



## RotorHazard FPV Timer - OTA Updater and Manager



by szafranski

If you are using RotorHazard FPV race-timer or Delta5 timer - this tutorial is for you.

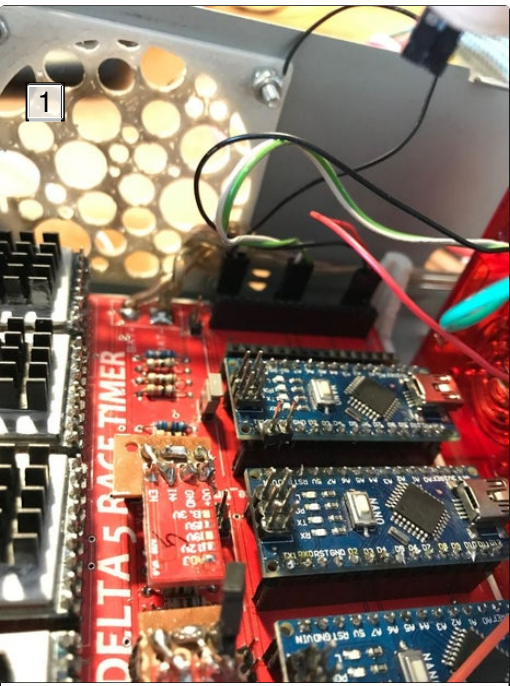
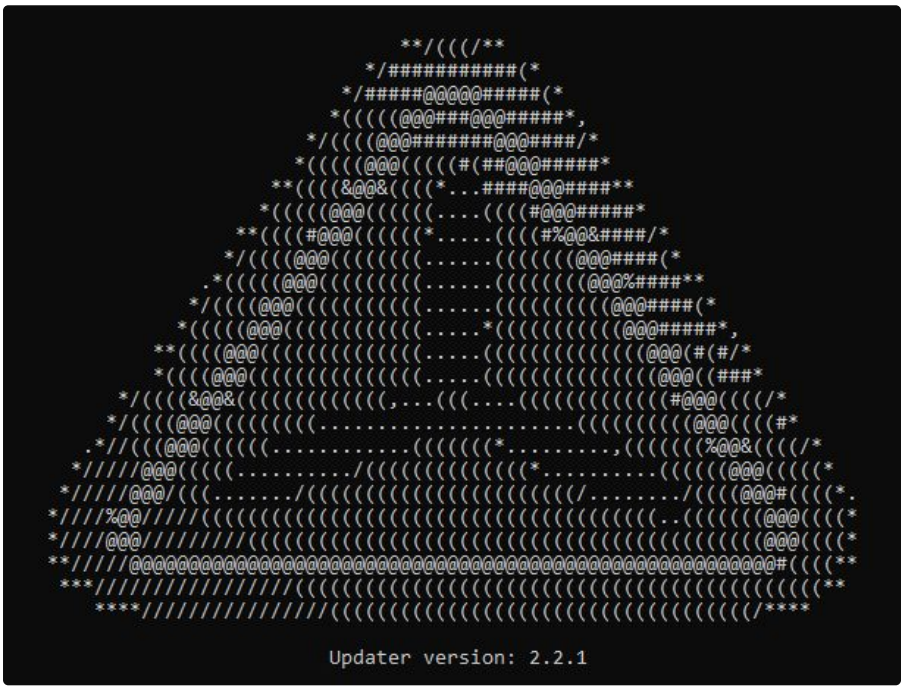
It will bring new functionalities to your timer and you will have ability to update server and nodes "Over The Air". Consider this as a combined improvements for software and hardware. You can use both or utilize just one of those.

Before you start:

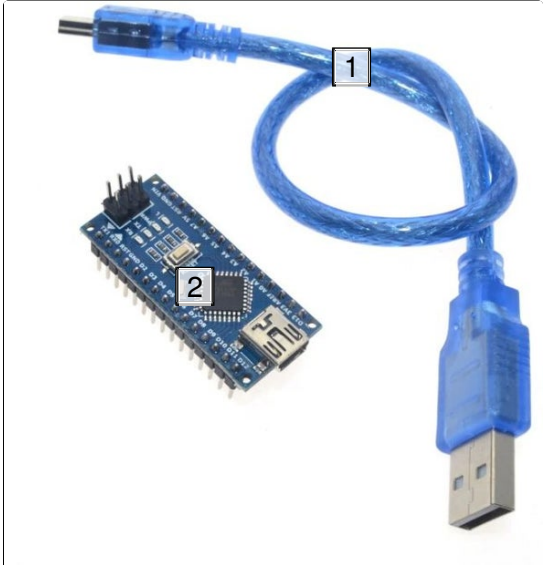
- Collect all required components.
- Make sure that you have internet connection.
- Read this instruction at least one time before performing the mod so you will know what to expect.
- You can also read about history of this project later in the instructable.

### Supplies:

1. RotorHazard race timer (already made or just components of it)
2. About 50 cm of thin wire + jumper wires or tool for attaching female gold-pins to the wire
3. Soldering iron + solder
4. 2 resistors: 5kOhm and 10 kOhm - or any other combination with 1/2 ratio in 5-20 kOhm range  
OR 3 resistors of the same value in 10-20kOhm  
OR logic level converter - but using resistors is easier
5. Few gold pins - optional
6. PC connected to the internet
7. miniUSB to USB A cable - probably



1. my friend is as architect - he designed this for some reason...



1. miniUSB cable  
2. Arduino Nano



1. female to female jumper wires

Step 1: Prepare Your Arduinos

Make sure before any attempts of using this software and going further, that your Arduinos DOES NOT contain any programs utilizing serial connection. Basically "Serial.begin" cannot be implemented on them. Last official code from RotorHazard repository had that feature enabled so if you already used to use that firmware you have to wipe it out. Just flash Arduinos again using PC and USB cable before embedding nodes in the timer. You can flash anything on them, for example "Blink" sketch - from "Examples" menu.

If you have newly bought Arduinos - you are most likely safe and you can skip this step.

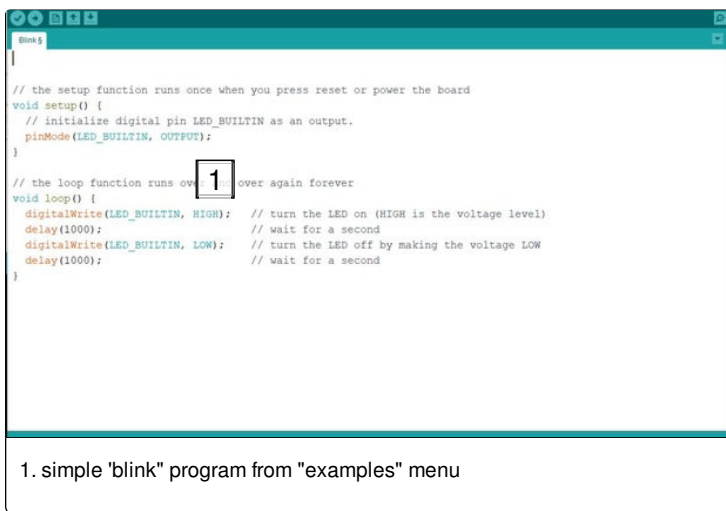
You would have to do it just once - before placing Arduinos in the timer.

It is explained in FAQ and "history" step why is it crucial

Link to Arduino IDE download page:

<https://www.arduino.cc/en/main/software>

Important: If you want to proceed with hardware-based steps for now and do this step later - it is ok. Just remember to do this before embedding Arduinos in the timer.



```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

1. simple 'blink' program from 'examples' menu

## Step 2: Make Additional PCB Wiring - TX/RX Line

Connect all of the TX pins together and all of RX pins together - on Arduinos. You can do it underneath the PCB so it will still look clean at the top.

Only the white and purple wires on the photo are UART lines. Rest of the cables are reset lines and can be connected from the other side. You can do it however you want.

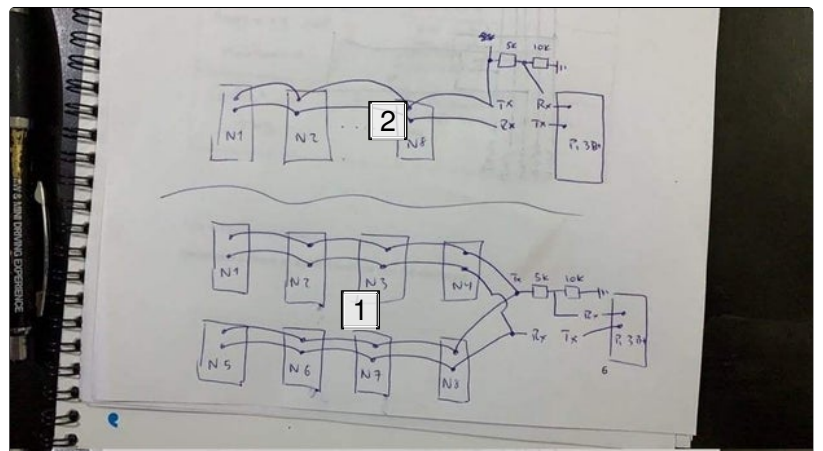
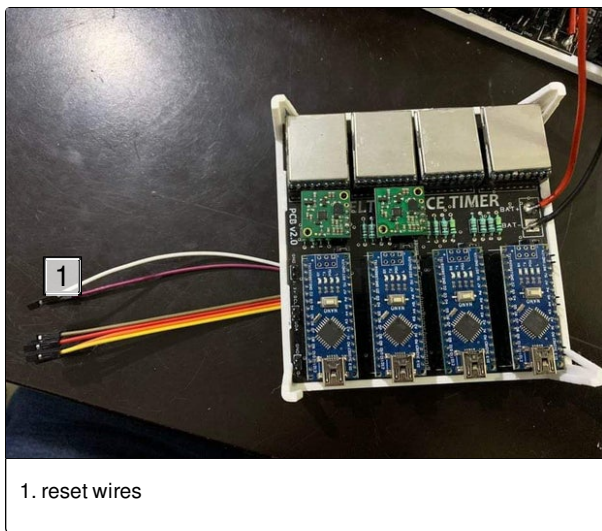
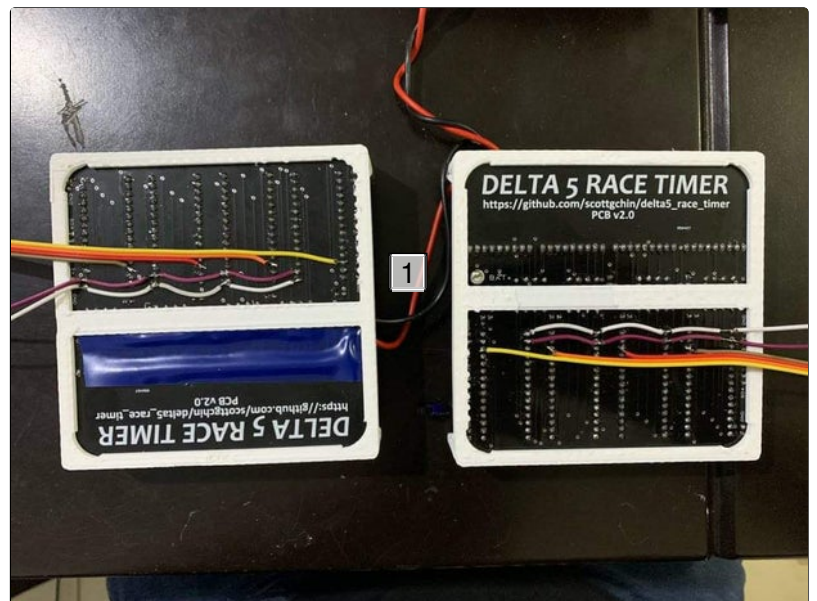
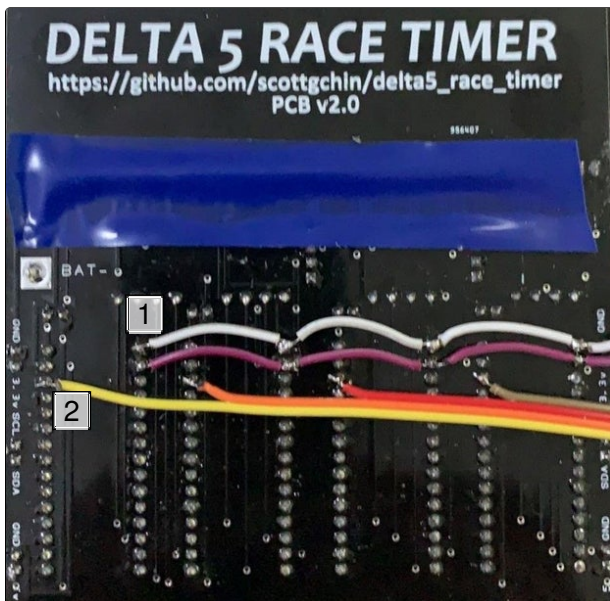
If you have 2 PCBs connected together - connect all of the pins RX/TX pins from both PCBs.

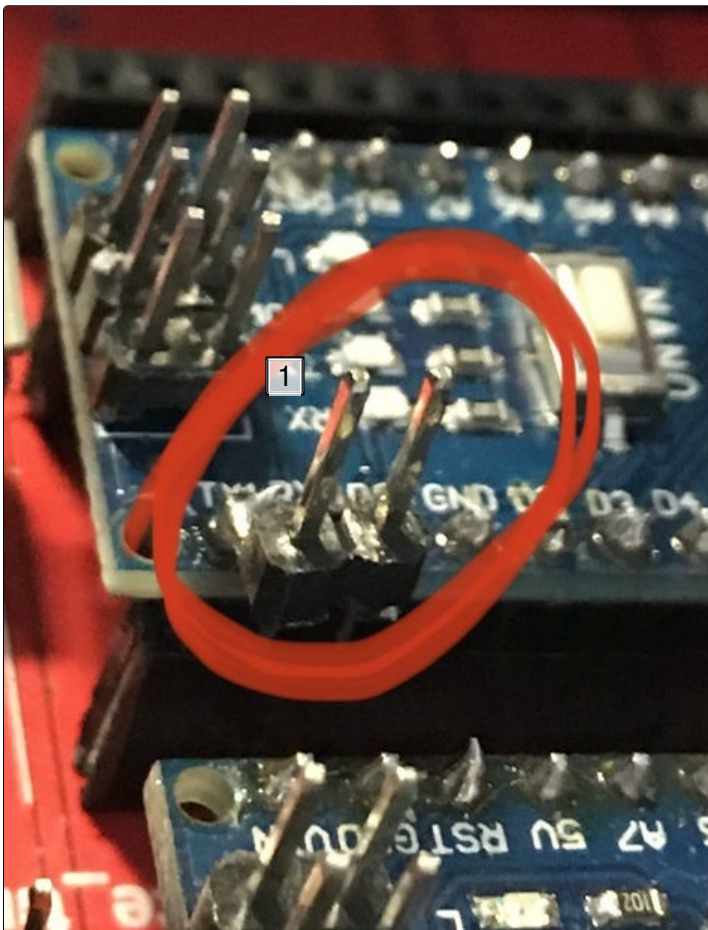
I soldered two male gold-pins to one of the Arduinos so I have RX and TX line exposed near the Raspberry.

You can do it the same way.

If you are starting with your build from the scratch - assemble your PCB according to instructions from RotorHazard or Delta5 timer page, before adding mods mentioned above.







1. additional gold-pins - UART



1. RX and TX lines  
 2. ground auto-numbering mod (not required)  
 3. diode - some basic protection against reversed voltage

### Step 3: Make Voltage Divider or Use Logic Level Converter



The common Arduinos RX line has to be connected to Raspberry's TX pin (pin 8 - UART0 TX).

Arduinos TX pins have to be connected to Raspberry's RX pin (pin 10 - UART0 RX) via voltage divider or with logic level converter. It is caused by the difference of voltage tolerance between the Arduino and the Raspberry. Arduino operates at 5V logic level and Raspberry at 3.3V. Converting the logic level is only required on the line where Arduinos TXes are connected to Raspberry's RX, cause Arduino transmits at 5V.

On the other line the Pi is transmitting at 3.3V and Arduinos are only receiving so it is save without voltage divider or level converter.

I placed small PCB with the voltage divider on RX and TX pins. You don't have to do it this way. I just didn't

want to have more cables than needed in my timer.

You can just make "Y" shape divider as a part of the cable - photo attached.

You can make voltage divider in very smart way by utilizing 3 resistors of the same value. Two of them connected parallelly which effectively makes them combined resistance halved - see the photo.

IMPORTANT! Don't make mistake with TX/RX. 5V is too much for Raspberry's pins on UART line!

Small help with calculating resistor values:  
<http://www.ohmslawcalculator.com/voltage-divider-calculator>

**NANO PINOUT**

1. UART pins  
 2. reset pin is doubled on the ISP header - you can use that one if you want  
 3. reset pin

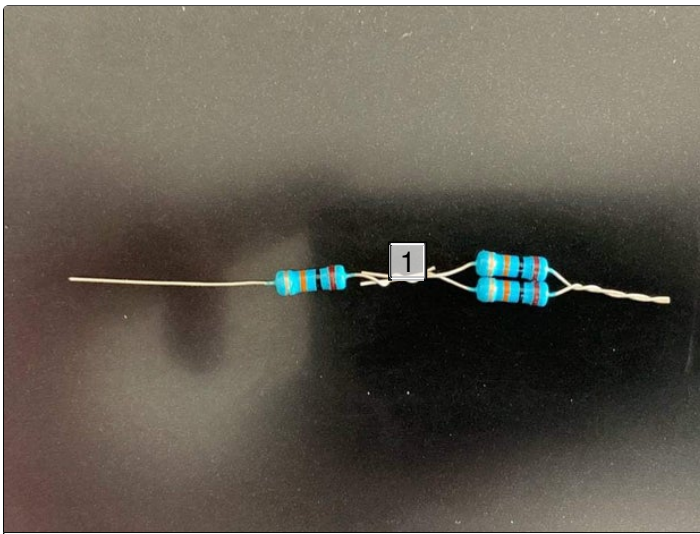
Absolute MAX per pin 40mA recommended 20mA  
 Absolute MAX 200mA for entire package

USB JACK Mini Type B

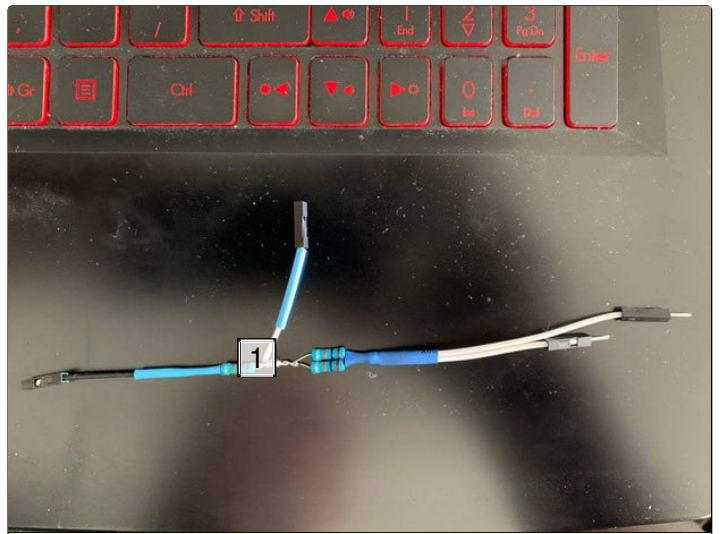
Analog exclusively Pins

bq

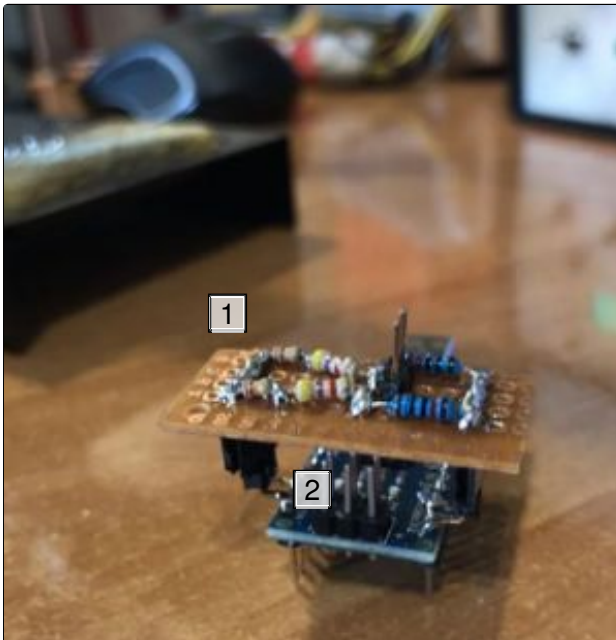
1. Arduino - TX - 5V  
 2. Raspberry - RX - 3.3V  
 3. 5kOhm  
 4. 10kOhm



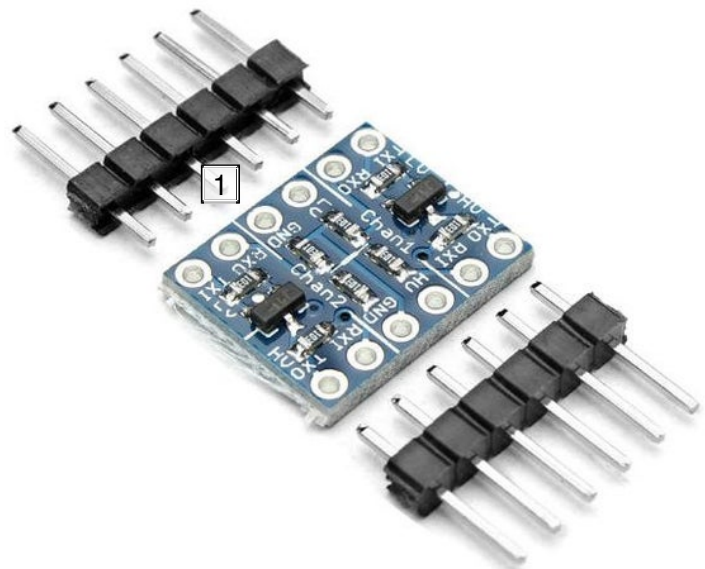
1. 3 same value resistors



1. Y shape solution



1. voltage divider PCB  
2. just one of the nodes is done like this



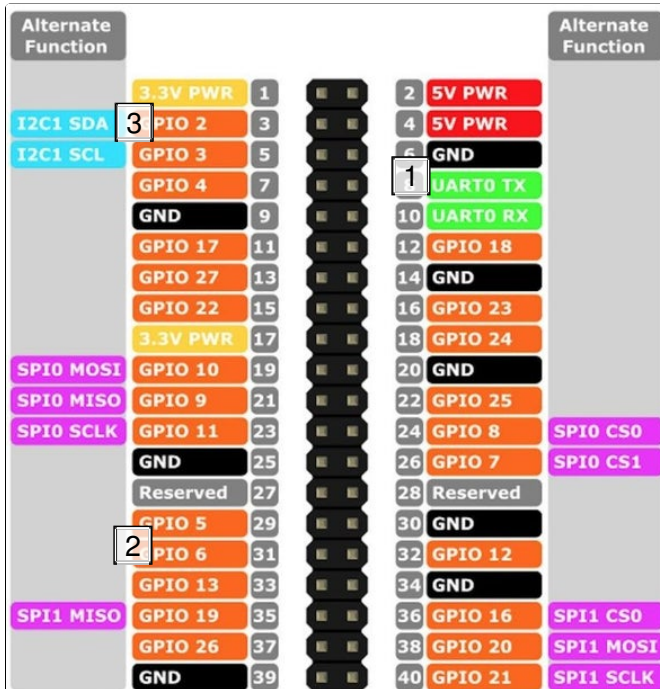
1. logic level converter

## Step 4: Connect RST Pin of Every Arduino to the Raspberry's GPIO Header

If you want to use another pins - remember to use general purpose pins. Not reserved ones, gnd, 3.3V, 5V

If you are using different pins than suggested ones, you will have to make changes in one of the files. It will be explained further in the instructables.

Important: If you haven't done Step 1 already - do it now.

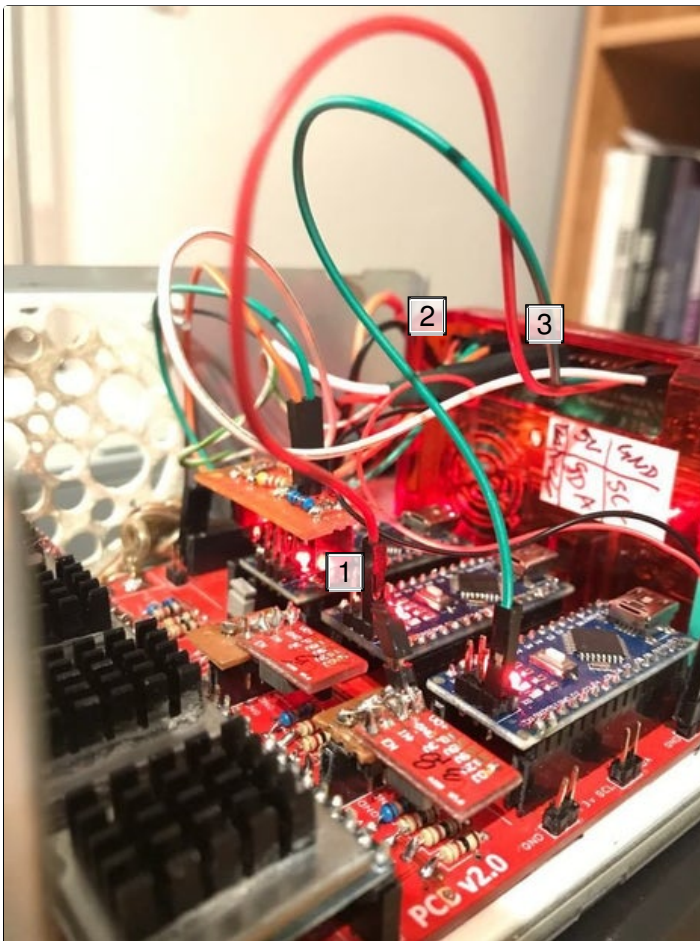


1. UART pins
2. pins used as reset pins by default
3. I2C pins - already used in the timer

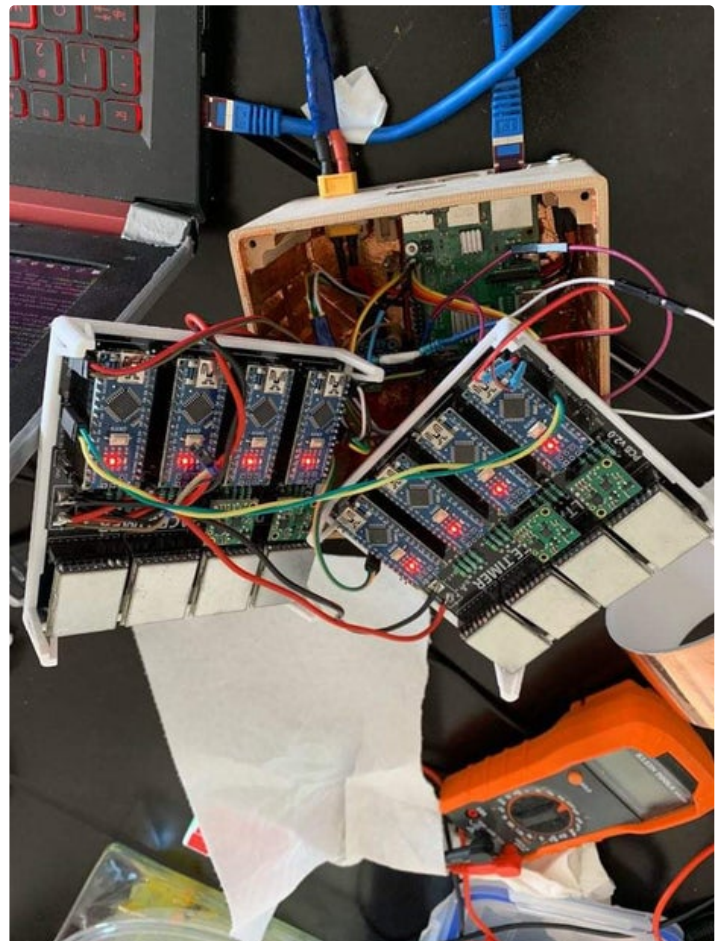
Arduino node number	Raspberry's GPIO pin number
1	12
2	16
3	20
4	21
5	6
6	13
7	19
8	26
RX line	UART0 TX
TX line	UART0 RX

1. numbers refer to GPIO numbering, not actual pin order





1. reset wires
2. UART0 pins
3. GPIO header



## Step 5: Login Into Raspberry Via SSH and Download the Updater

Open the SSH connection with your Raspberry.  
Establish connection to the internet.

You can also hook up display to the Raspberry and  
login into Raspbian at the Raspberry itself.

After logging into Pi download repository from  
github page using this command:

```
git clone https://github.com/szafranski/RH-ota.git
```

Enter downloaded folder:

```
cd RH-ota
```

And open update script:  
`python update.py`

If you got an error after entering first command you  
probably have to install git from apt.

Use command:

```
sudo apt install git
```

```
pi@raspberrypi:~/RH-ota
C:\Users\ssh pi@192.168.137.120
pi@192.168.137.120's password:
Linux raspberrypi 4.19.97-v7+ #1294 SMP Thu Jan 30 13:15:58 GMT 2020 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb 21 14:22:43 2020 from 192.168.137.1

SSH is enabled and the default password is 'pi'. The 'pi' user has not been changed.
This is a security risk - please login with the 'pi' user and type 'passwd' to set a new password.

pi@raspberrypi:~$ git clone https://github.com/szafranski/RH-ota.git
Cloning into 'RH-ota'...
remote: Enumerating objects: 792, done.
remote: Total 792 (delta 0), reused 0 (delta 0), pack-reused 792
Receiving objects: 100% (792/792), 5.72 MiB | 1.86 MiB/s, done.
Resolving deltas: 100% (390/390), done.
pi@raspberrypi:~$ ls
backup_RH_data  bme280  pi_ina219  RH-ota  RotorHazard  RotorHazard.old  rpi_ws281x
pi@raspberrypi:~$ cd RH-ota
pi@raspberrypi:~/RH-ota$ ls
firmware  how_to  README.md  resources  rpi_soft.py  update.py
pi@raspberrypi:~/RH-ota$ python update.py
```

1. downloading software

## Step 6: Prepare Raspbian OS and UART

If you are doing this for the first time you have to install software that connects with Arduino and has ability to program it. Do it by entering Additional Features menu and select point 1 - "Install avrdude".

Next you have to enable serial port on GPIO header (UART protocol) and prepare it to be connected with external device. It is utilized to be the console output

by default so it is basically useless until you make some changes.

Enter Additional Features menu and go to point 2 - "Enable serial protocol".

Next you will be asked to reboot the Raspberry. Do it.

```
#####
###                                     ###
###                                     ###
###                                     ###
### You are about to flash nodes file. Please do not interrupt this operation. ###
###                                     ###
###                                     ###
#####

FEATURES MENU

1 - Install avrdude
2 - Enable serial protocol
3 - Fix GPIO pins state
4 - Raspberry as Access Point - coming soon
5 - Useful aliases - coming soon
6 - Go back
```

1. those words changed cause software has much more features now

## Step 7: Configure Downloaded Software

After rebooting the timer configure your software by typing:

cp distr-updater-config.json updater-config.json

and then

nano updater-config.json

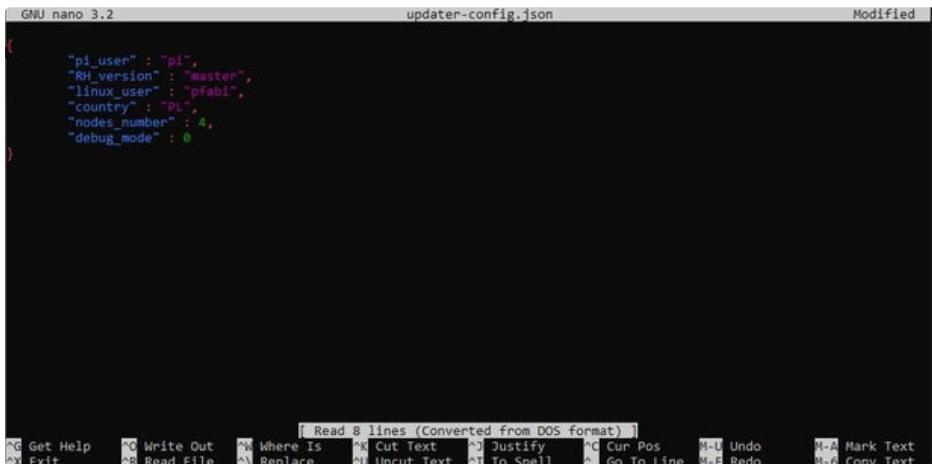
You can also choose what version of RotorHazard do you want to use. It is directly related to node firmware version that can be used so you can change those by changing "RH\_version" in the file.

Default version is always last stable release of the software - so it can be quite outdated.

Remember to always update server software if you have updated firmware in the first place.

In the future there will be just one config file implemented so it will be even easier. EDIT: There is now :)

If you want to use another pins for reset purposes for some reason it has to be changed in the update.py file - see on attached photo.



```
GNU nano 3.2 updater-config.json Modified
{
  "pi_user": "pi",
  "RH_version": "master",
  "linux_user": "pfabi",
  "country": "PL",
  "nodes_number": 4,
  "debug_mode": 0
}
```



```
11 preferred_RH_version = data['RH_version']
12
13 if preferred_RH_version == 'master':
14     firmware_version = 'master'
15 if preferred_RH_version == 'beta':
16     firmware_version = 'beta'
17 if preferred_RH_version == 'stable':
18     firmware_version = 'stable'
19 if preferred_RH_version == 'custom':
20     firmware_version = 'stable'
21
22 nodes_number = data['nodes_number']
23
24 ##### ... Enter pins connected to reset pins on Arduino-nodes
25
26 reset_1 = 12 1
27 reset_2 = 16
28 reset_3 = 20
29 reset_4 = 21
30 reset_5 = 6
31 reset_6 = 13
32 reset_7 = 19
33 reset_8 = 26
34
35 if data['debug_mode'] == 1:
36     linux_testing = True
37 else:
38     linux_testing = False
```

1. change pins assignment if you want to use different than default

## Step 8: Use Downloaded Software



Run software by typing:

```
python ~/RH-ota/update.py
```

If you get an error you probably have to install python. Type: 'sudo update && sudo apt install python'

MENU:

If you want to update or install (or possibly even downgrade) server software enter point 1.

If you want to flash firmware on Arduino-based enter point 2.

Follow the instructions on the screen.

It should work automatically :)

After updating either of components - nodes or server - remember to "disable" and re-enable all the channels on the server page, before using timer - or power-cycle whole system.

If you want to be absolutely sure that software and/or firmware updated correctly check out version number or API number during manual server opening. You can also flash the 'Blink' hex file, confirm that nodes are not being recognized and then flash new firmware - but it shouldn't be necessary.

Besides that there are interesting additions in 'Features menu'. You can use them. This part of the software will be developed even further.

```
#####
###                               ###
###      RotorHazard              ###
###      OTA Updater and Manager   ###
###                               ###
#####

MAIN MENU

1 - Server software installation and update
2 - Nodes flash and update
3 - Start the server now
4 - Additional features
5 - This is my first time - READ!
6 - Exit
```

```
#####
###                               ###
###      RotorHazard              ###
###      OTA Updater and Manager   ###
###                               ###
#####

FEATURES MENU

1 - Install avrdude
   Enable serial protocol
3 - Fix GPIO pins state
4 - Raspberry as Access Point - coming soon
5 - Useful aliases
6 - Self updater - coming soon
7 - Go back
```

1. enter 1 and 2 after first opening the updater

```
AUTOMATIC UPDATE AND INSTALLATION OF ROTORHAZARD RACING TIMER SOFTWARE

This script will automatically install or update RotorHazard software on your Raspberry Pi.
All additional software dependencies and libraries also will be installed or updated.
Your current database and config file should stay on the updated software.
After rebooting please check by typing 'sudo raspi-config' if I2C, SPI and SSH protocols are active.

Source will be 1 'er' repository of RotorHazard software on github - or version chosen by you.
Make sure that you are logged as use 2 .
You can change those by opening file 'rpi_soft.py' in text editor - like 'nano'.

Enjoy!

'i' - Install software from scratch
'u' - Update existing installation
'a' - Abort
```

1. can be changed  
2. should match username of the Raspberry

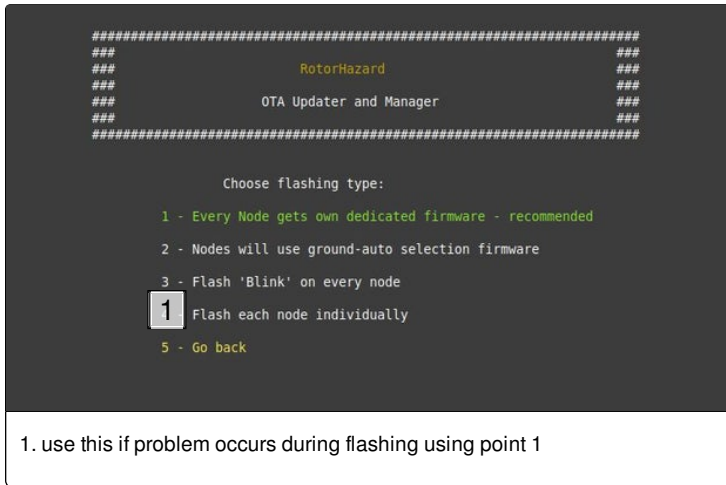
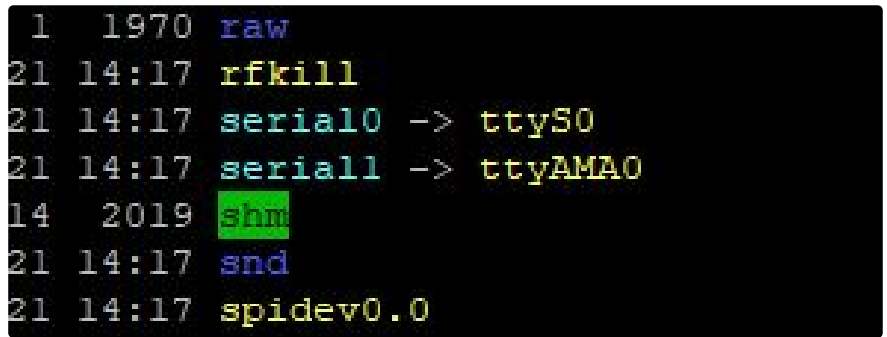
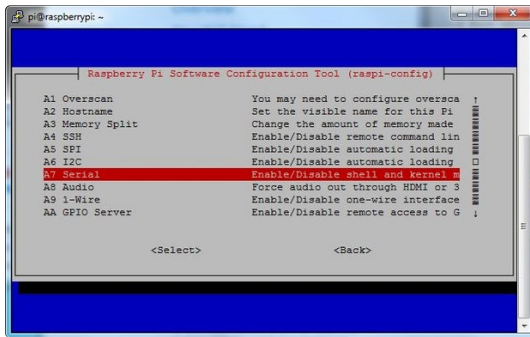
name		Default
Stores frequency settings and sensor tuning values.		
Preset: <span>Apply Node 1</span> <span>R1367</span> <span>IMD5C</span> <span>IMD6C</span> <span>Raceband 8</span>		
	<b>Node 2</b>	<b>Node 3</b>
	<b>1</b> Disabled	R3 5732
	0	5732
<span>EnterAt</span>		

1. channel disabled



- Arduinos can't be flashed - check the wiring. 3 times. Remember that EVERY Arduino has to be connected via reset pin and UART (RX/TX pins). Updating script doesn't want to open - check if python is installed and install it by using command 'sudo apt install python'
- "It just doesn't work" - it is
- Software works but in some cases it crashed or exits to system - make sure that your username in both files (update.py and rpi\_soft.py) matches your system username. Updating server software takes long time - when server is being updated, the Raspberry itself is being updated as well. If you haven't updated Raspbian (Raspberry's OS) for some time it can take up to 20 minutes. If you get only errors - check internet connection.
- Few nodes updated with no problem but few of them were being flashed very slowly and I can't see them after opening the server - use option "Flash each node individually" and flash problematic nodes this way.
- "I tried few times and I really think that my Raspberry's UART isn't working properly" - open configuration assistant on your Pi by typing 'sudo raspi-config' -> Interfacing options -> Serial and after hitting enter choose "no" for first question and "yes" to second one. Reboot if asked. You can also type 'cd /dev' and then 'ls -l' and look for 'serial0 -> ttyS0'. It should look as on attached image. If it is, it definitively means that your UART is working.
- There are few font colors used in the software. If you feel like you can't see something - just change console window background color in settings or use another for SSH or terminal usage.
- After enabling Serial on the Raspberry and rebooting I can't connect to Raspberry via SSH - probably some communication is being performed on UART line and prevents Raspberry from booting. Unplug those wires for a moment and reboot the Pi. Check if serial connection is working properly according to previous point, plug wires again and check if Raspberry boots normally.
- You want to know what you are doing and automatic processes aren't your thing - software written for this project is as combination and compilation of about 100 hours of thinking, learning, wiring, troubleshooting and researching. If you want to know more what is actually done open 'py' files in some text editor and you can figure out what's going on after some time.
- After opening the program or entering 'Aliases' menu I can see random character all over the place - resize your terminal window. And possibly scroll up a bit."
- I want to flash my own 'hex' files. How to create them and what to do next?" - you have to use Arduino IDE. Install it - preferably on the PC and after creating/opening ino file compile it and find created hex file. More instructions about doing that can be found here: <https://www.instructables.com/id/HOW-TO-GET-HEX-F...> After that copy hex files to the Raspberry using scp or upload them on GitHub and download on the Pi. If you are using Desktop version of Raspbian you can just use standard ways of downloading the files or even create hex files using Arduino IDE on Raspberry itself. Just remember to don't use serial communication in your sketch. It is explained in next step why it is so important.
- "My grandma has better programming skills than you" - maybe, but my used to make better pancakes
- I want to skip some process or abort immediately - press Ctrl+C.
- "How can I get new nodes firmware in the future?" - I will do my best to maintain the repository up-to-date. You could also make your own 'hex' files in process described above. "Self updater" should arrive at some point as well. EDIT: Just use "self updater" in Features menu. Nodes firmware version that updater contains always corresponds with Updater version number. For example. Updater version "2.2.1" has nodes firmware with "API level 22" etc.





## Step 10: Things I Had to Go Through / History of Development

At first I was trying to utilize the SPI protocol. I have connected spi1 (second SPI bus of the Raspberry Pi - first is occupied by LEDs) to Arduinos. After some hustle I was able to program the nodes but it was risky cause SPI protocol has ability to erase whole chip - bootloader included. So if something went wrong Arduino became useless. I had to pull it out and program with external programmer. Besides that 2 of 5 Arduinos used during testing are completely bricked now cause SPI protocol can change "fuses" of the chip. It is very low level programming stuff and can't be restored easily. The worst thing was that if SPI bus was connected to the Raspberry it couldn't be used to changing the channels of the 5808 receivers. I tried to use SoftwareSPI on Arduinos but it required changing pins assignment, more wiring etc. It was messy and unelegant. Besides, Arduinos had to be flashed with 'hex' files without the bootloader, files had to have additional delay during boot etc. Moreover special version of avrdude - program used for programming the Arduino - had to be used with special programmer compiled... Bad things. For me it was still worthy but I was worried that no one would want to do it too and that someone could brick Arduinos before race or something.

I then realized that Raspberry has an UART line and that Arduino normally can be very easily programmed with this protocol. Basically every time Arduino is connected via USB it uses UART - and USB to serial converter (this small chip at the bottom of the Nano. I have connected the dots and after some troubleshooting process - enabling the UART on the Pi is not that obvious - I successfully connected Raspberry's UART to PC's COM port. At that point I was sure that Arduino can be programmed this way. Luckily programming via UART is a standard way of flashing so normal version of avrdude can be used. Even the software programmer is called "arduino". Only thing left was to make a way to establish connection with each Arduino before programming so it can be ready to be flashed. Right now resetting the nodes is done "manually" in python script which is easy to control and doesn't require changes in

avrdude config files etc. Last important discovery was that serial connection has to be disabled in node code. It was caused because of the way how Raspberry GPIO operates and how it handles resetting Arduinos right now. The Serial communication is awesome feature of those nodes but in that case it shouldn't change anything - explained in third dot - below.

The biggest advantages of the way that this program operates right now are:

- bootloader stays on its place, Arduinos can't be broken (or odds are as small as with programming with the PC)
- even if power outage occurs during flashing it shouldn't cause any issues
- code on the nodes is exactly the same as standard code from RotorHazard - just with serial communication disabled - it isn't crucial unless you want to connect the nodes directly to the PC (but you probably won't if they are embedded in lap timer already)

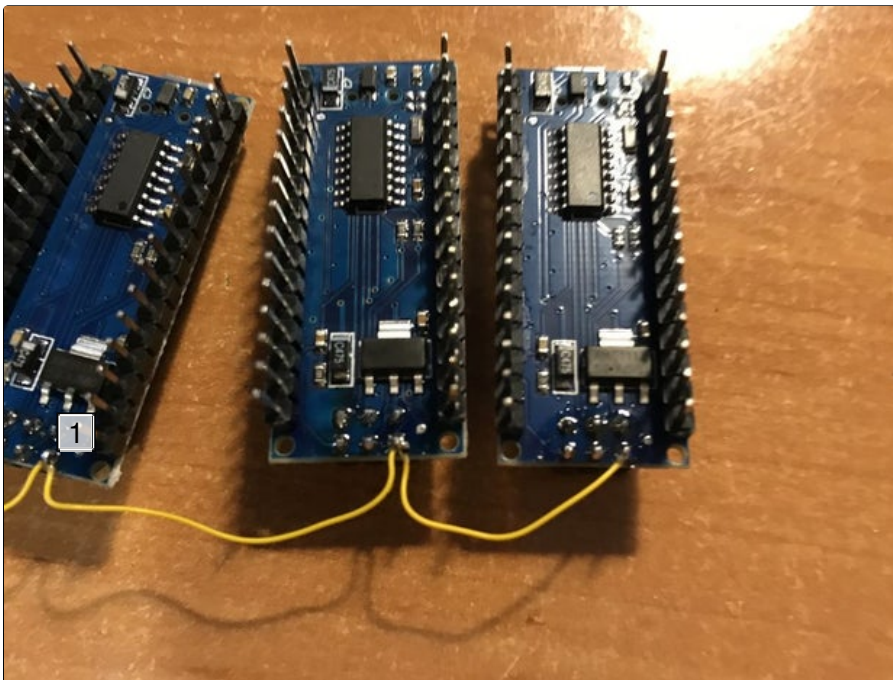
At the end I combined two separate programs so now you can update both - Raspberry's server and Arduino-based nodes using same program. Some additional features like making Raspberry as standalone Access Point automatically will come later.

And one more thing - I had absolutely zero knowledge or idea how programming in python works. I even didn't know what "indents" are - took me 1 hour to figure it out. I had some background with C language and Arduino IDE, but never did anything in python - so I learned the basics so this project can be developed :)

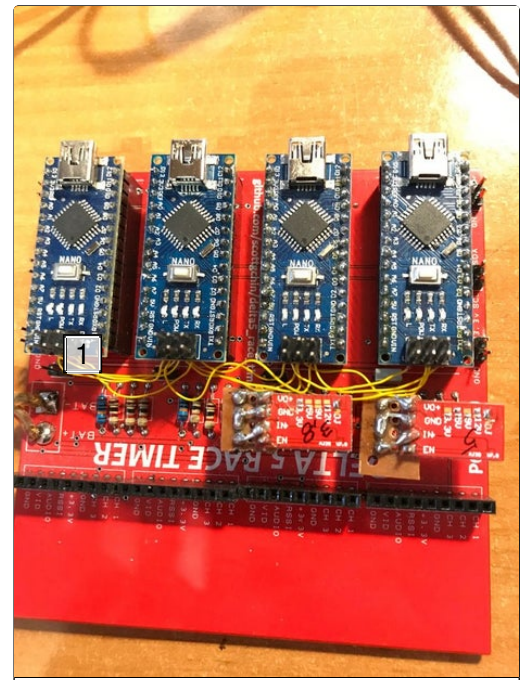
And I didn't know how GitHub works from creators point of view as well - so I've learned how to use desktop version - it really helps a lot.

Rodrigo Cardenas was a first reader of the instructables and beta tester for the first semi-official version of the software. He found many issues and helped me in process of debugging the code. We spent weekend on conversations and testing. It was worthy. Software looks awesome now and steps that have to be taken hardware-wise are well known - after about 50-70 hours of tinkering and testing.

And the story of this very instructables is quite funny. At first I was thinking about making some quick video or explain just a few steps. But someone mentioned that full written document would be preferable. And he got more than 20 likes on the Facebook group. So I had no choice :)



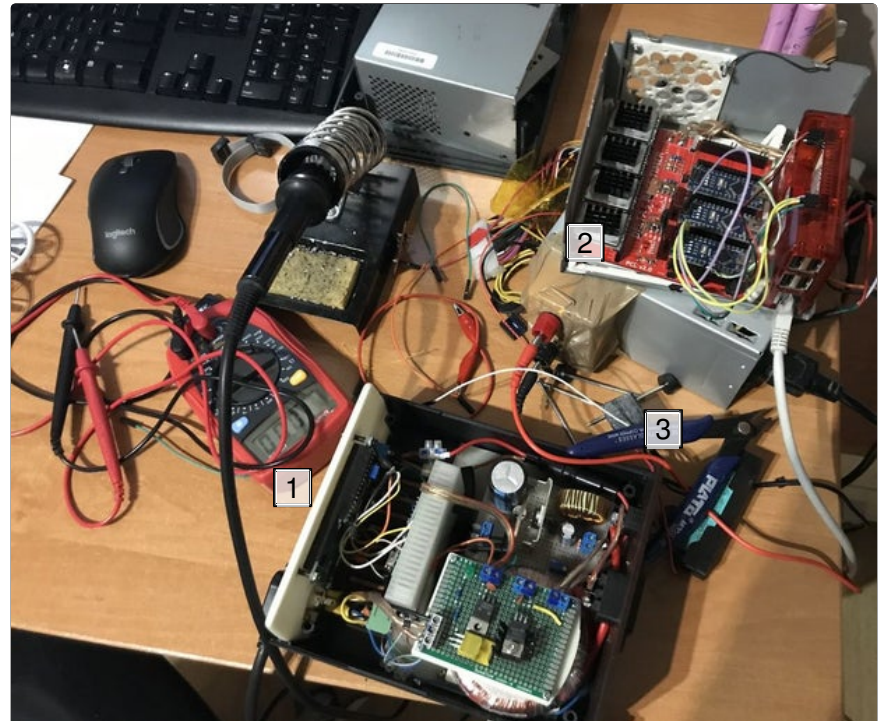
1. first attempts



1. SPI lines connected

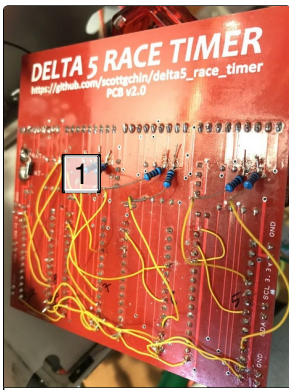


1. big mess

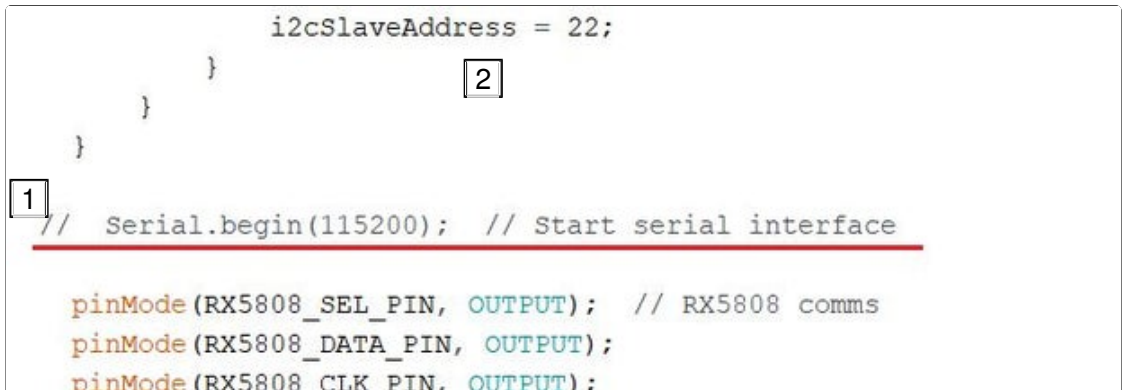


1. DIY soldering station  
2. test bench  
3. but those if you don't have them

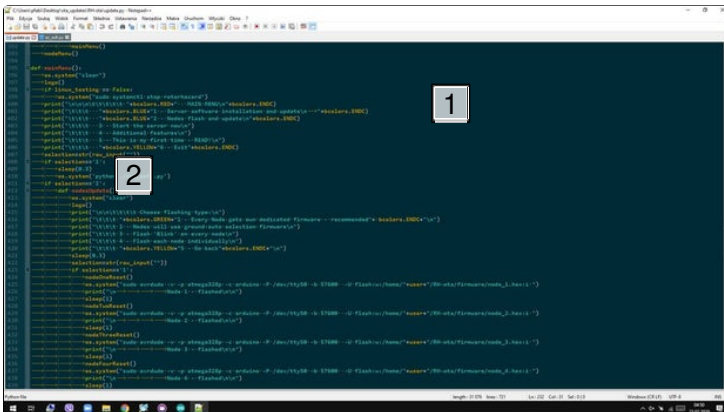




1. turning point - too much wires and changes



1. commented out  
2. rhnode.cpp file



1. fall in love with Notepad  
2. python

## Step 11: Afterthought - Added After First User Confirmed That It All Works:

Quick anecdote: My friend asked me yesterday if I did this project for free or will it be paid by anyone. I was surprised, but he told me that I put a lot of effort and time into it. I thought about it. So... If you feel this way you can make a donation. I have a PayPal account.

Link: [https://www.paypal.com/cgi-bin/webscr?cmd=\\_s-xcli...](https://www.paypal.com/cgi-bin/webscr?cmd=_s-xcli...)

[ [https://www.paypal.com/cgi-bin/webscr?cmd=\\_s-xclick&hosted\\_button\\_id=3J8LU6NDVN6WL&source=url](https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=3J8LU6NDVN6WL&source=url) ]

Remember that I am and will be happy to do my best to help you anyway. It is not conditional.

Also remember that I am not a part of an official RotorHazard developers team.

If right now you realized "Hmmm, RotorHazard itself also requires some support" - here is the link the the Patreon page:  
<https://www.patreon.com/rotorhazard>

Do what you feel - or don't :)

Cheers!



1. :)

---

## Step 12: Thanks + Disclaimer

Thanks for all the people from the RotorHazard Facebook group and developers and contributors (of Delta5 timer as well). This is my small contribution to the project but without you this awesome timer couldn't exist. Thanks for all advices to the people on the Group. You are awesome. Especially thanks for people that were willing to test this software and this mod and people who responded to my questions on Facebook and gave me some ideas (even random and general ones) and for the feedback or just a kind word and a little bit of enthusiasm.

BIG thanks for Rodrigo Cardenas for additional testing and being first user. And also for much better photos than mine :) Clayton Harp, Michael FPV and few other people from the Group helped me as well. Thank you!

Disclaimer: I've done my best to make this instruction as clear as possible. I am also sure that this procedure can't broke your electronics or software. I did lots of testing. Also one of the timers after this mod has proven to be working on some big race event, right after using both - server and firmware - obtained with usage of this software and this mod. From the other hand I don't have control over your soldering skills or special situations with software like unusual

Raspberry's configuration etc. You perform this mod at you own responsibility - but you can look for help on the Facebook Group if you need some. Speaking software-wise - of course if any bugs occur, I will attempt to fix them.

Feel free to give me the feedback and comment if you wish. Also you can make commits on GitHub page if you have some programming background. This project can be developed further and further! Even this instructable probably will be improved and few dozens additional insights has been added since first publication.

You can contact me here, using Facebook group or on GitHub platform.

Szafran - TheTinker

GitHub page of RotorHazard project:  
<https://github.com/RotorHazard/RotorHazard>

GitHub page of this project:  
<https://github.com/szafranski/RH-ota>



**RotorHazard**  
FPV RACE TIMING

1. <https://www.facebook.com/groups/rotorhazard>



Thanks for sharing :)



Thanks :) This instructable will be polished soon.