Development of Software for Mobile Robot Control over a Radio Frequency Communications Link

Matt Lister; Elizabethtown College; Elizabethtown, PA
Joseph T. Wunderlich; Elizabethtown College; Elizabethtown, PA

Keywords: RF communications, mobile robots, control software

ABSTRACT

In the spring of 2001, work began on a mobile robot for an advanced computer engineering class. One of the key parts to the success of this robot would be the ability for it to be controlled from a computer terminal. In order to make this feasible, the computer would communicate to the robot through a wireless connection. The robot's on-board computer would perform any computations needed, so the robot didn't need to send any information to the controlling computer. The communication link was designated to be one way, from the computer to the robot. For reasons including cost efficiency, ease of use, and range, RF communication was chosen.

Using two development boards from Linx Technologies, we had the capability of communicating through the computers serial port. The boards boasted a line of sight range of about 1000 feet. The board acts as nothing more then a transmitter and receiver, so any error correction must be dealt with in the software. Often errors come from interference created by surrounding RF devices, and appear as incorrect bits in the data stream.

In order to handle such errors, methods involving redundancy were used. When the stream of information is sent, it is initialized by a start byte. This is followed by the commands, each of which is repeated three times followed by a stop byte. On the receiving end, the software waits for the start byte and then stores each byte that follows. They are then compared with one another and determined to be either clean data or an error. If the data is clean, the robot then executes the commands.

By monitoring the data received by the receiver we were able to see the need for such software. Upon powering up the board, miscellaneous bits are received; and if there is no error correction, the robot could misinterpret these for a command. The boards also received stray bytes while idle, so the software would also eliminate this. Software development was started and initially tested over a crossover cable connecting the two serial ports. This allowed us to ensure that the software was working with out seeing any errors at first. Finally the software was used with the RF boards and seen to be functional.

This paper will explore, in depth, the methods used by software to correct errors that may develop in RF communication. The materials used will be discussed in greater detail along with the creation of the software. Experimental results will show the methods to be precise, and fit the design application needs.

1. INTRODUCTION

The communication programs for use on both the robot and the base computers were written using Visual C++[1]. This software needed to be able to send and receive data as well as control external hardware. User input is a critical part of the robots operation, so this needed to be incorporated in with the software[2]. In order to write software to work correctly with the hardware, a set of protocols was first agreed upon. These include what the robot would be capable of doing, what information would need to be sent to external control circuitry, and what input would be read into the robot's computer.

2. SOFTWARE DESIGN

Initially, the operation of the robot was to be quite simple; it was to be able to move forward and backward and turn left or right. Keeping these commands separate (i.e. not turning right while moving forward) simplified distance calculations, and with the use of optical encoders attached to the wheels, this data was easily gathered. The direction heading was measured with the use of a digital compass that output the direction in a binary number.

2.1. Robot Control

The user controls robot movement in one of two ways. The first possibility is a real-time instruction that tells the robot to execute a particular command until another is given. Essentially, the operation is similar to that of a remote controlled car. The other possibility is a list of commands for to be executed in a particular order. The data contained in this list includes the direction of which the robot is to go, the length at which the robot is to execute the command, and the speed at which the command is to be executed. The direction commands can be one of five choices; forward, reverse, left, right, and stop. The distance command is either a distance in feet to travel, or an angle in degrees at which to turn. This command is limited to a number of 180 or less to

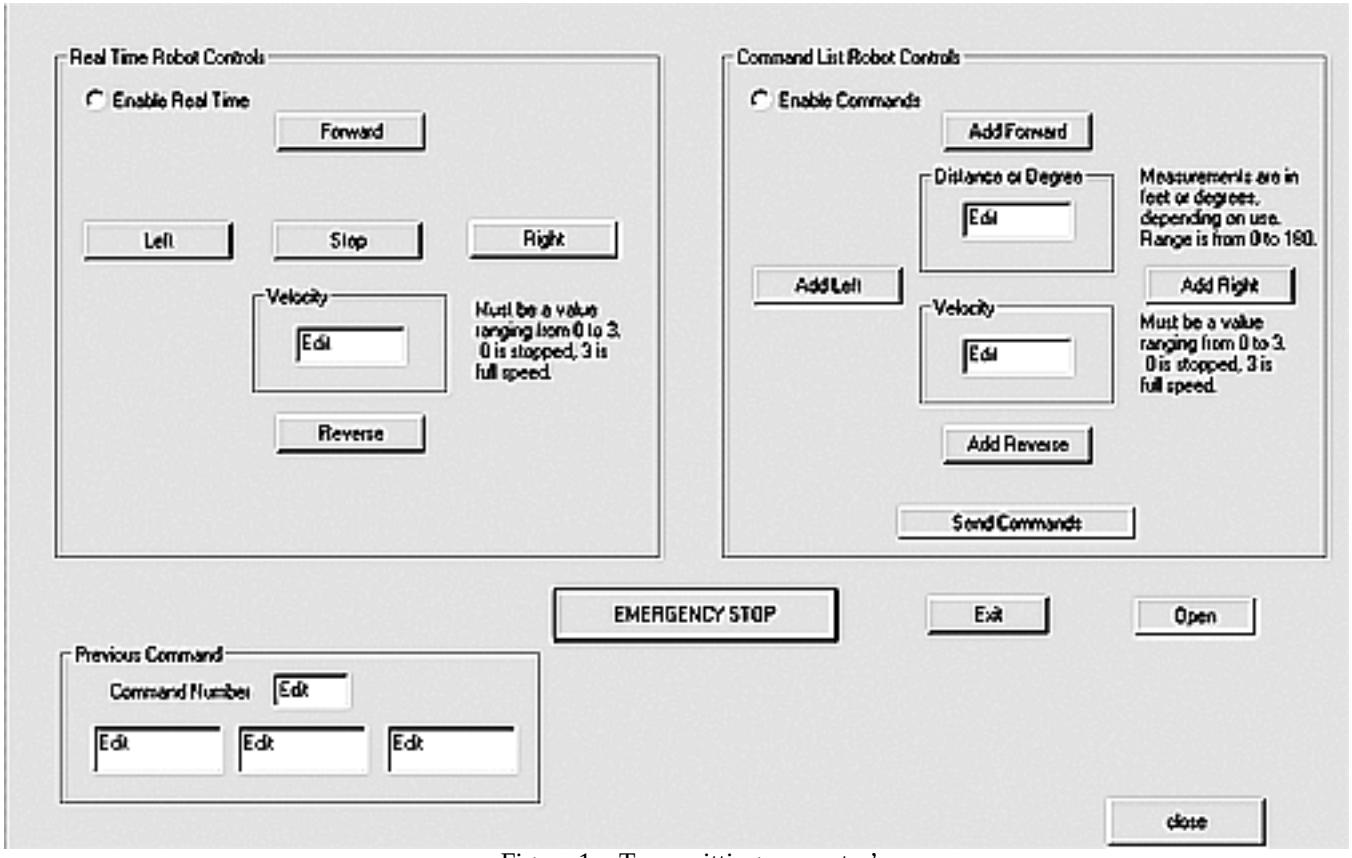

Figure 1 – Transmitting computer's user

accommodate an 8-bit or smaller number. Since 180 degrees is half a turn and less then the 255 units available in an 8-bit number, it was used as the limiting factor. Longer distances are simply accomplished by a repetition of commands until the desired distance or direction is achieved. The speed is to be one of four settings being stop, slow, medium or fast.

The user interface is set up for ease of use, as it resembled the directional pads of a game controller as seen in figure 1. The user has selection boxes were either real-time mode or command list mode can be chosen. On the command list side, an add command button will add a command to the list of commands to be sent the robot. The "Send Commands" button will send the list of commands to the robot and clear out the current command list. The real-time side executes each command upon pressing a directional button. If the "Forward" button is pressed then a forward command is sent, along with its speed, to the robot

2.2. Communication Between Computer and Robot

The c++ code for sending receiving data through a serial port was found and broken down for use in the controlling software. The basics of this code were driven by an interrupt type function. Upon detecting a byte through the serial port, the software would store that byte, and then perform the necessary actions upon it [3]. It was this part that was incorporated into the code.

The communication path chosen for the robot were RF development boards. These boards allowed bytes to be transmitted from one computer to another without a hard wired connection. Once a byte is sent to the serial port, it is processed by the hardware on the development board and transmitted to the receiver board, which then processes the data and sends it to the robots serial port, the computer never sees any difference between this type of connection and a hard wired connection [4]. Currently, only one-way communication is available, that is the base computer sends and the robot receives.

This communication hardware has a range of 1000 feet, but is susceptible to some noise [4]. Occasionally erroneous bytes are detected that have no meaning. The hardware does not see these as problematic data and sends it to the robots serial port along with the acceptable data. The software on the robot must distinguish between what is usable data and what is garbage. Through methods of error detection and correction, this is accomplished.

Redundancy was chosen as the method of error correction for this problem. By sending out the same data repeated times, this is a simple yet effective way to assure that the robot receives what it is supposed to [5]. Also added into the redundancy for added protection is a set of start, stop, and clear bytes that would trigger the robots receiving process.

The transmitting program on the base computer was created so that a packet of information containing all

necessary instructions was sent to the robot. This packet includes the start and stop bytes, the command, direction, and speed bytes, and a clear byte between each command. Each command is sent three times to accomplish the redundancy. Figure 2 shows what this packet contains.

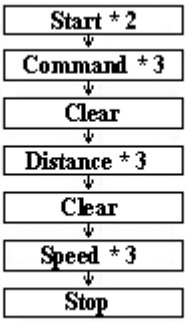

Figure 2 – Transmission packet

The robot's computer consists of a miniature 486 motherboard running Windows 95. Upon turning on of the robot, the receiving program starts and waits for data from the serial port. Once data is received the processing is started and the commands are executed.

The receiving program sits idle while waiting for the packet of instructions. Once it sees the start byte it begins storing the following byte into their correct variables. The software uses the clear byte incase an error occurred and not all commands were received properly in a cycle of redundancy. The program expects to receive three bytes of data for each command; if it records only two bytes then receives the clear byte, it know that an error occurred and it stores the next bytes in the following commands. Once the stop byte has been received, the error correction process is started.

Each instruction is an 8 bit binary number, and each set of these instructions is compared to correct any errors that may have been picked up through the transmission process. This is done by comparing each bit of the instruction to the same bit of the other two instructions. If two of the three match then that bit is considered correct and the next bit is processed. Once the software has gone through each of the three instruction sets the commands are executed and the process is started again. Figure 3 shows the effect of a 1 being transmitted that picks up a zero bit instead. The 1 is transmitted three times by the base computer, however the receiver detected two 1's and a zero. This data is then processed and the correct bit is stored as the proper command. This process is performed on each bit in the command byte that is sent out to the robot. Figure 4 shows the same process only with a 0 being sent three times and 1 being detected as an error.

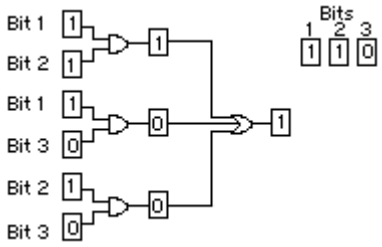

Figure 3 – Correcting data that was supposed to be 1

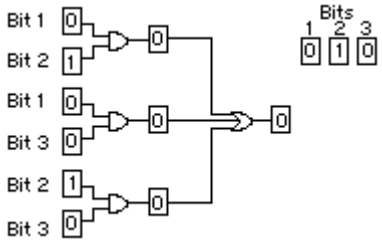

Figure 4 – Correcting data that was supposed to be 0

Figure 5 shows the basic logic equations for completing these operations, resulting in a correct instruction byte.

Byte1 AND Byte2 = Byte12

Byte1 AND Byte3 = Byte13

Byte2 AND Byte3 = Byte23

Byte12 OR Byte13 OR Byte23 = Instruction

Figure 5 – Logic operation performed on received bytes

In order for the robot to know what to do, the commands are sent out through the parallel port of the robot's on-board computer. Several eight-bit pieces of information are sent out and interpreted by the robots motor drive circuitry. When the robot has completed the task, a signal is sent back in through the parallel port and the next command can be executed.

2.3. Output to Motor Control

The motor control protocols were designed to accommodate the onboard computer's 8-bit parallel output [6]. It was decided that by using 2 bits as address bits and the remaining 6 as data bits, the parallel port could simulate a 24-bit output, as seen in figure 6. The first two bits let the hardware know which set of data was being output; then the data was latched into the hardware until all 24 bits had been read and the motor controls would respond appropriately. The software had the task of translating the commands into a string of 24 bits, which was then sent to the parallel port.

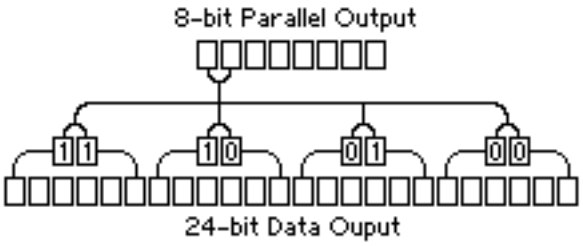

Figure 6 – Converting 8-bit data to 24-bit data

3. Test Procedures

As the software was being created, numerous scenarios were tested. Initially the behavior of the RF development board had to be monitored. To accomplish this, a simple program was written that would display whatever the receiver board detected. This is where random bytes were noticed. Often these were the result of other RF devices operating at similar frequencies such as portable telephones and a wireless camera that was to be used on the same robot.

Communication between the base computer and the robot's computer was tested with a crossover serial cable connected between the computers' two serial ports. This functioned as a medium that allowed for the testing of motor control protocols without random bytes interfering. Once all bugs were worked out and the control protocols were functioning properly, the error correction was added to the software. It was first tested using the cable; and errors were intentionally put through.

Finally the RF boards were connected in place of the crossover cable and the software was again tested. Testing consisted of operating the robot's motors with the various commands at different distances from the base computer.

4. Conclusion

At the time of completion of the control software, not all of the mechanical aspects of the robot had been finished. This prevented any field-testing of the robots operation. The test results that came from testing the software with partially working components showed that the programs successfully did their job. While the software was running, the robot never performed any unexpected actions.

Upon completion of the robot, field-testing will be conducted to verify full functionality of the robot. Future software modifications may include further investigation of various error and detection methods.

REFERENCES

1. Gurewich, Nathan, "Sam's Teach Yourself Visual C++ 5", Sam's Publishing, 1997.
2. Kruglinski, David, "Programming Microsoft Visual C++", Microsoft Press, 1998.
3. Bergsman, Paul, "Controlling the World With Your PC", HighText, 1994.
4. LINX Technologies, "HP Series-II Receiver and Transmitter Design Guide", 1999.
5. LINX Technologies, "Considerations for Sending Data With the HP Series", Application Note AN-00160, 1998.
6. Nelson, Mark, "Serial Communications Developer's Guide", IDG Books Worldwide, Inc., 2000.

BIOGRAPHIES

Matt Lister
10262 Old Cordova Rd
Easton, MD 21601

E-mail: listerm@etown.edu

Matt Lister is a senior at Elizabethtown College where he is majoring in computer engineering. Prior to college, Matt graduated from Easton High School in 1998. He has completed several robotics projects including a multifunctional mobile robot, a simple task mobile robot, and simulation of mobile robot movement.

Joseph T. Wunderlich
Elizabethtown College
One Alpha Drive
Elizabethtown, PA, 17022

E-mail: **wunderjt@etown.edu**

Joseph Wunderlich is an Assistant Professor of Computer Science and Computer Engineering at Elizabethtown College. He came to Elizabethtown from Purdue University where he was an Assistant Professor of Electrical and Computer Engineering Technology. Prior to that he worked as a researcher and hardware development engineer for IBM on large-scale multiprocessor computer systems. Dr. Wunderlich received his Ph.D. in Electrical and Computer Engineering from the University of Delaware, his Masters in Engineering Science/Computer Design from The Pennsylvania State University, and his BS in Engineering from the University of Texas at Austin. Dr. Wunderlich is a member of IEEE and ASEE.