Creating a Wireless Interface between a Robot and Handheld PDA

Sid Govindan
Advisors: Drew McDermott, Ed Jackson
2003

Abstract

This project involved creating an infrared interface between a ubiquitous handheld computer, the Palm Pilot m500, and a frequently used robot micro-controller, the PIC 16F877. The interface was designed to overcome the programming limitations of micro-controllers and utilize the extensive processing capabilities of the Palm. In this interface, the micro-controller merely serves as a go-between from the Palm to the robot and the robot to the Palm. Here the Palm Pilot does all the processing and storage of data and issues a command to the PIC only when some aspect directly under the PIC's control, such as the robot motor speed, needs adjustment. The Palm Pilot can also request sensor information from the PIC, information that will be received by the PIC from the robot's onboard sensors and sent back via the infrared channel to the Palm Pilot for processing. The interface has been created to emulate the PIC C commands normally used to program the micro-controller. This allows a user already familiar with PIC C to now program the Palm Pilot with similar commands but with the support of a more powerful compiler, larger memory, and faster processing speed. Furthermore, the graphical display on the Palm, the ability to permanently store data, and the availability of easily attached peripherals such as GPS modules, digital cameras, and wireless LAN modules create multiples dimensions of robot programming which could never be realized with a micro-controller alone.

Micro-controllers are used in virtually every electronic device. These multipinned chips serve as the brains of the device and can be quickly programmed to carry out simple on-board tasks. However, their simplicity brings with it several drawbacks, particularly in terms of their memory storage and processing capabilities. For example Microchips PIC 16F877 has only 368 bytes of FLASH memory (less than half a megabyte). The FLASH data memory is retained only as long as power is supplied to the device and there is no straightforward way to extract stored data from the chip and convert it to an understandable form. Because the chip is essentially a piece of hardware, it is built with modules to handle capture/compare/PWM features as well as respond to hardware interrupts. Though such features are essential at the hardware level, combined with
the diminutive memory stack size, they detract from the raw software processing power. While the micro-controller is undoubtedly restricted, it is important to note that it was conceived with the notion of having a device which can do very specific hardware level tasks quickly and reliably.

The recognition of the specificity of the micro-controller as well as the comparably large memory (7.3 MB for the m500), processing power and versatility of commercial available handheld computer such as the Palm, served as the impetus to this project. Restricting the micro-controller to perform only the most basic of hardware tasks, yielding the more intense processing to the handheld, and developing an interface between the two, seemed an intuitive step. Created with the specific goal of being user-friendly and able to hold and process large amounts of data, the handheld picks up where the micro-controller left off. Unlike the micro-controller, the Palm is designed to be an abstraction above the hardware level, relinquishing all the details of hardware processing to the operating system. Such an abstraction can be very appealing to a programmer because no longer does he/she have to be concerned with base limitations and hardware interrupts, but rather can focus on programming more complicated tasks for the device on a purely software level, with the simple recognition of what the hardware can and cannot do.

The interface requires communication between the Palm and the robot. This was achieved by using the IR port on the Palm and connecting the PIC to an Agilent IRDA transceiver. The PIC is physically connected to the robot via a circuit board. The entire interface is diagrammed below (Figure 1):

![Figure 1: The interface between the Palm and the PIC is achieved through wireless infrared. The PIC is physically connected to the robot.](image)

Note that there is only one line between the Palm and the PIC while there are two between the PIC and robot. Both communications are two-way, however, because the Palm and PIC communicate wirelessly, there is only one channel between them. That is to say, the Palm and PIC must receive and transmit commands on the same line while the PIC and robot receive and transmit commands on separate RCV and XMIT lines.

The simplest way to connect a Palm to another device is using the universal connector (see Figure 2) which through a commercially available adapter can be made to transmit serial RS232 commands through an output wire which could be directly connected to and understood by most micro-controllers. However, this universal connector is the connection site for most peripherals including the wireless LAN module. Thus instead of tying up this versatile port of communication, we decided to use the IR port on the Palm (see Figure 2). Normally this port is used to transfer data between Palms or IR-equipped computers.
Wireless communication is challenging because connections can easily be broken and the receiving and transmitting is done through the same line. To counter these problems, complicated handshaking and packetizing protocols are created to ensure that two devices are able to talk to one another and that both will adhere to a prescribed set of rules concerning the sending and receiving of data. The Palm Pilot subscribes to the most prevalent IR protocols, the IRDA IRComm standard. Thus, under normal circumstances, the Palm will only initiate communication with a device that also follows the IRComm standard. Unfortunately, given the complexity of the standard it would be very hard to program a micro-controller to interpret the IRComm standard and respond with the appropriate handshaking procedure. Furthermore, the IRComm sends data in a complicated package scheme which, even if the micro-controller could get through the handshaking, would require intensive time-consuming processing in order to extract the important data.

Virtually every Palm program uses the flexible exchange manager as a means of transmitting IR data because the manager automatically incorporates the IRComm protocol. However, for the purposes of talking directly with the PIC, this method was unsatisfactory for that very reason. After many trials we finally developed a hack that avoided using the exchange manager and instead piped data that would normally be sent through the serial port, directly to the IR port. The data was thus simply a stream of serial bits that underwent IR modulation and were beamed through the IR port.

We then constructed the receiving end of the transmission. After solving various hardware issues and undergoing a mysterious twist of fate involving the discovery of a pre-built IRDA transceiver module, we used an oscilloscope to
verify that beamed IR signals were making their way down the receive line of the transceiver. Connecting the PIC to the receive line as shown in Figure 3, we began programming at the hardware level to get the PIC to begin interpreting the Palms messages. Solving more hardware issues the PIC finally received the letter A, as verified by an integrated LCD display attached to the PIC. From that point, the hardware code was modified to include a serial interrupt handler which would allow the PIC to store messages in a buffer even while it was in the midst of other tasks. The buffer was designed to receive at most 60 bytes worth of new commands, and overwrite the least recent commands if that amount is exceeded. The buffer can be considered a circular data structure where markers indicate where data was last read in and out. This allows the PIC to receive a large amount of data while it is processing other messages, only to come back and process the just received messages.

Figure 3: Circuitry of PIC connected to IRDA module

In order to maximize the integrity of the data stored in the buffer, a start byte and stop byte system of reading in the data was created. In this scheme the Palm always sends a predetermined and distinct start and stop byte to the PIC indicating the start and end of the data packet as diagrammed in Figure 4. The PIC was programmed to only start storing a data packet after receiving the start byte and to stop storing that packet upon receiving the stop byte. This prevented the PIC from storing extraneous bytes from other IR devices that did not follow this specific albeit simple protocol.

Figure 4: The start and stop byte scheme helps insure data integrity.
After verifying that we could send data reliably to the PIC, we experimented with having the PIC send back data to the Palm. Though this direction of communication was anticipated to be used less frequently, it would be required to send back sensor information. By attaching a transmit line to the IRDA module shown in Figure 3, we established this direction of communication. An important note is that the PIC would receive the very data it sent and try to store it in the receive buffer because the transmit diode was located right by the receive one. Even disabling the serial interrupts before transmitting did not solve this problem because the PIC is designed with its own character buffer which stores all data on the serial RCV pin even if no serial interrupt occurs. To overcome this problem, a get character command was issued each time a character was sent out, thereby removing the very character that was sent from the receive buffer. This approach worked well and soon a two way communication was established. The Palm receive program was developed with another hack that piped all data read in by the IR port directly to the serial port receive buffer. From that point, a very similar buffering technique to the one described for the PIC receive was used. The PIC also sends its data with a start and stop byte, however these bytes differ from those used by the Palm in order to prevent either device for thinking its own commands (which may be reflected back) were sent from the other device.

For an interface to be useful it must be seamless. That is to say, it must appear to the user that it allows direct access to the data, or in this case, the robot, without a noticeable go-between. In other words it involves transforming the interface diagrammed in Figure 1 to the one shown below in Figure 5.

![Figure 5: The interface must appear to allow direct communication of the Palm Pilot with the robot, without any middle-man.](image)

We actualized this model by creating an interface which allowed commands analogous to those used in PIC C, to be used directly in the Palm program. For example, to set output high on a pin in PIC C requires the command: output_high(PIN No.). The analogous command in the Palm is PIC_output_high(PIN No.). Analogous commands were created for some of the most useful PIC commands including setting the pulse-width-modulation (PWM) speed, writing entire strings to the LCD display, and delaying for a certain number of milliseconds. The PIC need only have one program which is never varied. This program interprets packets of information organized as shown in Figure 6. The start/stop byte protocol is the same as in Figure 4, however the protocol was modified to send a variable number of parameters separated by commas, specifically, the function to be called and the arguments to that function. For example, to delay
for a given number of milliseconds, a command would include the start byte, the
function number for delay, a comma, the time to delay and the stop byte. Upon
receiving this packet, the PIC was programmed to store all the data between
the start and stop bit then parse till a comma or end of packet, first obtaining
information about which function to call and then the value of the arguments
of that function. To ensure consistency between the sent function id number
and the received function id number a shared header file between the Palm and
PIC was used which listed these values (PalmBot_shared.h).

Figure 6: This revised protocol is vital in creating the seamless interface as it
allows a direct call to a specific function and indicates the arguments to be used
by that function.

The PIC software was modified to also incorporate this packetized way of
sending data. It is much more limited however as the PIC compiler does not
allow for the creation of functions which accept a variable number of arguments
and therefore, is currently only allowed to send two parameters: the function
to be called and the value to be passed to that function. Two parameters were
allowed in anticipation of multiple sensors on the robot. When queried by the
Palm to send sensor information for a particular sensor, the PIC gets the sensor
information for that sensor by calling the Palm specified function in the data
packet. The PIC then sends back an analogous packet including that sensors
identification number and its data, that is, two parameters, the first of which
allows the Palm to know what sensor data that data represents and the second
being that actual data.

The interface was tested by writing simple programs that called the various
functions just as one would do so in Palm C. All of them worked as expected.
Two notes are that the millisecond delay, PIC_delay_ms( time ) in the Palm,
currently emulates the delay in the PIC. That is, the Palm does not process any
commands during this time. It is extremely simple to have the Palm still process
commands after sending the data as shown in the code, however, the delay was
emulated in order to prevent the Palm from sending too many commands (more
than the buffer can hold) while the PIC was delaying as that is currently not
accounted for. Another note is that the Palm has an PIC_lcd_puts( string )
function instead of the PICs lcd_putchar( char ) function. This is obviously an
advancement over the PICs function but does not have a direct analog in the PIC
C library. Likewise the hello message sent by the Palm and the corresponding
here command sent by the PIC have no analogs, but were implemented with the
sole purpose of establishing the connection between the devices and initiating
the interface. The other implemented PIC functions are listed in the header file
PalmBot_pic.h.
To demonstrate the ease of using the interface and the graphical user interface capabilities of the Palm, a simple program, PalmBot\_path\_follower.c was created. This program used a drawing module which was also created and allows the user to draw a series of connected lines by using the stylus pen as shown below in Figure 7. After saving the path for beaming the user can beam the data to the robot. Upon pressing beam, the Palm immediately issues the hello command every 0.1 seconds, until it receives the reply here from the robot. At this time it displays ROBOT FOUND unless the time-out of 1 minute had been exceeded where in it displays ROBOT NOT FOUND. If the robot was found, the Palm computes the angles between the lines and determines for how long the robot should continue to go straight, when it should turn, which direction it should turn and for how long it should turn in order to stick close to the path. Such math processing would have been impossible for the PIC. Furthermore, every 0.1 seconds, the Palm queries the PIC to send back the sensor information. Fitted with a distance sensor in the front, the robot sends back data that translates to the distance to the nearest object. The Palm sees if this distance is below the threshold value and stops the robot in its tracks if it needs too. A picture of the palm bot robot is included below (Figure 8).

![Figure 7: The graphical interface allows the user to draw a simple path for the robot to traverse.](image-url)
The potential of this interface is very promising. If fully utilized, I believe it can easily out-perform any currently micro-controller based robot. By virtually replacing a limited controller with a powerful yet portable computer, the opportunity for smarter, quicker, expandable and more accurate processing opens the door to a new era of quick and simple electronic device programming.