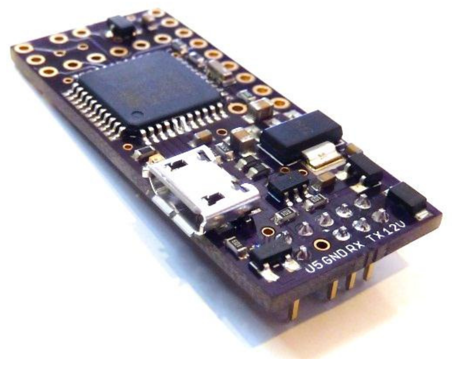
MiruPCB

- the solution for a painless implementation of the MiruMod
- small size 0.7” x 1.6” and light weight 3.5 g
- basic configuration does not require wires
- integrated Logic Level Converter and connector for drone
- built in USB connection for software updates and SETUP mode
- auto detection of SETUP mode when USB port is plugged into computer
- serial bridge function for drone
- convenient connector points for GPS, I2C and remote activity LED

Hardware



The MiruPCB facilitates the implementation of the MiruMod for the Parrot AR.drone. It combines an ATMEGA 32U4 processor with a logic level converter to communicate with the drone, a RC-receiver interface, a GPS 3.3/5V interface and a 5V I2C interface, as well as an USB device. The RC-receiver stacks below or above MiruPCB, there are no wires required to control the drone by RC. The board is programmable with the Arduino IDE.

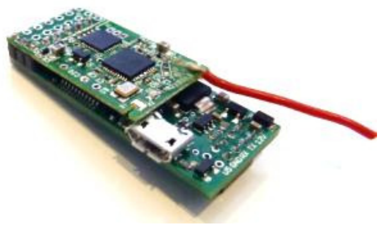
Software

Apart from the ‘standard’ mod, which uses an Arduino Nano or ProMini with an Atmega328p processor, the MiruPCB employs a different ATMEL processor, the ATMEGA32U4. It has a USB port integrated and eliminates the use of a FTDI chip to program it. The MiruMOD sketch rx2atp.c, from revision 0.20 on, supports the MiruPCB. Just select Micro instead of Nano or ProMini in the Arduino IDE. The new sketch has an USB driver and accommodates the different I/O structure of the 32U4 processor.

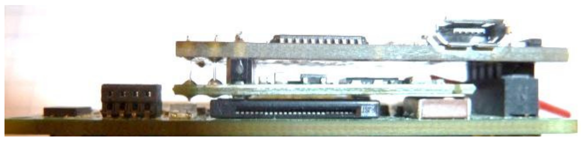
On MiruPCB the software auto detects SETUP mode. It is entered when a USB connection can be established after boot. Once SETUP mode is entered (5 flashes) it monitors the USB connection for a CTRL-B and the drone interface for activity. Activity on the drone interface starts a terminal bridge between the USB port and the drone serial port. This way one can monitor the drone boot messages and start a shell on the drone through the terminal emulator program on the PC. This feature is not available for the ‘standard’ mod processors, due to the hardware structure of the Arduino boards used.

The MiruPCB comes preloaded with the sketch available at manufacturing. Once you make a USB connection to a PC, it will show up as a FTDI board, which the Arduino IDE has a driver for. To load a new sketch select Micro in the Arduino IDE and the serial port detected. If for some reason you have to reinstall the boot loader, check http://arduino.cc/en/Guide/ArduinoLeonardoMicro#toc8 to understand how to download a sketch after a new boot loader has been installed. It is more complicated because installation of the ‘standard’ boot loader also installs a sketch (blink) that does not support the USB interface. In a nutshell, once the new boot loader is installed you have to hold the board in RESET until the compile is finished and the IDE starts searching for the interface. After the sketch has been loaded, the board will show up as a serial port again.

RC-receiver installation



The picture on the right shows the intended mounting of the RC-receiver above MiruPCB. I am using a stripped ORANGERX for demonstration purposes.



On a drone 2 it is possible to mount the receiver underneath the MiruPCB; the picture on the left shows the mount on a drone 2 motherboard.

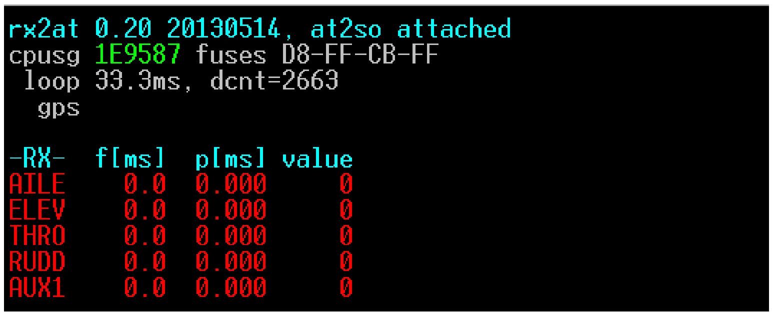
Building steps

1) Connect MiruPCB to a PC. Install the FTDI driver from the Arduino IDE if necessary. After that you should be able to run a terminal emulator to communicate with the board (see SETUP mode for instructions).
2) Modify the sketch for your particular setup and load it into the MiruPCB. Keep in mind the Arduino IDE and the terminal emulator cannot share the USB link to the board.
3) Mount the RC-receiver, run SETUP mode again and make sure everything is working.
4) If you plan to use a GPS, barometer or compass, connect it temporarily, run SETUP mode and check if everything is working.
5) Mount MiruPCB and other equipment, if any, in drone. Use a 3M double sided tape to secure the MiruPCB to the drone.
6) Power up the drone…

SETUP mode

Connect the MiruPCB to a PC, the drone does not need to be connected, if it is, it should not be powered. Start HyperTerminal or other equivalent terminal emulator program, select the right COM port, the communication parameters are 115200 baud, 8 bits, 1 stop, NO parity and flow control NONE. The communication parameters are not that important, the USB connection does not use them, but if you are using an Arduino you need to set them because it is communicating through RS232.

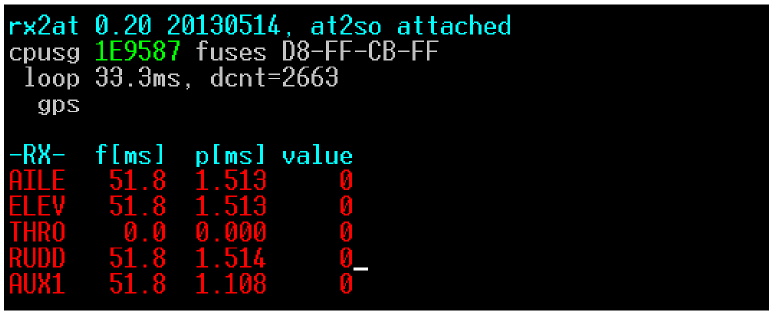
Type CTRL-B on HyperTerminal, this starts SETUP mode. You can retype the CTRL-B anytime; it will erase and rebuild the screen.
On a brand new board, without anything connected, the screen should look like this:



```
rx2at 0.20 20130514, at2so attached
cpusg 1E9587 fuses D8-FF-CB-FF
 loop 33.3ms, dcnt=2663
  gps

-RX-  f[ms]  p[ms] value
AILE    0.0  0.000     0
ELEV    0.0  0.000     0
THRO    0.0  0.000     0
RUDD    0.0  0.000     0
AUX1    0.0  0.000     0
```

In this example the GEAR channel is not used, otherwise it would show up at the bottom of the list.

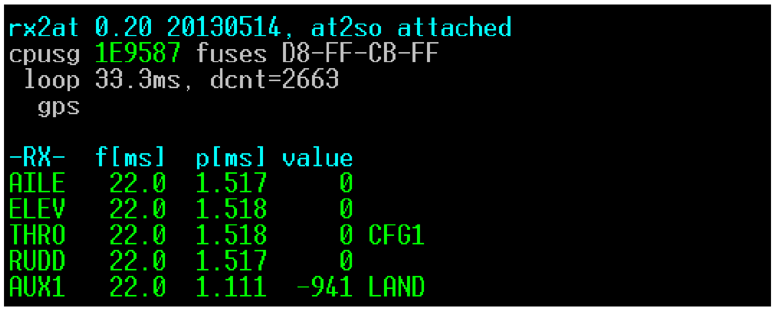
Once the RC-receiver is connected some of the timing columns should be filled in and the screen should look something like this:



```
rx2at 0.20 20130514, at2so attached
cpusg 1E9587 fuses D8-FF-CB-FF
 loop 33.3ms, dcnt=2663
  gps

-RX-  f[ms]  p[ms] value
AILE   51.8  1.513     0
ELEV   51.8  1.513     0
THRO    0.0  0.000     0
RUDD   51.8  1.514     0_
AUX1   51.8  1.108     0
```

Note: the TX is NOT turned on at this point! This example uses an ORANGERX receiver. The ORANGERX powers up with a long frame time and a ‘dead’ throttle channel if it has never had contact with the transmitter.

Assuming you already did a ‘bind’ between receiver and transmitter, the screen looks like this once the transmitter is powered up and linked with the receiver:



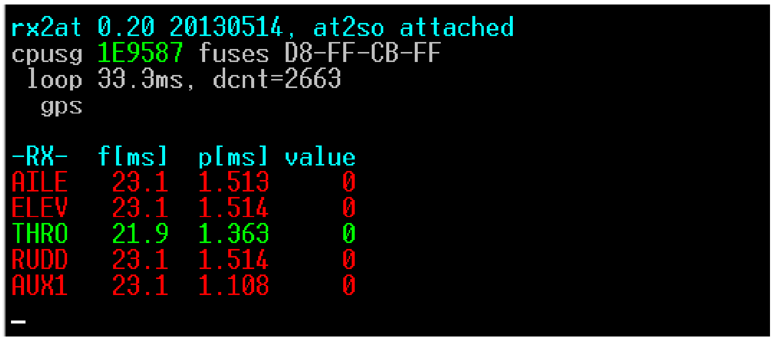
```
rx2at 0.20 20130514, at2so attached
cpusg 1E9587 fuses D8-FF-CB-FF
 loop 33.3ms, dcnt=2663
  gps

-RX-  f[ms]  p[ms] value
AILE   22.0  1.517     0
ELEV   22.0  1.518     0
THRO   22.0  1.518     0 CFG1
RUDD   22.0  1.517     0
AUX1   22.0  1.111  -941 LAND
```

Note: the flight mode switch is on LAND and the current configuration is CFG1, all sticks are centered. At this point you should check and adjust your TX/RX setup. Make sure the flight mode can be switched between LAND, FM1 and FM2. Check the stick channels AILE, ELEV, THRO and RUDD. They should all be independent from each other. Left AILE should give a negative value, up ELEV should give a negative value, left RUDD should give a negative value and THRO down should result in a negative value. The value should be roughly between -1000 and +1000 at extreme deflections of a stick and 0 in the center.

Pawelsky made a good video showing a properly configured mod in SETUP mode:
http://www.youtube.com/watch?v=RdLxztkRGZg

When moving the RUDD stick in LAND left and right, you will notice the words FTRM and ESTP appear in the RUDD line of the display. If you move the AILE stick to the right, VID+ should appear. If you switch the flight mode to FM2, switch it shortly to FM1 and then back to FM2 you should see the FM3 flight mode. Touch the AILE/ELEV stick; it should drop the flight mode back to FM2. Right after the sequence of switching into FM3, there is a short window (0.4 seconds) before FM3 is entered. If you move the AILE/ELEV stick within that window the 'sketch' will trigger an 'animation' on the drone. An 'animation' is a nod on drone 1 and a flip on drone 2. The display will indicate which animation ANIM1…4 got triggered.

Turn off the transmitter to simulate a failsafe condition, on my setup this looks like this:

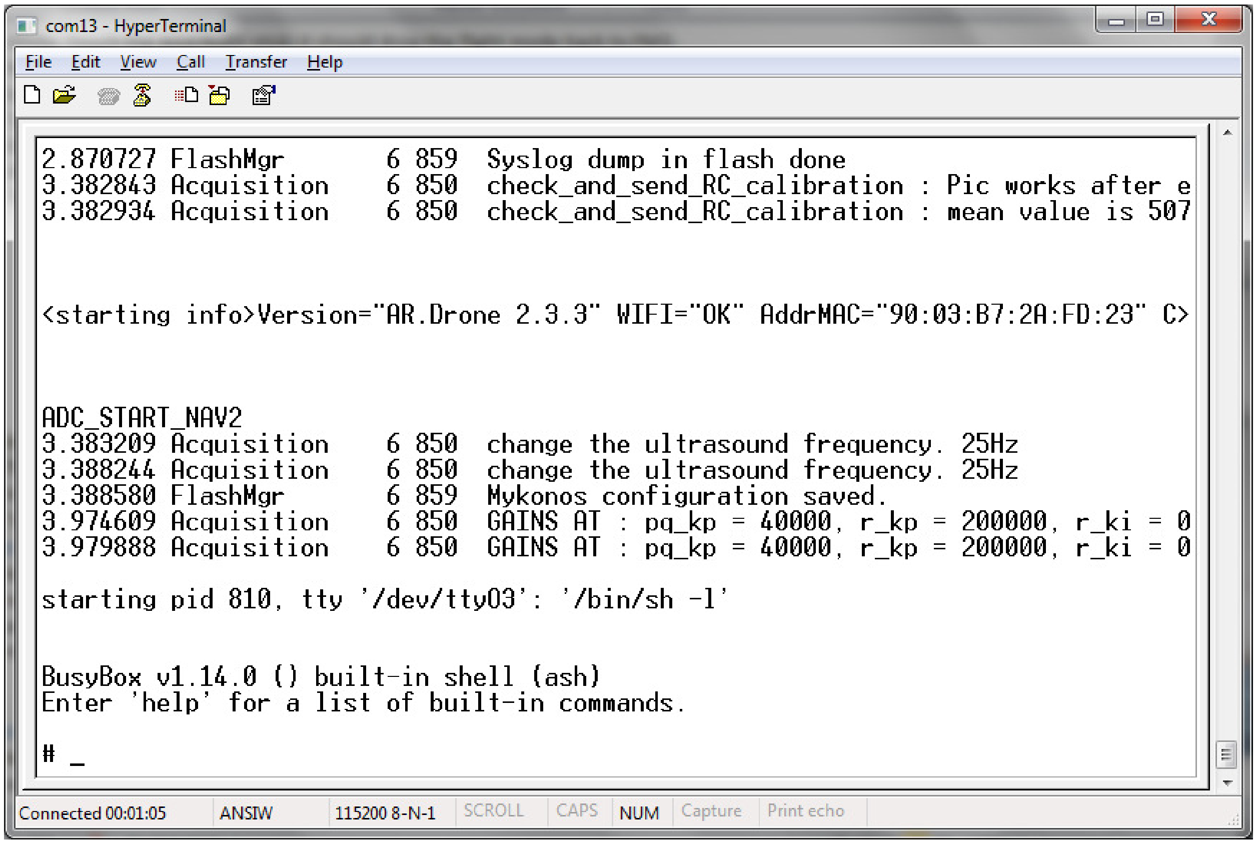


```
rx2at 0.20 20130514, at2so attached
cpusg 1E9587 fuses D8-FF-CB-FF
 loop 33.3ms, dcnt=2663
  gps

-RX-  f[ms]  p[ms] value
AILE   23.1  1.513     0
ELEV   23.1  1.514     0
THRO   21.9  1.363     0
RUDD   23.1  1.514     0
AUX1   23.1  1.108     0
_
```

The ORANGERX turns off all channels but the THRO channel. As long as any line in the display turns red, the TX disconnection has been detected by the 'sketch'.

Monitor mode

Monitor mode is entered when the mod is in SETUP waiting for a CTRL-B response from the user and there is activity on the drone interface. This can be accomplished by powering up the drone while the mod is waiting for the CTRL-B.

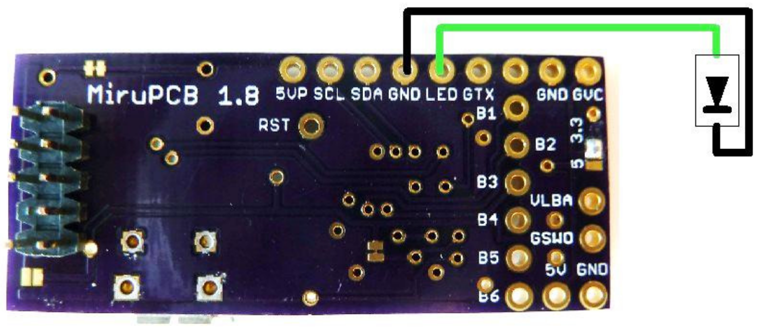


```
2.870727 FlashMgr       6 859  Syslog dump in flash done
3.382843 Acquisition    6 850  check_and_send_RC_calibration : Pic works after e
3.382934 Acquisition    6 850  check_and_send_RC_calibration : mean value is 507



<starting info>Version="AR.Drone 2.3.3" WIFI="OK" AddrMAC="90:03:B7:2A:FD:23" C>



ADC_START_NAV2
3.383209 Acquisition    6 850  change the ultrasound frequency. 25Hz
3.388244 Acquisition    6 850  change the ultrasound frequency. 25Hz
3.388580 FlashMgr       6 859  Mykonos configuration saved.
3.974609 Acquisition    6 850  GAINS AT : pq_kp = 40000, r_kp = 200000, r_ki = 0
3.979888 Acquisition    6 850  GAINS AT : pq_kp = 40000, r_kp = 200000, r_ki = 0

starting pid 810, tty '/dev/tty03': '/bin/sh -l'


BusyBox v1.14.0 () built-in shell (ash)
Enter 'help' for a list of built-in commands.

# _
```

When you hit ENTER after all the boot messages are done, the drone starts a shell for the interface. This is an easy way to check if the drone serial interface is working. If you don't see any messages something is seriously wrong. If you can see the messages but can't get a shell started by pressing ENTER, the serial receiver on the drone might be bad…

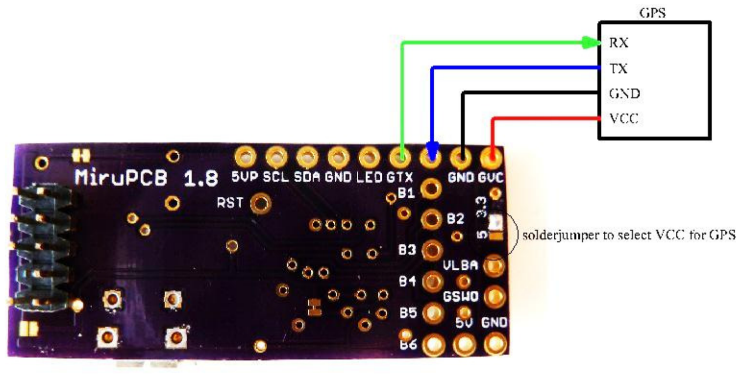
Remote LED connection

MiruPCB has an output for driving a remote version of the activity LED. The output is driven with 5V and has a 330 Ohm series resistor integrated.

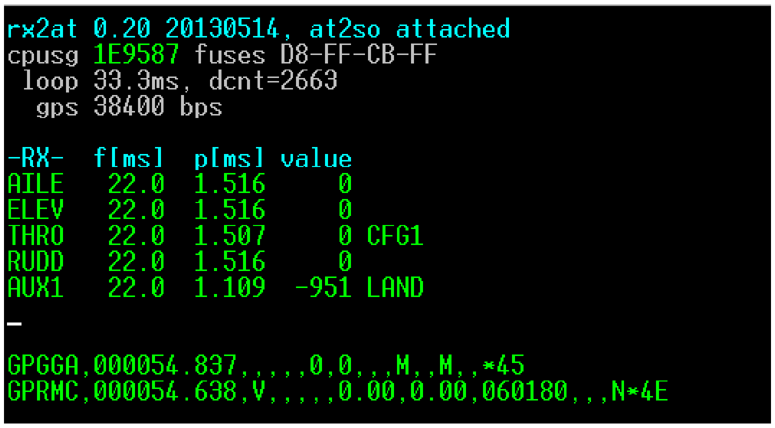


GPS connection

MiruPCB supports 3.3V and 5V GPS units. The GPS unit should understand and output NMEA sentences. The default baud rate must be less or equal 57600 baud for the GPS serial port to work with the mod. MiruPCB has an integrated 4K7 resistor in the transmit line.



Here is a picture of the mod in SETUP mode when a GPS is connected:

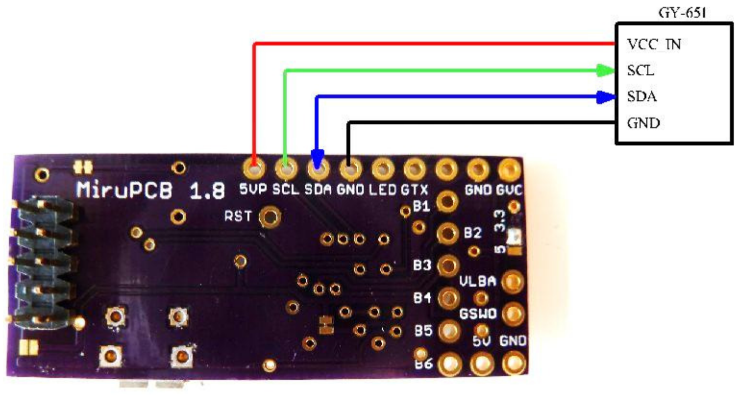


```
rx2at 0.20 20130514, at2so attached
cpusg 1E9587 fuses D8-FF-CB-FF
 loop 33.3ms, dcnt=2663
  gps 38400 bps

-RX-  f[ms]  p[ms] value
AILE   22.0  1.516     0
ELEV   22.0  1.516     0
THRO   22.0  1.507     0 CFG1
RUDD   22.0  1.516     0
AUX1   22.0  1.109  -951 LAND
_

GPGGA,000054.837,,,,,0,0,,,M,,M,,*45
GPRMC,000054.638,V,,,,,0.00,0.00,060180,,,N*4E
```

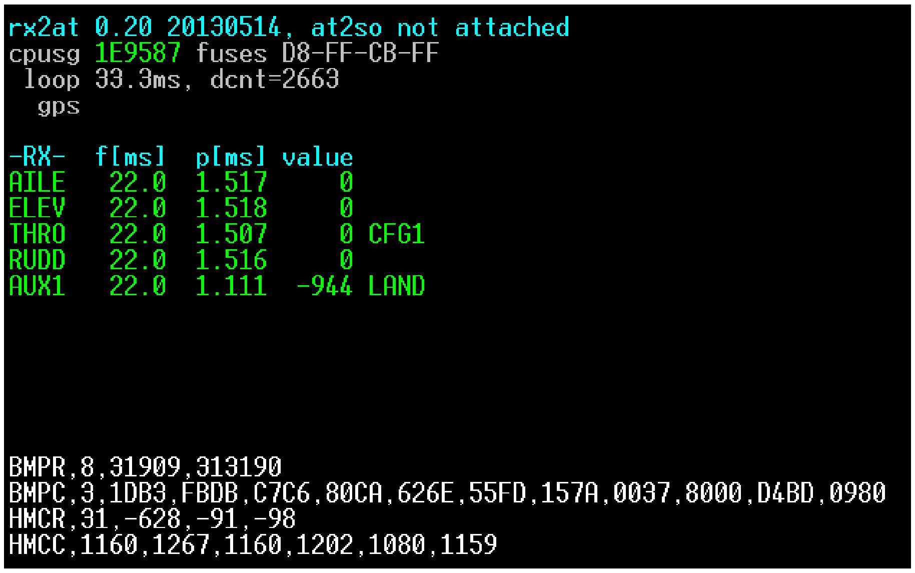
The program auto detects the presence of a GPS by listening to the GPS port and trying to figure out what the baud rate of the messages is. Once detected, it tries to set the baud rate to 38400 baud, disables all messages except GPGGA with a 5Hz update rate and GPRMC with a 1Hz update rate.

I2C connection

MiruPCB has a 5V I2C interface. It can be used to add a barometer and/or a magnetometer.



The picture below shows how the SETUP mode displays data from a GY-651. The GY-651 is a small I2C board which has a Bosch BMP085 barometer and a Honeywell HMC5883L magnetometer.

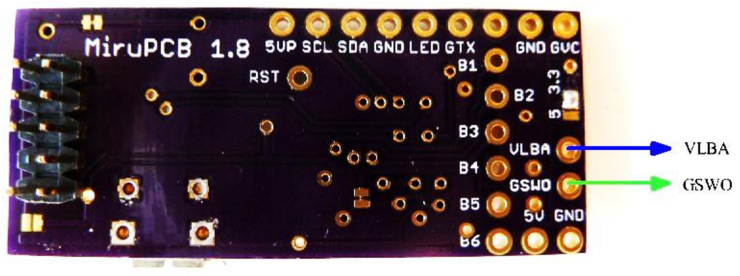


```
rx2at 0.20 20130514, at2so not attached
cpusg 1E9587 fuses D8-FF-CB-FF
 loop 33.3ms, dcnt=2663
  gps

-RX-   f[ms]  p[ms] value
AILE    22.0  1.517     0
ELEV    22.0  1.518     0
THRO    22.0  1.507     0 CFG1
RUDD    22.0  1.516     0
AUX1    22.0  1.111  -944 LAND






BMPR,8,31909,313190
BMPC,3,1DB3,FBDB,C7C6,80CA,626E,55FD,157A,0037,8000,D4BD,0980
HMCR,31,-628,-91,-98
HMCC,1160,1267,1160,1202,1080,1159
```

Note: when the support software for the barometer and magnetometer is enabled in the ‘sketch’, there is not enough code space to accommodate the companion program in the FLASH memory of the processor. You need to install the companion program on the drone to remedy this situation.

Optional outputs

MiruPCB supports two optional outputs. They are configured in the 'sketch'. Keep in mind these outputs do not have a lot of power, the signals are meant to drive a switch for the devices you want to activate.



VLBA, Visible Low Battery Alert

When enabled this output activates on a low battery condition.

GSWO, Gear Switch Output

When enabled this output reflects the GEAR channel status.

Startup sequence

The following list is the startup sequence of a working implementation:

	User	MiruPCB LED	Drone LEDs	
1	Turn on TX			
2	Power drone		Red	
3		1 blink/sec	Red	Radio not ready, turn it on, bind?
		2 blink/sec	Red	Flight mode not LAND, switch it
		3 blink/sec	Red	Drone still booting
		4 blink/sec	Red or Green	Drone still talking
4		ON	Green	Drone has booted
5		ON	Green	'sketch' contacts drone and switches baud rate to 38400 baud
6		ON	Green	'sketch' searches companion program
		1 + 1 blink	Green	Shell does not respond
		1 + 2 blinks	Green	Problem with D=/data/video/usb command
		1 + 3 blinks	Green	Problem checking for pilotXXX.arm program
		1 + 4 blinks	Green	Problem checking for at2soXXX.arm program
		1 + 5 blinks	Green	Problem with D=/data/video command
		1 + 6 blinks	Green	Problem checking for pilotXXX.arm program
		1 + 7 blinks	Green	Problem checking for at2soXXX.arm program
		Dim	Green	Uploading companion program from FLASH
		1 + 8 blinks	Green	Can't locate/upload companion program
7		ON	Green	Launch companion program
8		ON	Green	Waiting for companion program start ACK
		1 + 9 blinks	Green	Companion program detected other controller and terminated
9		15Hz blink	Blink orange	Companion program started
10		15Hz blink	Blink green	Custom configuration sent to firmware
11	RUDD right	15Hz blink	Red	Check user emergency trigger
12	RUDD right	15Hz blink	Green	Clear emergency
13	RUDD left	15Hz blink	Blink green	Drone flat trim, GPS data invalid
			Blink red/green	Drone flat trim, GPS data valid
14		15 Hz blink	Green	Drone and Mod ready to go
15	Switch to FM1/2	OFF	Launch blink	Drone launches

Companion program

The ‘sketch’ is the program for the Arduino processor or MiruPCB. It is called ‘sketch’ because of the naming conventions the Arduino IDE uses for the source of a program to be compiled for a target processor. This project makes use of the well-established Arduino IDE for compiling and downloading software to a board with an Atmel processor. All Arduino IDE ‘built-ins’ are circumvented by doing a <Sketch><Add File…> and choosing the source code of this project. There is only one file to add to the ‘sketch’. Make sure you select the right processor in <Tools><Board><Micro> before you compile. It also helps a lot if you select the right <Tools><Serial Port> for the download.
You can get a free copy of the Arduino IDE at www.arduino.cc .

The ‘sketch’ needs an ally program on the drone to issue commands to the drone firmware on its behalf since it does not communicate to the drone like an iDev over Wi-Fi. This program is called companion program within this project. The program is compiled for the drone processor. You need a cross tool chain to modify and recompile it; the drone itself does not have a compiler installed so you have to cross compile it on a different system. Do not worry about this too much; the mod’s distribution has precompiled versions of the companion program.

The ‘sketch’ is called ‘rx2at’ because its task is to translate RC-receiver pulse lengths into AT commands for the drone. It turned out that the translation can also be done by the program on the drone so the values of the receiver signals can be interpreted in the companion program of the drone and a more efficient transfer protocol can be put to the task.

There are several versions of the companion program with different functionalities. All versions have the ability to translate commands from the ‘sketch’ to the drone firmware and provide status information to the ‘sketch’. You can consider this as the core of the companion program. The most basic one, which contains nothing but the core, fits into the FLASH of the Arduino processor and the ‘sketch’ knows how to load and get it going on the drone, providing for a true plug&play experience of RC control of an AR.drone. The basic version of the companion program does not have any bells and/or whistles, the data from the ‘sketch’ is screened for the most rudimentary commands and everything else is dropped into the bit bucket (aka ignored).

Currently the ‘sketch’ (rev 0.20+) knows about and searches for a companion program in five different locations in the following order:

1. a file with the path /data/video/usb/pilotXXX.arm on the drone
2. a file with the path /data/video/usb/at2soXXX.arm on the drone
3. a file with the path /data/video/pilotXXX.arm on the drone
4. a file with the path /data/video/at2soXXX.arm on the drone
5. the file in FLASH if present

The XXX stands for the revision of the running ‘sketch’. If none of the files can be found or accessed, the mod will bow out since an essential part to control the drone is missing.

at2soXXX.arm is the minimum code you need running on the drone to make the mod function. It can reside in FLASH or be preinstalled on the drone. If it is in FLASH the mod vanishes and leaves no traces on the drone once the drone is powered off.

In the distribution the 'sketch' is called 'rx2atp.c'. 'rx2atp.c' has the companion program appended and will produce a 'sketch' that can upload the companion program from FLASH once the 'sketch' is downloaded to the processor by the Arduino IDE. It is your task to configure the 'sketch' for your particular installation before you download it into the processor. Most configurations are easy and are done within the first 50…60 lines by editing macros. Make sure you read and understand the comments. If you have a non-standard RX channel layout you have to search for the procedure 'main()' where the assignment of inputs to RX channels is done. If you are tinkering with the 'sketch' rx2atp.c you might run into the problem of overflowing the capacity of the FLASH. For these situations 'rx2atp.c' can be configured to drop the companion program. If you replace the 1 of the macro AT2SO in the first line by 0, you have the version that does not have the companion program attached and therefore gives more room for your additions to the 'sketch'. If go this route, you have to install the companion program on the drone so it can be found by the 'sketch'. On a drone 2 the 'at2soXXX.arm' can reside on the USB stick or, like on a drone 1, in the /data/video directory. The /data/video directory is the drone ftp server 'home', so it is easy to transfer the file from the distribution to the drone.

The companion program at2soXXX.arm creates a log file in /tmp/at2so.log. This log file is temporary and does not survive a power cycle on the drone. If you need to transfer it to another computer you have to 'telnet' into the drone while it is still there, copy it to /data/video and then get it with 'ftp'.
Changes to the core of the companion programs are not for the faint of heart. It got a lot more complicated when Parrot introduced their session concept, which is beyond me up to the day of this writing.

pilotXXX.arm is a more sophisticated companion program. It writes its log to the FLASH memory of the drone. It uses /data/video/P%d.txt as the file name of the log, %d is substituted with the boot sequence number of the 'sketch' modulo 5. So you will see P0.txt … P4.txt. This is done so the log does not consume too much space on the FLASH drive of the drone. A single log file is limited to 4Mb, after reaching that limit it is turned off. These logs survive a reboot of the drone, hence the cycling of the file names, they can be retrieved with 'ftp'. The first log entry is always the BOOT record, which contains the boot sequence number. This means if you telnet into the drone and do a 'grep BOOT /data/video/P*.txt' you can identify the last log generated by the highest boot sequence number. In case you are deleting log files on the drone, make sure you issue a 'sync' command when you are done. The same applies if you are transferring new versions of a companion program to the drone, otherwise changes may not be fully committed to the FLASH drive, which usually results in disaster…