to control the robot and monitor the readings of some of its sensors. All of the goals that we set for the project were successfully accomplished. In the end we successfully had a robot being controlled by the iPAQ through a Bluetooth link, with the robot also providing sensor feedback to the iPAQ. Such a design has many applications in a variety of fields. An enhanced version of this robot may be marketed as a new search robot that can be wirelessly controlled while sending back its sensors output. Big demand has emerged lately for similar products especially after the September 11th events.
Table of Contents

Executive Summary 2
Introduction 4
Design Alternatives and Tradeoffs 5
Marketing and Cost Analysis 7
Project Technical Details 9
Tasks and Schedule 17
Product Demonstration 18
Conclusions 20
Bibliography 22
Appendix Amigobot 23
Project Gant Chart 30
Appendix Software Code
Introduction

Bluetooth is gradually gaining acceptance throughout the technological field and being adopted by all those in need for a short distance wireless link. Despite having a transfer data rate limited to 1 Mbits/second, the Bluetooth protocol does offer a full list of unique services to its users, which sets it apart from other protocols. While other protocols such as WIFI have gained control of wireless Internet and applications of that nature, Bluetooth is sprawling around the manufacturing field becoming the hottest technology offering cheap and secure transfer of sensors’ data. It is also becoming the most used protocol in the areas of robotics involving remote controlled capability and multi interactive robotic environments. In our project, we are establishing a Bluetooth link between an amigobot and an iPAQ PDA. Our primary goal of the project was to wirelessly control the robot movement using the iPAQ. The secondary goal was to send feedback from the robot sonar sensors and display it on a GUI in the iPAQ.

In order to add Bluetooth capability to the Amigobot, a Bluetooth converter was purchased. Two PIC18F452 micro-controllers made by Microchip were employed on top of the robot to form a communication medium between the converter and the robot. The first micro-controller receives the commands from the Bluetooth converter via UART and forwards the command to the second PIC via SPI. The latter PIC analyses the command and sends the corresponding control data to the Amigobot via UART. It also polls three of the robot’s sensors every second and sends a summary of their input back to the master micro-controller which then forwards it to the converter and then on to the iPAQ. The second PIC also has to make sure to regularly send update packets to the robot in order to keep its connection as specified by the robot’s protocol. All of these communication streams were implemented thanks to a complex interrupt based system synchronizing each channel.
according to its priority. The code was written in C language and compiled using Ht-Soft’s PICC18 compiler.

On the iPAQ side, the BTAccess library was used in order to utilize its Bluetooth ability. The BTAccess library was written especially for the Windows CE environment of the PDA, and contains a multitude of Bluetooth related functions and objects. The graphical user interface was developed using C++ and used to set up the connection and send the control commands over Bluetooth. Two parallel running C++ threads were implemented in order to support the full duplex Bluetooth communication channel by receiving and sending data simultaneously.

In the final version of the project, the robot was successfully controlled by the pda. Different move options were made available to the users including rotation, direction of movement, and three different speed settings. An acknowledgment system was also integrated to insure the reliability of the channel. The PICs also send regular updates on three of the robot sensors that it polls, and the result was displayed on the GUI. So, overall this project was very successful in that it successfully achieved its primary and secondary goal. An enhanced version of this robot could prove to present many marketable opportunities for the future, including a full implementation on the sonar feedback to actually map out and traverse terrain that is not in plain view of the operator.

**Design Alternatives and Tradeoffs**

One of the most important characteristics of iBlueBot is its implementation of Bluetooth wireless technology. The iPAQ 3970 has Bluetooth capabilities already built-in. The first step of designing the software application for the Pocket PC was to find out how we can control and use the Bluetooth capabilities installed in the iPAQ. The options were as follows.
- Either purchase a UNIX operating system for the iPAQ and download an open source Bluetooth protocol stack project.
- Or use the originally installed Windows CE operating system and use a C++ library to access the Bluetooth Widcomm protocol stack built into iPAQ.

The latter option was selected for many reasons.

1- The front end of the iBlueBot must be user friendly because one of the features of iBlueBot is giving the user the ability to control the movement of the robot. Because we had no experience in building user-friendly applications in Unix, we built our GUI with Embedded Visual C++ (available for free).

2- Though most Bluetooth protocol stacks are very similar, we cannot assure that available UNIX open source Bluetooth Stacks would work on iPAQ.

3- Time constraint. The limited time we had did not allow for spending too much time for figuring out the Bluetooth Protocol Stack.

4- iPAQ is most widely used with Windows CE. Therefore, our application will be more useful if it’s programmed for Pocket PC OS.

Once we were settled on using the available Embedded Visual C++ software, we then looked at possible options for controlling our robot. In the limited window of time that we were given to order parts, we decided that we should probably implement a micro-controller to directly interface and deal with the robot’s immediate communications needs. It would have also been feasible to just control everything through the iPAQ, but we were not sure how the converter would interface with the Amigobot. It turned out that the Amigobot also uses serial communication, but if we decided to establish a direct link between the iPAQ and the Amigobot, it would limit our iBlueBot to being operated only in the presence of the iPAQ. With the robot’s immediate control being handled by the PIC micro-controller, we
took strain away from sending lengthy packets through Bluetooth link and we also opened up the possibility of having the AmigoBot executing predefined paths even when the AmigoBot was not within range of the iPAQ.

After a little search for different Bluetooth converters, we just decided to use the one that our instructor had suggested. The main reason was cost. There were many other Bluetooth options that would have probably worked better, but these kits cost upwards of 2000 dollars. The advantage these kits would have been to us is that they would have given us a better idea of how we were going to implement the Bluetooth link, as the kits available online had good references and had been available for quite some time. The BrainBoxes Bluetooth converter was a new product and we just didn’t exactly know what we were going to get from it.

The micro-controller selected was the PIC18F452. The main reason we selected this particular micro-controller is that all of the group members had taken a class that extensively utilized this chip. So because of this, we also had access to a good reference (the text for the class) and we were already familiar with the setup of the PICs. We did not explore any other micro-controllers, as the PIC was just the best for our limited time frame project.

Marketing and Cost Analysis

For an immediate introduction of this design project into the market, we believe that it should start competing in the new bomb disposal robots field. This is a type of product that has seen a lot of expansion lately, and governments are also allocating larger budgets for such equipment. This type of robot is used to remotely disarm potential bombs, and search for and dispose of mines in mined fields. There is already a full array of such products on the market with price ranges between 20,000 and 40,000 per unit. All of these products however use radio-controlled robots like the packbot of the iROBOT company. Our robot presents a clear advantage over all these designs since it uses a Bluetooth link. First it offers a virtually error
free protocol with a 128 bit encryption guaranteeing no suspicious interference, and no erroneous commands reaching the robot. The iPAQ control also gives the product additional versatility thanks to the ease of use of its GUI and its portability. To add to all of this, is the main advantage of our project: Bluetooth offers the opportunity to different robots to sense each other’s arrival and interact among each other. This gives a whole new meaning to the field of automated searches. It would become possible to deploy many Bluetooth robots in one mined field for example and have them separately search the terrain while updating their path to the others so that no two robots search the same place and insure the most efficient search possible. However, for such a product to see the light we would have to use a more advanced robot that is more resistant and better moving than the Amigobot. The un-tethered urban robots made by Inuktun Services may seem to be a better match despite its cost, which is around 20,000$. The expected budget for such a prototype may be broken down as follows:

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robot</td>
<td>20,000 $</td>
</tr>
<tr>
<td>iPaq</td>
<td>700 $</td>
</tr>
<tr>
<td>Bluetooth converter</td>
<td>200 $</td>
</tr>
<tr>
<td>Microcontrollers and circuits</td>
<td>10 $</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20,910$</strong></td>
</tr>
</tbody>
</table>

The final parts cost comes within the regular range of competing products while also offering the Bluetooth capabilities. Licensing other products rather than building our own can minimize all manufacturing costs. All that may be needed is a small assembly line to put everything together and download the right program on each component. A free maintenance schedule may be set up with the customer for a year, and then a charge can be placed on repairs and maintenance thereafter. Already around 300 hours of work were put in the project. In order to enhance it to raise it to market standard, an additional 300 hundred hours of engineering work should be expected. Considering a cost of 25$ an hour for entry level engineers working on the prototype, a 7500$ budget should be allocated for researcher. At
least the same amount should be allocated every year for further research and enhancements.

This product is not a high volume selling type of item, so it may be possible to just hire two engineers performing the assembly and programming of the different components. Overall, the company’s annual budget may be divided as follows:

<table>
<thead>
<tr>
<th></th>
<th>cost in $</th>
</tr>
</thead>
<tbody>
<tr>
<td>research labor</td>
<td>7500</td>
</tr>
<tr>
<td>research equipment</td>
<td>30000</td>
</tr>
<tr>
<td>assembly labor</td>
<td>80000</td>
</tr>
<tr>
<td>location rental</td>
<td>10000</td>
</tr>
<tr>
<td>marketing</td>
<td>10000</td>
</tr>
<tr>
<td>Overhead (transportation, administrative fee…)</td>
<td>10000</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>147500</strong></td>
</tr>
</tbody>
</table>

The robot will be sold at a price of 30,000$, with an expected yearly volume sales of just 30 units, yielding a net expected income of 900,000$. After the initial period of research and design, the product should yield roughly 750,000$ annually.

**Project Technical Details**

A Pocket PC application was developed with a Graphical User Interface (GUI) to be the front-end of the iBlueBot product. The GUI has multiple command types which are used to control the Amigobot in the most efficient way possible. Figure 1 shows the GUI and its features.

*Creating the Pocket PC application*

The objective of the Pocket PC application was to be able to send and receive data from and to the Bluetooth converter. The graphical user interface was developed for the iPAQ with Microsoft Embedded Visual C++ [1]. To utilize the built-in Bluetooth technology on the iPAQ, the Windows application had to have access to the Bluetooth Protocol Stack. For this purpose, the BTAccess library was used. BTAccess, available from High Point Software, is a C++ class mode library for accessing the Widcomm Bluetooth Protocol Stack.
built in the iPAQ. It offers classes and functions to manage Bluetooth functionalities such as starting a serial communication with other Bluetooth devices.

The application was multithreaded to be able to send and receive without interruptions. The read-in thread tries to read from the serial port once every second. It does not use the WaitCommEvent command to wait for a character to be received before reading the data from the port. The reason for not using WaitCommEvent lies in that when we use the Bluetooth converter to send data relatively fast to the iPAQ, the WaitCommEvent stops recognizing receiving events after a while. Therefore, WaitCommEvent waits forever. To solve this problem, we removed the WaitCommEvent and instead we set a timeout for reading data on the used port.

![Graphical User Interface on iPAQ](image)

**Figure 1.** Graphical User Interface on iPAQ

*The functionality of the GUI*

When the application is first started, a virtual port is created to read and write to COM8, and a thread is created for reading received data. COM8 is the virtual serial port used for outbound Bluetooth serial links.
The Read Thread

This thread keeps looping while the connection is still open. It tries to read one byte from COM8 once every time around the loop, and then it updates the distance bars in the GUI with the newly received byte. The timeout for Reading from the port is set to be 1 second.

The bars represent how close the robot is from objects around it.

The GUI has a Connect button that does the following when pressed:

1- Search for Bluetooth peripherals
2- Once the Bluetooth Converter is found, it connects to it.

Sliders were also used to control the speed and direction of the robot.

The documentations for BTAccess and embedded visual C++ were frequently referred to during the process of developing the final product. Furthermore, a demo program for BTAccess, developed by High Point[2], and the ECE4180 RemotCam project (fall 2002) [4180] was also helpful as a starting point for developing the Windows CE application for iBlueBot.

BL-521 Brainboxes Bluetooth Converter Configurations

The BrainBoxes BL-521 is a stand-alone Bluetooth adapter that can serially connect any device with a RS232 port with any Bluetooth enabled device and does not require any addition software. The Bluetooth converter was configured to use 19200 baud rate, and low security mode. One of the major problems we had with the Bluetooth Converter was its sensitivity to the values of CTS and RTS pins on the RS232 port. If these two pins are left floating, the Bluetooth converter will not function at all. To connect the PIC to the converter, we connected both CTS and RTS to ground. It is important to look over the connections if the communications are not behaving predictably to ensure that the CTS, RTS, RX, TX, and GND pins are correctly connected at this junction. To configure the converter you must use the supplied gender changer to connect to your PC.
PIC18F452-AmigoBot Interface

The Setup

The PIC 18F452 was used to interface the Amigobot and the compiler used was the Hi-Tech PICC18 compiler; a demo version can be found from Hi-Tech’s website [HI]. Chris Twigg programmed the boot loader QwikBug onto the PIC to give a means of accessing the PIC’s program memory. And finally, Tera Term Pro was used to connect to QwikBug from the PC through a serial connection. QwikBug and Tera Term Pro can be found at John Peatman’s PIC website, http://www.picbook.com [picbook.com].

To begin programming the PIC, there were a few things that needed to be setup in the code to ensure that the PIC would function properly. A Looptime subroutine and a BlinkAlive subroutine was borrowed from Peatman’s text[5]. The Looptime subroutine basically made sure that the mainline code was executed once every 10 ms, while the BlinkAlive subroutine blinked an LED every 2.5 seconds to give an indication that the code was still being executed. The parameters for the interrupts and I/O ports found in the Initialization() routine were also found in Peatman’s text[5].

After several weeks and many versions of the code, the Initialization() parameters were finally correct and finalized. The main parameters of interest in the Initialization were the ones that setup the UART baud rate, serial peripheral interface, low and high priority interrupts, and the I/Os. On the other side of the communications was the Amigobot that also had a UART port which was setup to match the speed of the PIC’s UART. To establish a physical link, a special cable was made to have a DSUB9 connection (to the PIC) on one side and an RJ11 connection (to the Amigobot) on the other (See Appendix Amigobot). With everything setup, the first step was to establish communication with the Amigobot.

The Amigobot Technical Manual [4] was used to find out how the Amigobot communicated with other devices through its serial port. After looking over the manual, the
group first targeted the task of sending commands to the Amigobot. To accomplish this task, the Amigobot required a line of communication to be established. Using the Amigobot technical manual [4] we found the specific commands and the order in which the commands needed to be sent, but the checksum calculation instructions in the manual were somewhat ambiguous. We referred to code from a previous group, who had used the NIOS board with an Amigobot, to see how they calculated their checksums [7]. They had hard-coded their checksums, but the values were correct so we just used their calculated checksums to figure out the equation for calculating our checksums on the fly (See Appendix Amigobot for packet details and checksum calculation). When the code was tested, we found that the Amigobot was not responding and so after some debugging we found out that we had to swap a couple wires on the serial line (namely pins 2 and 3). After the swap was made, our Amigobot responded and a communications link was established.

After establishing communications between the Amigobot and the PIC, we realized that it would be necessary to add an additional PIC to the project. The reason behind this addition stems from the fact that the PIC only has one UART port that we originally planned to be used with the bluetooth converter, but once we implemented the Amigobot communications through the lone serial port, we were now without a serial port for the converter. To remedy this problem, we looked through the text [5] and found a way to connect two PICs together using the PIC’s serial peripheral interface (See Appendix Amigobot for specifics). Using this methodology we would now have resources from two PICs (namely the extra serial port interface). For the purpose of having instant response to commands issued from the iPAQ, we decided to set the PIC connected to the converter as the Master PIC, while the PIC connected to the Amigobot became the Slave PIC.

Using a derivative of the master-slave over SPI scheme found in Peatman’s text we setup the Initialization parameters of both PICs. It was decided that the iPAQ would only
send one byte to the PICs, and this byte would contain the command to be executed and any parameter that would be needed to go with the command. A list of commands and a figure of how we break up the commands is seen below in Figure 2.

![Figure 2. Table of commands and how the decoding works.](image)

When the byte reaches the master PIC, an interrupt occurs and while in the interrupt, the master PIC forwards the BYTE to the slave PIC. To do this, the master PIC drives one of its output pins low, and then transmits data to the slave by shifting bits onto the SPI buffer. Once the slave has received the byte, an interrupt occurs on the slave PIC and it then proceeds to read the byte and decode it using a large switch statement. The PIC then assembles an array of bytes and calculates the checksum to send to the Amigobot (See Appendix Amigobot for details on packet assembly and transmission). To make sure that everything was working correctly, we hooked up the iPAQ and sent various commands to the Amigobot.

In testing the communication, we noticed that it wasn’t working properly, so to test code, LEDs were connected to output ports on the PICs. If we had access to he ECE4175 lab
we could have used the ICD (in circuit debugger) from MicroChip to set flags and debug our code; but since we were without such hardware, we just used LEDs as flags. Basically we would set up our code to light up an LED once it reached certain critical sections of the code where we suspected there might be a mishap. Using this process, we were able to ‘trap’ and find out where some code was written wrong or where some code had to be rewritten. In the case of sending commands from the iPAQ to the Amigobot, it was discovered that the slave PIC was never executing the low priority interrupt (see Appendix Amigobot), and thus nothing was ever being sent to the Amigobot. Some initialization parameters were adjusted to correct the low priority interrupt and we finally got communication and the Amigobot began responding to all commands that the PDA was sending to it and thus our primary goal was achieved.

![Figure 3](image.png)

**Figure 3.** Timing diagram of PIC just receiving data every second instead of each time the SIP is sent.

The secondary goal was to receive sonar feedback from the Amigobot. To start on this part, a good understanding of the Amigobot’s server information packets or SIP was needed. The technical reference manual indicated that the Amigobot sent one SIP every 100 milliseconds, so we decided to employ a high priority interrupt to receive the SIP every time the Amigobot sent it. When we implemented this scheme we discovered that the slave PIC was having receive buffer overrun errors. To remedy this problem, a new approach was taken that consisted of just turning on the high priority interrupt once every second and then just receiving the packets and clearing the overrun error if it ever came up. Figure 3 assumes that the timing stays consistent but that is not always the case as the Looptime is not always
10ms because interrupts frequently increase the loop time. Also, the packets were actually limited to the array of 50 bytes that were set aside, so if the interrupt was enabled halfway through a packet, it would just receive the rest of that packet, and then wait till the next 100 milliseconds and start receiving that packet until the 50 byte array is full. Though this is probably not the best way to go about retrieving the data, we were limited to just 1 week to work on this and it seems to work well enough to show that data from the Amigobot can be read through the blue tooth converter and displayed onto the iPAQ. We could have tested cases for improved time between polling the sonar and for a better way of reading the data from the RCREG in the PIC so as to eliminate the occurrences of buffer overrun errors.

As for the SIP, we referred to an appendix from the NIOS board robot group for an example SIP, and discovered that we would only need six bytes from the whole series of 33 bytes. But the problem is that when we were enabling the receive interrupt on the PIC, it would only read up to 50 bytes and then disable itself. So we had to write logic to locate the header byte and then make sure that at least 29 more bytes were read and fit into the 50 byte array. We then extracted the six bytes of interest from the 50 byte array and converted the values into millimeters (See Appendix Amigobot for details on this topic). We ran into many problems when we tried to test this code. The first problem stemmed from the SPI interface, which now had to be altered, and then random issues with the converter (discussed in a previous section). The SPI issue was addressed by having the master PIC query the slave PIC once every second to see if the slave PIC had new sonar data. If new sonar data were available, the master PIC would then transmit the new data to the iPAQ. A more in depth explanation of this whole process is found in the Appendix and looking at the code.

Testing this code turned out to be one of the toughest steps of the project. We had to use 7 outputs to light LEDs at many parts of the code. We had to find out how often the UART was actually getting good data, and how often a buffer overrun error occurred from
the SIPs being sent to the slave PIC. And because of these errors we had to check to make sure that we were even finding the header bytes of the SIP, thus ensuring that valid data was being received from the Amigobot. We also used a byte to character function that converted the individual bits of the bytes that the master PIC was sending to the iPAQ into a series of ASCII character representations of the bytes, thus enabling Tera Term Pro to display the actual data being sent to the iPAQ. This proved to be a very useful debugging method as one group member could work on the data from the sonar, while the other member could concentrate on just getting the converter to send data to the iPAQ. On an extra note, the Appendix Amigobot has very detailed explanations on certain blocks of code.

**Tasks And Schedule**

The project was broken down to four major building blocks: connecting the iPAQ to the converter, sending and receiving data through the converter, interacting one PIC with the converter in order to send and receive data, connecting the other PIC to the Amigobot and sending control packets to it, and getting input from the robot sensors and forwarding it to the iPAQ. Since the beginning of the semester, the group was divided into two subdivision: Tamer and Walid researched all the Bluetooth related material and were assigned the responsibility of insuring the transfer of data from the iPaq to the PIC connected to the amigobot, Lim was the member concerned with the Amigobot functionality. The group had encountered major shipping delays in the beginning as both the Amigobot and the Bluetooth converter only arrived by the end of June. However, we were able to remedy these delays rapidly by borrowing a robot and using a commercially available Bluetooth dongle as a temporary replacement for the converter. Once all the parts arrived, all the roles were reassigned in such a way Lim was still working on the robot, while Tamer was the one working on the iPAQ GUI and its communication with the converter, and Walid was running the communication bridge from the converter to the first PIC, and on to the second one. Once
we were able to wirelessly control the robot, we moved into implementing the receive channel. Tamer coded an additional receiving thread on the iPAQ that received packets and displayed the sensor output. Walid added the receive functionality to the first PIC and moved into assisting Lim who had studied the sensor data and implemented a complicated algorithm in his code to extract the right information. All of the tasks were challenging in different ways. Tamer’s part may be considered the most difficult mainly because he had to learn the unfamiliar Windows CE and the functionality of BTAccess libraries. Walid had to deal with configuring the converter and interacting it with PIC in the absence of a manual describing the hardware. Trial and error had to be used in a few cases in order to debug the converter’s behavior. Lim had to learn the complicated robot communication algorithm and implemented it on the PIC while keeping it in sync with Walid’s PIC. Overall, our group was able to stick to its schedule most of the time except for a few tasks, which had to be extended by one week, like controlling the robot by the iPAQ and displaying the sensor feedback on the PDA. The main reason behind our group’s success in its time management is the modularity that our task assignment provided. No task was overwhelmingly reliant on the other, and whenever a task was getting late on schedule, the whole group would meet and undertake it until it is back on track. The Bluetooth dongle was also a great help to the advancement of the project. It allowed Walid and Tamer to exploit the Bluetooth capability of the PDA and the converter separately before interacting them together. Refer to the Gantt chart on page 30 for a rough overview.

**Product Demonstration**

The prototype demonstration was done in three steps to cover all the capabilities of the robot. The demonstration was demonstrated in an open space to test all the robots speeds. After connecting the iPaq to the robot, we start by showing all the possible movements of the robot. First go forward and change speeds to each of the three available. Then we stopped
the robot and made it rotate to the left and to the right. We then proceeded to make it go forward again and then started making it move in zigzag to the left and right while changing the speed settings. Secondly, we demonstrated the functionality of the sensor feedback. Using a big piece on carton with the robot still, we block each of the three sensors one at a time and verify the changes on the PDA’s display. Lastly, we drove the robot toward the wall and verified that the PDA’s display for the front sensor changed while the robot is moving towards the wall.

In order to get the prototype connected and working a series of steps has to be followed:

1) Make sure that power is off on the Amigobot (power switch is located at the bottom of Amigobot).

2) Also make sure that the serial connection from the converter to the master PIC is initially disconnected. This must be followed because otherwise the converter will send data to the PIC before the PIC has started executing code, which, in turn will

![Converter hook-up to master PIC.](image)

Make sure this is initially disconnected when power is turned on

**Figure 4.** Converter hook-up to master PIC.
actually prevent the PIC from ever executing the code until it has been reset. If this step is neglected, just turn off the power and restart from Step 1. In the worse case, the master PIC’s memory will be erased and would then need to be reprogrammed.

3) Now flip the power switch on, and you should hear the Amigobot say the phrase “connected”.

4) Once you hear this, and the power has been on for a couple seconds, it is now safe to connect the converter to the serial interface of the master pic (see Figure 4).

5) Check to make sure that both blink alive LED’s on the PICs are blinking and also make sure that the converter is on (indicated by a green LED).

6) Open the Bluetooth application on the iPAQ.

7) Press connect in order to set up the Bluetooth serial link, and wait for connected to be displayed on the status panel.

8) At this point the robot is fully operational. It can be wirelessly controlled using the go, stop, left and right button in parallel with the speed bar. NOTE: The way that we have it programmed, the robot will not respond to any commands until the GO command is issued.

Conclusion

Overall, the project was a total success. Our initial goal was to remotely control the robot with the iPAQ, and we also added sending feedback as a secondary goal. The main goal was completely achieved with even a large array of robot movements implemented such as rotation, zigzagging and a three-speed setting. The secondary goal was also accomplished. A receive channel was setup and was used to receive command acknowledgements, and sensor feedback that was displayed on the iPAQ. The whole prototype is still in need however of enhancement in order to make market ready. A better robot is an important addition to insure its entry as a competitor in the bomb disposal market. So an additional
prototyping cycle will be needed after this in order to integrate the new robot in the design. However, for immediate application we may be able to separately market the PIC/Converter combination. The decision of implementing the converter on the microcontroller turned out to be very beneficial should this be presented to the market. It is possible now to add any kind of sensor to the prototype and have it send the output to the iPAQ. Such Applications are becoming very popular in the manufacturing field. The general approach to the project was very judicious. Lots of care was given to insure a certain level of modularity and portability among components. Because of the separate component structure of the iBlueBot project, a future group may be interesting into duplicating the design into another robot and try to make them both interact in order to accomplish tasks.
Bibliography

[1] Microsoft Mobile Devices, Available HTTP:
http://www.microsoft.com/mobile/pocketpc/default.asp

http://www.high-point.com/

[3] ECE4180 RemoteCam, Available HTTP:
http://users.ece.gatech.edu/~hamblen/489X/to2proj/remcam/

http://www.ece.gatech.edu/research/labs/diglab/downloads/AmigoTech.pdf


[6] Hi Tech Website, Available HTTP:
http://www.htsoft.com/

[7] ECE4006 NIOS Robot Group website, Available HTTP:
http://users.ece.gatech.edu/~hamblen/4006/projects/nios_robot/
Appendix Amigobot
iBlueBot Physical Layout

Though it looks quite messy, the current state of the product serves its purpose well for prototyping, and of course the finalized version of the product will have all these components on a printed circuit board.

Some of the components from Figure A1 are numbered and are the main physical components of the product thus far:

1. This is one of the two MAX232A UART converters that converts the +/- 12 volt UART signal to a TTL logic level signal that the PIC can read.
2. This is the slave PIC18F452 from MicroChip.
3. This is the master PIC18F452.
4. This is the UART connection from the MAX232A chip (from the slave PIC) to the DSUB9 connector that is connected to the Amigobot.
5. Under this little flap up panel is resides the pins for the 12 Volt power source which is powering a voltage regulator on the breadboard.
7. The RJ11 end of the special cable (DSUB9 to RJ11) that connects the Slave PIC to the Amigobot.
8. The voltage regulator that steps down the 12 volt from the Amigobot to the 5 volts that all the components use.
9. The AmigoBot, provided by the instructor
10. Not shown is the bluetooth enabled Compaq iPAQ provided by the instructor.

Figure A1. The current physical layout of the iBlueBot
Figure A2. Schematic iBlueBot
Sending Data to AmigoBot from slave PIC

Retreiving data from SPI buffer and building a variable array

A variable Array is set up to have at least 9 elements. This example will cover a standard command (commands with six bytes). The other important commands such as the SYNC commands and OPEN command are covered in the Amigobot Technical Manual.

```c
unsigned char TXcommand[] = {0xFA, 0xFB, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

The first two elements contain a defined header that the Amigobot recognizes. The third element gives you the number of data bits + 2 (accounting for the checksum).

When an interrupt occurs in the slave PIC, it reads the data off of the SSPBUF (the SPI data buffer) and then takes that data and decodes the command in the `CommandHandler()` subroutine. The data will be in the form of a byte, where the first nibble (high nibble) is the command and the second nibble is the parameter (used only in Speed command thus far). In the following little section of the `CommandHandler()` subroutine, it can be noted that the command byte read from SSPBUF is now contained in the variable `COMMAND`. In the code, the switch statement is switched `HNIBBLE`, and in this case it is the rotate right command. The command handler then assembles the data part of `TXcommand[]`, namely elements 3 through 6. The 3rd element defines the actual command that the Amigobot recognizes. The table for all these ‘client commands’ is found in the Amigobot Technical Manual. In this case, 0x0D corresponds to command #13, the DHEAD command in the technical manual. Referring to the technical manual, the DHEAD command has an argument of type int (according to the technical manual it means it is composed of two bytes). The argument is will specify how many degrees to rotate the robot. The line: `TXcommand[4] = 0x3b;`, indicates that the following arguments will be a negative integer. The other alternatives are 0x3b (for positive int) and 0x2b for a string type (this is covered pretty well in the technical manual). Then the next two bytes filling up element 5 and 6 are the actual argument (low byte first).

```c
void CommandHandler()
{
    char LNIBBLE;
    char HNIBBLE;
    char CSUMH;
    char CSUML;
    HNIBBLE = (COMMAND >> 4) & 0x0f; //shifted to right 4 bits then mask
    LNIBBLE = (COMMAND) & 0x0f;  //mask off top 4 bits
    switch (HNIBBLE){
        case 3://right 30 degrees
            TXcommand[3] = 0x0D;
            TXcommand[4] = 0x1b;
            TXcommand[5] = 0x1e;
            TXcommand[6] = 0x00;
            break;
        //end case3
```
Calculating Checksum and Sending byte array to Amigobot

In following the example, \texttt{TXcommand[]} looks like this thus far:

\texttt{TXcommand[]} = \{0xFA, 0xFB, 0x06, 0x0D, 0x1b, 0x1E, 0x00, 0x00, 0x00, 0x00\};

The following code calculates the checksum. Basically it is just adding the 4 bytes together but as if they were 2 words. In this case the code basically does \texttt{0D1b + 1E00 = 2B1B}. It then updates the variable array by inserting the 2 byte result into elements 7 and 8.

\begin{verbatim}
CSUML = TXcommand[4] + TXcommand[6];
if (TXcommand[4] > CSUML)//check if carry bit
{
    CSUMH = 1 + TXcommand[3] + TXcommand[5];
} //endif
else
{
    CSUMH = TXcommand[3] + TXcommand[5];
}
TXcommand[7] = CSUMH;
TXcommand[8] = CSUML;
\end{verbatim}

And the Result:

\texttt{TXcommand[]} = \{0xFA, 0xFB, 0x06, 0x0D, 0x1b, 0x1E, 0x00, 0x2B, 0x1B, 0x00\};

To send the data, a loop was created to traverse through the array and send each byte to the Amigobot starting with the header byte. The loop was derived from the NIOS robot group’s code to traverse through their predefined arrays.

\begin{verbatim}
loopsize = (TXcommand[2]) + 3; // using bytecount of command + 3 for the header and byte count byte
i = 0;
for (i = 0; i < loopsize;i++){
    TXREG = TXcommand[i];
    while (!TXIF){} //wait till pic is ready for next byte
}
\end{verbatim}
Retrieving Sonar Data from Amigobot

This was a bit tougher than sending data to the Amigobot. The tricky part was the fact the UART receive buffer on the PIC is limited in size to only 2 bytes. And if too many bytes are sent (as with the Amigobot which sends 33 bytes at a time), a buffer overrun error occurs on the PIC which disables receive functions and you end up losing data. Due to time constraints we didn’t really solve the problem, but we formulated a way around the problem by just sampling the data once every second by simply enabling the receive interrupt, and then disabling the interrupt once 50 consecutive bytes were received. Obviously that would mean that we would miss data and the refresh from the sonar sensors would be slower than if we triggered an interrupt every time the Amigobot sent a packet, but nonetheless we do show that it is possible to send data from the Amigobot to the iPAQ.

To roughly time the receive function, the function `getAmigoData()` was executed once every one hundred 10ms Looptimes.

The code first starts out by clearing any existing overrun error flag from a previous receive. It then enables the receive and waits at least 150ms with the following code:

```c
RCIE = 1;
while (k <= 15){
    while(!TMR2IF){}
    TMR2IF = 0;
    k++;
}
RCIE = 0;
...
traverseSIP();
```

In the high priority interrupt there is also a counter that will disable the receive after 50 packets were received, but if 150 seconds passes by before the 50 packets were received, this code will also disable the receive. So during the 150 ms window of time, the PIC has enabled interrupts and is repeatedly going into the interrupt and receiving bytes of the SIP, until the 50 byte array `AmigoSIP[]` is full or the allotted time runs out. The `traverseSIP()` function is then invoked to extract bytes 23-28 of the embedded SIP in `AmigoSIP[]` (but not necessarily indices 23-28 of the `AmigoSIP[]` array which is probably shifted, take a look at the code to get a better understanding of this function) and copy these bytes into a variable array:

```c
unsigned char SONARDATA[] = {0x00,0x00,0x00,0x00,0x00,0x00};
```

To understand the rest of the code, an example `SONARDATA[]` array is examined. Lets assume we received the following data from the 50 byte `AmigoSIP[]`. 

![Figure A3. Perfect case timing diagram for current receive implementation.](image-url)
SONARDATA[] = {0x03,0x5A,0x33,0x07,0x3B,0x28};

The values at SONARDATA[0] and SONARDATA[3] are the sonar numbers (from 1-8 since there are 8 sonar). From this array, it can be noted that the Amigobot only sends information for 2 sonar sensors at a time. So this adds to the unreliability and slow updating of this methodology. The bytes at index 1 and index 2 represent the sonar data value corresponding the sonar number specified in index 0. The same applies to values index 4 and 5, which corresponds to the data associated with the sonar number specified at index 3. The high byte for the data is actually the value at index 2 and 5 respectively.

The next step is to combine the value of the two bytes and then multiply it by a factor of .555 (specified in the Amigobot Technical Manual) to convert the value to millimeters.

fSonardata = (float)((SONARDATA[2]<<8)+SONARDATA[1]);

Then this value is passed into sonardatahelper() to be converted into 3 thresholds for display on the iPAQ (see sonardatahelper() code for more information). Once the data has been converted, it is then assembled into 1 byte and then sent to written to SSPBUF, which will be read by the master PIC once every second.
<table>
<thead>
<tr>
<th>Task #</th>
<th>Task description</th>
<th>Week1</th>
<th>Week2</th>
<th>Week3</th>
<th>Week4</th>
<th>Week5</th>
<th>Week6</th>
<th>Week7</th>
<th>Week8</th>
<th>Week9</th>
<th>Week10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Ordering Parts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Developing Proposal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Experimenting the functionalities of iPAQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Developing Simple Front End Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Research &amp; Develop Methods for using BlueTooth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Assemble PIC board and Robot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Establish BlueTooth Link between iPAQ &amp; converter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Control Robot with PIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Connect PIC to PIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Control Robot with iPAQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Send feedback to iPAQ and display it</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Demonstration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Writing Final Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Developing Website</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>