

# Flight Control

## A Communications System for an RC Helicopter

Whether you are looking to control an RC helicopter or a full-sized car, this system will work for you. By facilitating communication between a base station and a vehicle, the system enables an operator to process commands and receive data.

Anyone who lives near a model airplane club can attest to two things: the annoying buzzing of nitro engines and the amazing abilities of some of the scale models. Along with planes, model helicopters are sometimes encountered at these clubs. They are more maneuverable than their larger brethren, capable of performing aerobatic maneuvers and inverted flight. A characteristic that scale-model and full-sized helicopters share is their difficulty to pilot, especially for inexperienced technology students. So, when it came time to develop our final project in our technology program at Camosun College, we chose to work on an autonomous, self-balancing model helicopter.

An important part of this project involved developing a communications system for sending commands to the helicopter and receiving status information in return. Using simple, readily available hardware and extensive software, we successfully implemented the communications system to keep the base station connected to the helicopter.

In this article, we'll describe our communications system. It comprises firmware and a communications protocol.

### APPLICATIONS

Our communications system was originally designed to control an RC helicopter. The system uses software, firmware, and a communications protocol to send commands to the helicopter and receive information from it. The command packets and data packets are

transmitted between a base station connected to a laptop computer and a receiving module on the helicopter. Sensor data and user commands control an array of servo motors. With the right modification of servos and sensors, this concept for control could easily be implemented to aid in the control of other types of RC or full-scale vehicles, including boats, planes, or cars.

Who can use a system like ours? A search and rescue team could send out multiple autonomous helicopters equipped with cameras. The smaller-scale autonomous helicopters would have a better chance of finding people in distress than a single full-scale helicopter. Also, the smaller-scale autonomous helicopter would come with a smaller price tag than the full-scale version. But the applications for this communications system are not limited to search and rescue. This system could be used to help remove human involvement with virtually any vehicle's operation. You can use the system to control vehicles for cinematography, aerial mapping, law enforcement, surveillance,

inspection, and more.

### SYSTEM OVERVIEW

The communications system is designed to transfer data between the base station and the target device. Our target device is an electric RC helicopter. The communications system consists of PC software at the base station, hardware on the base station, hardware and firmware on the helicopter, and a communications protocol to tie everything together.

The PC software consists of three subsections: the communications class library, the indicator user control library, and the Google Maps user control (see Photo 1). The communications library links the GUI to the helicopter. It is built around a serial port connection and it provides a simple interface for

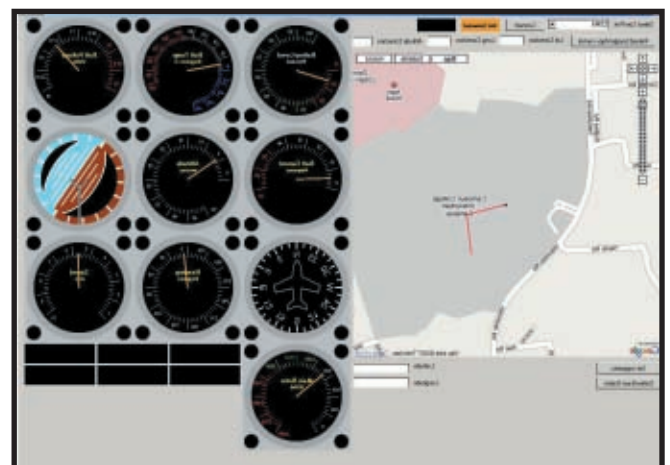


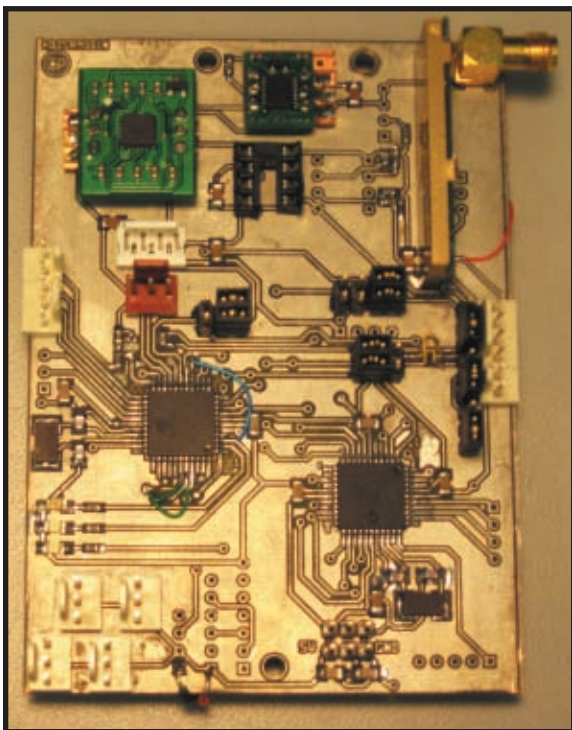
Photo 1—This is the graphical user interface for our system. The customizable indicators enable us to keep track of any data received by the base station.

the GUI coder. The indicators library shows the user telemetry data from the helicopter. It can display customizable dials, a compass control, an artificial horizon, and indicator lights. The overall indicator design mimics a helicopter or airplane instrument panel. The Google Maps user control defines waypoints for the helicopter and shows its position.

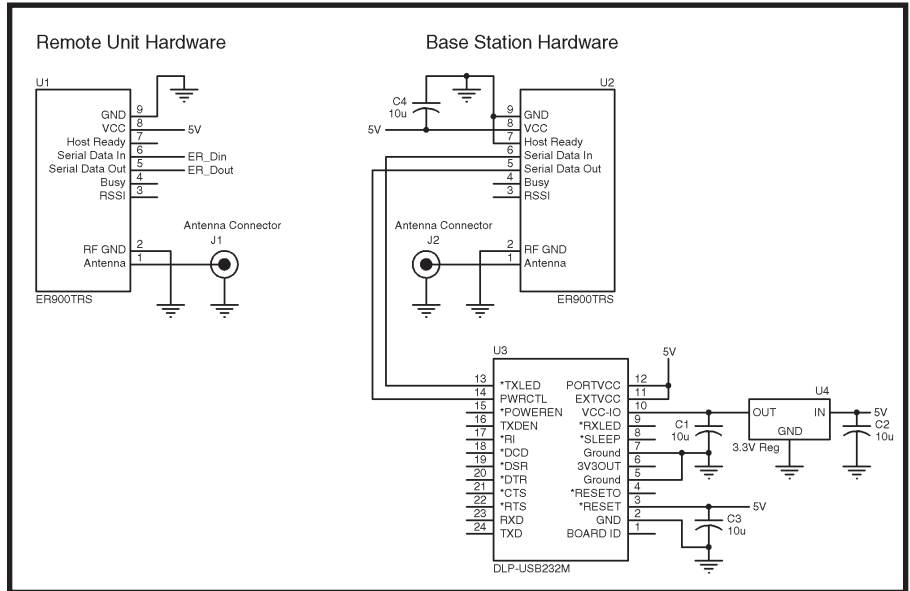
For base station hardware, we used an antenna, a USB-to-Serial module, and a transceiver module mounted to a PCB. The hardware on the helicopter for the communications system consists of an antenna, a transceiver module, and a Microchip Technology dsPIC30F3011 microcontroller. The communications protocol is a rule set that governs the transmission of data between the base station and the helicopter.

## HARDWARE

We knew from the start that the most difficult aspect of this project would be the software. For this reason, we used simple pieces of proven hardware to get the job done so we could have more time to focus on the software. The communications system hardware on the helicopter is virtually the same as the hardware in the base station.



**Photo 2**—This is the main control board of our helicopter. Visible at the top right of the photo are the radio module and the antenna connector.



**Figure 1**—This is a schematic of the base station and remote unit hardware.

We used a pair of Low Power Radio Solutions ER900 TRS 900-MHz transceiver modules to send data between the base station and the helicopter (see Figure 1). We chose the modules for their simple serial interface, their operation in an unlicensed band, and their ability to take care of error handling. The transceiver modules are mounted to PCBs, which we designed and manufactured at Camosun College (see Photo 2). The base station board, which holds one of the transceiver modules, also holds a USB-to-serial converter. The USB-to-serial converter is the link between the transceiver module and the software on the PC.

The helicopter radio module is coupled to a 4" omni-directional whip antenna and interfaced to the USART of the onboard microcontroller. This antenna works well on the helicopter because it is small and fairly lightweight. For a base station antenna, we started with a second 4" omni-directional whip antenna. The signal was somewhat limited using this antenna, so we chose to upgrade to a 12-dBm, 4' long, nine-element Yagi-Uda antenna. This gave us

increased range and better signal quality. Plus, we got it for free because one of our instructors purchased it on the condition that we give it to him (for one of his evil schemes) when we were done.

## PERFORMANCE EVALUATION

We tested the communications system and determined the data rate was 517 bps. We calculated this with a measured time per byte of 17.4 ms, which is close to the datasheet's minimum specified value of 15.978 ms. The latency of the packets varied depending on packet length, with an average delay of 46.75 ms. We measured this delay as the time from when a set of outgoing data is loaded into the radio's buffer to when an acknowledgement is received. Our latency measurement does not take into account the unpredictable delay caused by the non-real-time nature of Windows XP.

The message throughput can vary depending on which error rate is considered acceptable. We made two measurements. At 3.03 packets per second, we determined the error rate was 2.05%. At 4.08 packets per second, the error rate was 2.33%. The two packet rates were used to illustrate the trade-off between the packet throughput and the number of errors that occur. As packet throughput increases, the error rate also increases.

## PROTOCOL

The protocol we developed is a set of rules that enables us to maintain

Header		Length	Data			Checksum		Footer	
0xA5	0x5A	0x03	0x54	0x45	0x32	0x00	0xCE	0xCC	0x33
—	—	—	Family	Command	Payload	Sum of data and length		—	—

**Table 1**—This is an example of a packet sent between the base station and the helicopter. This packet, when received and decoded by the helicopter, will set the speed of the engine.

reliable communication between the base station and the helicopter. Work on the communications protocol began even before the official start date of our project. This was a large task, which we needed to complete quickly so we could use it to test other aspects of the helicopter's operation. The full code communications protocol is available on the *Circuit Cellar* FTP site.

The communications protocol is fairly simple in the sense that we did not incorporate any error checking. This was possible because the radio modules we used have some good error-checking routines of their own. There are several bytes within the protocol that have specific tasks. The header bytes are used to "wake-up" the receiver so it can begin recording and decoding the incoming transmission. The length byte tells the receiving station how many bytes are in the data portion of the packet and includes the command group, command, and payload portions. The command group byte tells the receiving station (either the helicopter or the base station) what type of command is being sent. For example, it might be a testing/tuning command, in which case the command group byte would be 0x54. We included a command group to make decoding simpler and better documented. Another programmer looking at the state machine can easily go to the group in question and have fewer commands to sift through.

We use command bytes that make sense. For example, the hexadecimal number 0x50 is an ASCII character "P." This is the character used in the command to change the pitch servo pulse width ("P" for pitch). However, "P" is also the

command byte for the base station to request a preflight packet ("P" for preflight). The command group byte differentiates the two. The command byte then tells the receiving station exactly what command is sent. In Table 1, 0x45 indicates an engine speed change. (Hexadecimal number 0x45 is an ASCII character "E," for engine speed.) The checksum is used to verify that the data received is the data that the transmitter intended to send. It is defined as the two least significant bytes of the sum of all the bytes in the length and data portions of the packet. The footer simply indicates the end of a transmission.

In our master/slave system, the base station is the master and the helicopter is the slave. This has a number of implications. The base station initiates all communications, and the helicopter doesn't transmit unsolicited data. The only exceptions are error messages, because

the base station can't know when an onboard error will occur. To ensure that all transmissions from the base station are received, each one is acknowledged by the helicopter in one of two ways. Command packets are acknowledged by the helicopter by transmitting the entire packet, plus an ACK byte (0x06, which comes after the length byte). Information requests are acknowledged by simply sending the information requested. If an error is detected in the packet, the packet is ignored and no ACK is sent. It is the master's responsibility to retransmit any commands or information requests that are not acknowledged.

Table 2 is an overview of the complete communications protocol. For a detailed description of all packets and payloads, refer to the Communications Protocol document posted on the *Circuit Cellar* FTP site.

## FIRMWARE

The data sent via our protocol has to be decoded and used once it is received by the helicopter. This task is performed by our firmware program. The firmware handling our communications is loaded onto a dsPIC30F3011 on the helicopter. We chose this microcon-

troller because it has powerful processing abilities, which we use in our helicopter for control calculations, and because we happened to have a stash available to us at school.

The firmware, which is essentially an onboard packet parser, comprises two separate state machines. The first state machine checks the packet header and data length, copies the data into a buffer, verifies that the checksum is correct, and checks if the footer is valid. If everything is fine, it sets a valid data flag. This starts up the second state machine, which parses the valid data packet. It checks the first byte to determine which command "family" the command belongs to and

Group description	Group byte	Command	Command description
Testing/tuning	0x54	0x45	Engine speed adjust
	0x54	0x50	Pitch servo adjust
	0x54	0x52	Roll servo adjust
	0x54	0x43	Collective servo adjust
	0x54	0x51	Anti-torque servo adjust
	0x54	0x66	Set operations mode
	0x54	0xDD	General-purpose data dump
Flight operations	0x46	0x45	Engage engine
	0x46	0x48	Hover
	0x46	0x43	GPS Correction factor
	0x46	0x47	Go to GPS coordinates
	0x46	0x52	Return to base
	0x46	0x50	Request pre-flight packet
	0x46	0x4D	Discreet movement
	0x46	0x49	Request for information
Telemetry data	0x74	0x4C	Location
	0x74	0x48	Heading/speed/altitude
	0x74	0x5A	Attitude
	0x74	0x42	Battery status
	0x74	0x45	Error report
	0x74	0x50	Preflight packet
	0x74	0x52	Rotor RPM

**Table 2**—This is a listing of the communications protocol used in our system. The protocol is flexible and it can expand to up to 65,535 commands, which would be broken into 256 groups. As an added bonus, the code is easily modified to accommodate the extra commands.

# Pololu Robotics & Electronics



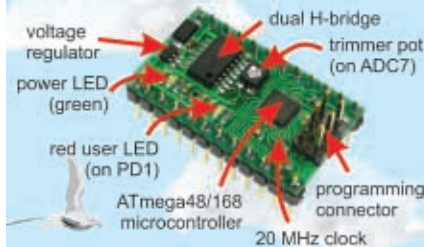
**Robot Kits**  
Line followers  
Robot arms  
Hexapods  
Chassis

**Mechanical Components**  
Gearboxes, servos  
Wheels, ball casters



**Motion Control**  
Servo controllers  
Motor controllers

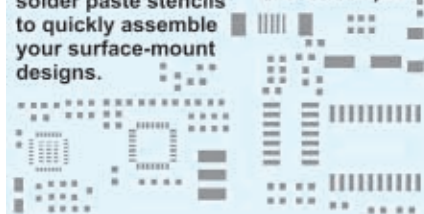
## Robot Controllers



## Solder Paste Stencils

Use our low-cost solder paste stencils to quickly assemble your surface-mount designs.

**From \$25**



## Custom Laser Cutting

**From \$25**



Cut your own custom chassis, front panels, and more!

**1-877-7-POLOLU**  
**www.pololu.com**

6000 S. Eastern Ave. 12D, Las Vegas, NV 89119

checks the actual command. It then determines which action is appropriate to take based on the command. Table 1 shows a complete example of an engine speed packet.

## SOFTWARE

The PC software is designed to keep you in control of the helicopter and informed of its status. It has three main components: the communications class library, the indicators user control library, and the Google Maps user control. All parts of this software are coded in C# using the .NET framework and the Microsoft Visual Studio 2005 IDE.

The communications library is used to link the GUI to the helicopter. It is built around a serial port connection. The library provides a simple interface for a GUI coder to use the communications protocol. It enables all packets to be sent by a simple function call and each received packet raises an event for the GUI to handle. All other functionality, such as error checking and packet parsing, is dealt with internally by the communications class library. The library is centered on a .NET serial port object and a high-speed timer. Each time the timer event is raised, the library checks the serial port buffer to see if its contents are a valid packet or if there is a transmission error. The serial port object raises an event when bytes are received, but we found the event was not raised reliably on every byte. Additionally, using the event forced us to pass the event data across threads. This is because the serial port's events are on a separate thread from the GUI. Because the PC software initiates most communications with the helicopter, it is essential to receive the helicopter's acknowledge packet to verify that the desired data has gone out. For this reason, each time a packet is sent, a timer is started. If the timer expires before the expected response is received, a timeout event is sent to the GUI.

The indicators library is used to show the user telemetry data from the helicopter. It is highly customizable: we use it to create the indicators from a real aircraft control panel (i.e., a compass, an artificial horizon, and indicator lights). All user controls are scalable in size, enabling them to be resized on a form

size change or other events. Dial indicators can be customized easily for many different applications. The numeric range, labels, number of ticks, color ranges, and other aspects can be modified to suit any application. The left-most dial in Photo 1 is an example of a temperature dial that has been set up with a  $-40^{\circ}$  to  $100^{\circ}\text{C}$  range and warning bands for "under" and "over" temperature.

We drew all of the indicators using the graphics device interface (GDI). We started by making the gauge class, which inherits the basic Windows user control and implements all features that are common to center dial, artificial horizon, and compass classes. In turn, each of the classes inherits the gauge class. An important thing we learned was the value of using double buffering to prevent flickering of the things drawn using GDI.<sup>[1]</sup>

The Google Maps user control shows the position of the helicopter with GPS coordinates received via the communications protocol. It is also used to define way points for the helicopter. The user control requires an Internet connection. It is based on a .NET web browser control that loads Power Map, a JavaScript page containing interface functions to use with Google Maps. The JavaScript on this page forms a wrapper around the Google Maps API. It is what the map control interfaces to. Each time a GPS packet is received from the helicopter, we convert the packet's contents from the standard NMEA string format ([www.geoaps.com/NMEA.htm](http://www.geoaps.com/NMEA.htm)) to floating-point degrees. The map can be easily centered on the coordinates or a marker can be placed at this point. At startup, the Google Maps user control loads the Power Maps JavaScript. If an Internet connection is present, the Google Maps interface will load and a map will be shown. Each JavaScript function in Power Map is called by the method `HtmlDocument.InvokeScript` method.<sup>[2]</sup>

## IMPROVEMENTS

If you have the time and money, we would recommend adding error (parity and redundancy) checking to the communications protocol. We were limited on time, so we skipped this aspect of the project because our radio modules had error checking built in. Adding

error checking to the communications protocol would enable the protocol to be used on any radio module. Users of the communications system would not have to worry about finding a radio module with error checking built in. They would be able to find any radio module and plug it in.

Another important improvement would be to modify the protocol to allow for multiple slave devices, enabling the multiple helicopter scenario described in the introduction. This would require changing the protocol packets and firmware to include an address byte (or bytes).

The Google Maps control is coded specifically for our helicopter project, so another recommended improvement would be to make the Google Maps control more general-purpose and extend its functionality. There are several functions written in the JavaScript that are not yet implemented in the .NET control. One example is importing way points from an XML file.

We also have some recommendations to improve the indicators class. Currently, it is made to simulate an aircraft control panel. It would be beneficial to make it more general-purpose as far as how it looks. Also, adding more controls, such as thermometers, seven-segment displays, or redline range events, would add to the functionality. 📌

*Michael Ghazi's (michael@ghazi.ca) six years as a naval communicator in the Canadian Forces prompted him to undertake the task of designing the RF system for the helicopter. After completing his studies at Camosun College, he will be relocating to the greater Toronto area to find work in his field.*

*Stefan Kaban's (snkaban@gmail.com) interest in helicopters was piqued during a co-op job with the B.C. Ministry of Forests, where he was regularly exposed to helicopter operations. He plans to complete his degree in Electrical Engineering at the University of Victoria.*

*Scott Morken (scottm361@gmail.com) is a graduate of the Computer Engineering Technology program at Camosun*

*College. He discovered his new favorite programming language while working for two co-op terms as a C# developer. Scott has also worked on many other hobby software projects. Updated versions of some of the project software are available on his web site (www.red79.net/Projects.html).*

*Carl Philippsen (carlphilippsen@hotmail.com) is a graduate of the Electronics Technician Common Core program at North Island College and the Electronics Engineering Technology program at Camosun College. He is interested in a career in biomedical engineering and plans to further his education at the University of Victoria.*

*Kyle Wong (kindawong@gmail.com) worked as an electronics technologist for several years. He plans to attend the University of Victoria and enroll in the Electronics Engineering degree program.*

## PROJECT FILES

To download code, go to [ftp://ftp.circuitcellar.com/pub/Circuit\\_Cellar/2008/212](ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2008/212).

## REFERENCES

- [1] G. Schmidt, "Don't Flicker! Double Buffer!," The Code Project, 2007, [www.codeproject.com/csharp/DoubleBuffering.asp](http://www.codeproject.com/csharp/DoubleBuffering.asp).
- [2] Microsoft Developer Network, "HtmlDocument.InvokeScript Method (String)," [http://msdn2.microsoft.com/en-us/library/be9zzz62\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/be9zzz62(VS.80).aspx).

## RESOURCE

R. Scammell, Power Map, <http://hobbiton.thisside.net/advmap.html>.

## SOURCES

**ER900 TRS 900-MHz Transceiver modules**

Low Power Radio Solutions  
[www.lprs.co.uk](http://www.lprs.co.uk)

**dsPIC30F3011 Microcontroller**  
Microchip Technology, Inc.  
[www.microchip.com](http://www.microchip.com)

**Visual Studio 2005 IDE**  
Microsoft Corp.  
[www.microsoft.com](http://www.microsoft.com)

# Windows CE based Touch Controller

# CUWIN3500



- 7" wide color TFT display
- 800 x 480 resolutions, 260K colors
- Touch panel
- SD card & Ethernet supports
- RS232 x 2 / RS485 x 1 or RS232 x 3
- Speaker with Audio output
- Real time clock (Battery backup)
- Visual Basic, EVC supports
- USB I/F (active sync)
- Keyboard or Mouse supports
- ARM9 32bit 266MHz processor
- Windows CE 5.0
- 64MB FLASH, 64MB SDRAM
- Size(WDH) : 220 x 150 x 50 mm

CUWIN3500  
**\$599 / Qty. 1**

**COMFILE** TECHNOLOGY [www.comfiletech.com](http://www.comfiletech.com)  
Toll-Free: 1.888.928.2562