Pieter-Jan.com
Creativity In Automation & More

Getting the angular position from gyroscope data

Submitted by Pieter-Jan on Thu, 06/09/2012 - 02:41

In this article I will explain how I succeeded in finding the angular position (angle) of one axis of a quadrocopter by integrating gyroscope data. For this article I will use the gyrosocope from the Hobbyking HK401B module I hacked in a previous article. Because I wanted a fast prototyping method I used an arduino, but the principle is the same on any other device. I will include the code for a PIC microcontroller as well.

A little math

Before we take a look at the program we will go through the basic mathematics involved here, as they will make understanding the program a lot easier.

The gyroscope gives us **the rate of change of the angular position over time** (angular velocity) with a unit of [deg./s]. This means that we get the derivative of the angular position over time.

$$\dot{\theta} = \frac{d\theta}{dt}$$

However rate feedback is extremely useful in control engineering, it is usually used in combination with position feedback. To obtain the angular position, we can simply integrate the angular velocity. So, assuming that at t=0 theta=0, we can find the angular position at any given moment t with the following equation:

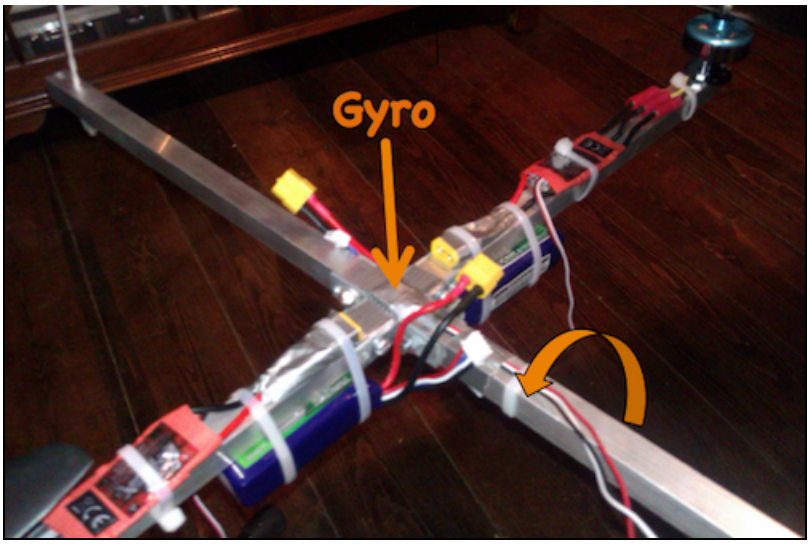
$$\theta(t) = \int_0^t \dot{\theta}(t)dt \approx \sum_0^t \dot{\theta}(t)T_s$$

The third part in this equation shows **the approximation we make when using digital systems**. Because we can't take a perfectly continuous integral, we have to take the sum of a finite number of samples taken at a constant interval Ts. Ts is called the sampling period. Of course this approximation will introduce errors. When gyrosocope data changes faster than the sampling frequency, we will not detect it, and the integral approximation will be incorrect. This error is called **drift**, as it increases in time. It results in the sensor reading not returning to 0 at the rest position. For this, it is important that we choose a good sampling period. The sampling frequency recommended for mechanical systems lies between 100 and 200Hz. I will use 100Hz (sampling period of 10ms) as this equals the cycle time of my quadrocopter PWM program.

Choosing a sampling frequency of 100Hz is a good choice for an "ideal case" mechanical system. It has to be understood though that **disturbance inputs** like shocks & vibrations because of the motors can have a much higher frequency and will not be measured properly. This is mostly the case for disturbances that work directly on the sensor, and it is a big problem of this integration method. Disturbances like wind and external forces that work on the system (quadrocopter) will be slowed down by the intertia of this system so that they will be measured correctly and can be compensated for.

The gyroscope sensor

This sensor was already discussed briefly in a previous article. It is an **analog sensor**, which means that it will output an analog voltage that will be a measure for the angular velocity. It is also a **one axis** gyroscope, which means that there is only one output. The axis that this sensor can measure falls together with the axis of the wires. It is shown in the picture below.



When the angular velocity is 0 deg./s, the sensor will output a voltage that is close to half of the supply voltage. A negative velocity will decrease this voltage while a positive velocity will increase this voltage. In the previous article I put a reference to a datasheet, but it doesn't seem to be correct. After 50 mails to Hobbyking, I gave up finding the datasheet, and just found out my values by trial/error. This what I came up with at a supply voltage of 5V:

- Offset (output at angular velocity = 0): **2.65V**
- Sensitivity (scale factor): **107.42 mV/deg./s**

The value for the offset can be measured with a multimeter. To find the value for the sensitivity, I wrote the arduino program that is shown below, and used the Serial monitor to check the correctness. E.g. hold the quadrocopter at 90 deg. and see if the measurement is correct or not. Trial/error! The weird number "107.42" was a result of the conversion from digital to analog.

The arduino program

Let's take a closer look at the arduino program now. If you are using the gyroscope from the HK401B like me, you can connect it by putting 5V between the black and red wires of the gyro (or whatever colors you soldered to the supply pins) and by connecting the white (signal) cable to the A0 analog input. If you are using another analog sensor, it should be easy enough to figure out how to connect it! The whole program works with 8 bit values. This means that the analog offset value of 2.65 results in a digital value of 136:

$$\frac{2.65}{5}256 \approx 136$$

The same principle would apply to the sensitivity if it was given in the datasheet. I found this value by trial/error so I knew it was 5.5 digitally.

However the arduino has a 10-bit ADC, I only use the 8 most significant bits. The last two bits of the ADC are usually not used because they contain a lot of noise and it is a lot more efficiënt to work with 8-bit values on an 8-bit processor ;).

It should be very easy now to see how I converted the approximated integral equation we saw earlier into an extremely simple program. The program runs at 100Hz. Every iteration we read the sensor, and convert its value into an 8-bit digital number (gyroValue). After this, the offset is removed and the gyroValue is being converted to [deg. /s] by multiplying it with the sensitvity. Finally we multiply it with the sampling period (*0.01 or /100) and add it to the angle, which is the sum (approximated integral) of all previous values.

It is very important to take a look at the datatypes you are using. I recommend using a float for the angle and the rate. The reason for this is that if you make more approximations by rounding off to another datatype, the drift would grow (extremely) fast. The only way this integral approximation will work, is by doing it very precise.

```
unsigned char gyroValue = 0;
unsigned char offset = 136;
float angle = 0;
float gyroRate;

void setup() {
  Serial.begin(9600);
}

void loop() {
  gyroValue = analogRead(A0) >> 2;   // Work with 8 most significant bits!

  gyroRate = (gyroValue - offset) * 5.5; // Sensitivity has been found trial/error
  angle += gyroRate / 100.0;

  Serial.print(angle);
  Serial.print("\n");

  delay(10);
}
```

The PIC program

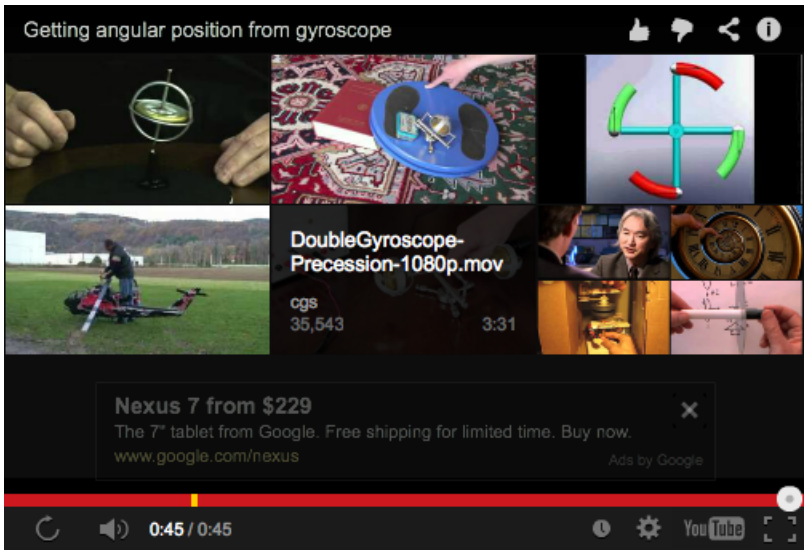
To demonstrate how easy it is to port this code on another platform, I included my implementation for the PIC16F690. A 20MHz crystal was used, and the gyro is connected the RA0-pin. I make use of the HI-TECH C Lite compiler.

```
#include <pic.h>
#include <pic16f690.h>

__CONFIG(FOSC_HS & WDTE_OFF & PWRTE_OFF & MCLRE_OFF & CP_OFF);

#define _XTAL_FREQ 20000000

/* Function prototypes */
void ADC(void);
void main(void);

unsigned char gyroValue;
unsigned char gyroOffset = 136;
float gyroRate;
float angle = 0;

/* Main program */
void main(void)
{
    // Init
    TRISA = 0b00000001;     // RA0 input (Analog)

    ADCON0 = 0b00000001;    // AD conversion init RA0 = input
    ADCON1 = 0b00000000;

    while(1)
    {
        // Read Gyro
        ADC();
        gyroValue = ADRESH;

        gyroRate = (gyroValue - gyroOffset) * 5.5;
        angle += gyroRate / 100.0;

        __delay_us(10000);      // 10 ms
    }
}

void ADC(void)
{
    GO_DONE = 1;
    while(GO_DONE);
```

```
}
```

Results

Here is a little video of the program in action. It is very clear to see the problem of **drift** introduced by approximating the integral by a sum. I also gave the sensor a few ticks with my finger to show that the drift is much stronger as a result of external forces that work directly on the sensor (e.g. motor vibrations, shocks, etc.).



UPDATE: To solve the drift error, we can combine the gyroscope data with accelerometer data using a **Complementary Filter**. This is described here.

Tags:
Quadrocopter

Grohar
Wed, 12/12/2012 - 13:08
permalink

Hello, how can i find Offset

Hello, how can i find Offset and Sensitivity?

reply

Pieter-Jan
Wed, 19/12/2012 - 12:47
permalink

For which gyro?

For which gyro?

reply

robertus
Sun, 14/04/2013 - 11:15
permalink

I have a problem when

I have a problem when compilling... at

Serial.print(intValue);
Serial.print("\n");

reply

Pieter-Jan
Tue, 16/04/2013 - 11:08
permalink

Aha, sorry. intValue has to

Aha, sorry. "intValue" had to be "angle" of course. I changed that in this post to make the code a little bit more readable but didn't test it afterwards. It's fixed now ;). Let me know how this works out for you!

reply

Dragos George
Tue, 16/07/2013 - 14:57
permalink

very good guide how to use

very good guide how to use gyro and accelerometer sensors. I add this tutorial on my article where a long list with sensors and tutorials about how to interface and programming accelerometer, gyro and IMU sensors
http://www.intorobotics.com/accelerometer-gyroscope-and-imu-sensors-tuto...

reply

chi-wei wu
Thu, 01/08/2013 - 01:11
permalink

very appreciated!

very appreciated!

reply

Sigit Dermawan
Wed, 25/09/2013 - 21:36
permalink

Can u explain how to write

Can u explain how to write that integral into detail program?

reply

saied
Tue, 22/10/2013 - 15:43
permalink

hi,can u explain what is rate

hi,can u explain what is rate integrating gyro(RIG)?
output pick off is mv/degree but input torquer is degree/second/ma.....is derivative in servo??
very TX

reply

Setting up the Raspberry Pi for programming from Mac OS X

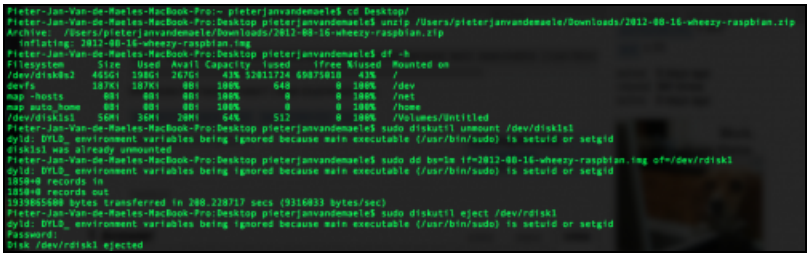
Submitted by Pieter-Jan on Mon, 03/09/2012 - 13:49

In this article I will explain how I set up my Raspberry Pi to be programmable from my macbook pro. I didn't want to use an external monitor & keyboard, nor did I want to connect my Pi to the router of my home network as this setup would not be very mobile. To achieve this, I made use of SSH and the mac's ability to share its internet with other computers. I will not use any GUI to do the programming, as I find that running a GUI on a small embedded computer like this is overkill. Instead I will do all the programming from the command line. A small example of how to do this is included here as well. This article was mainly written as a future reference for myself, but I hope it can be useful for others as well.

Step 1: Prepare the SD-card

First of all, you will need an SD-card with a bootable linux image. I recommend using the "Raspbian Wheezy" image, which is an optimized version of debian. It is downloadable from http://www.raspberrypi.org/downloads/. Instructions for putting the image on the card properly

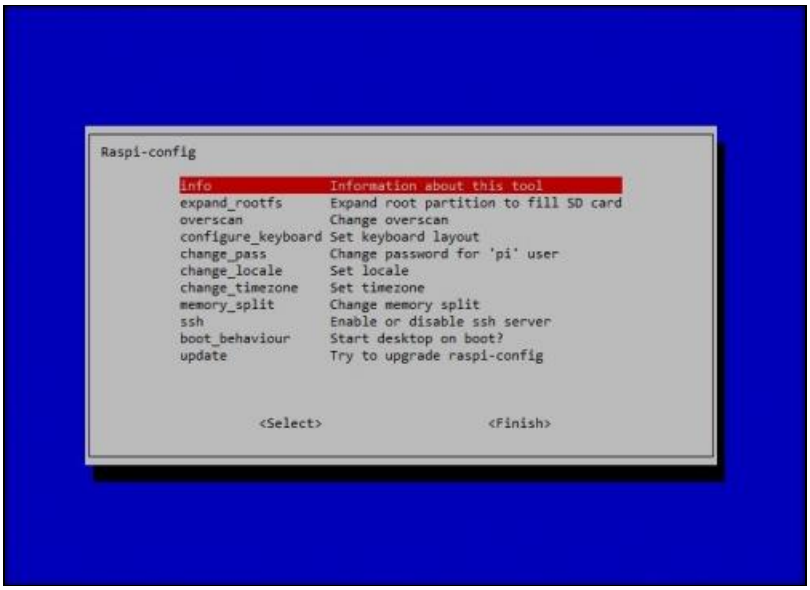
can be found on the eLinux website.



If this has succeeded, you can boot up your Raspberry Pi! For the initial boot we will need an external monitor & keyboard, so we can set everything up properly.

Step 2: Basic configurations

The easiest way to do the basic setup is by making use of the **raspi-config tool**, it should start up automatically on your first boot.



If it doesn't start automatically, you can also launch it with:

```
sudo raspi-config
```

I would recommend starting off by setting **configure_keyboard** and **change_timezone**. As we won't use any GUI for our programming, it is a good idea to set up **memory_split** so that the ARM gets 240MB of RAM and the VideoCore only 16MB as this will speed up things a lot. If you have a large SD-card (> 4G) you might want to consider using **expand_rootfs** to enable all this space. The downside of this is that backups of your SD image will be significantly larger.

The most important thing we had to do here, was to enable **SSH**. The more recent versions of raspbian have this enabled by default though.

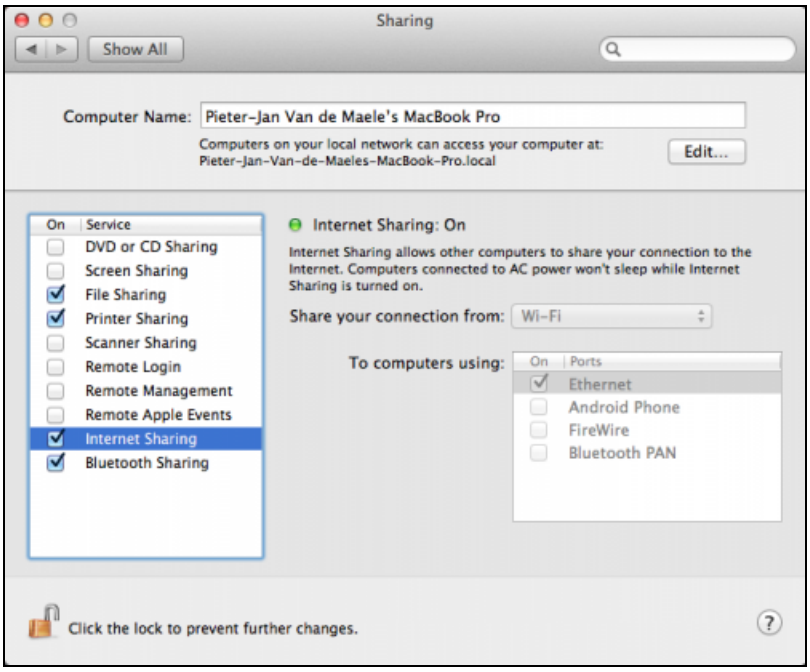
If you have used configure_keyboard, you should run:

```
sudo setupcon
```

to avoid a massive delay in the boot process.

Step 3: Find out the Raspberry Pi's IP adres

At this point, we need our Pi connected to the internet. If you have a router close to your working environment, plugging it in there would be fine. If you don't however, it is very easy to use the "internet sharing" option on a mac to provide internet to your Raspberry Pi. On your mac, go to: "System Preferences > Sharing".



As you can see, I chose to share my macbook's wifi through the ethernet port. If I plug in my Raspberry there, it will have internet.

Now go back to the Raspberry Pi's monitor & keyboard. To be able to connect to the Pi through SSH, we need to know the IP adres it is using. To find it out, we enter the following command:

```
ifconfig
```

You should see something like "inet addr: 192.168.2.2".

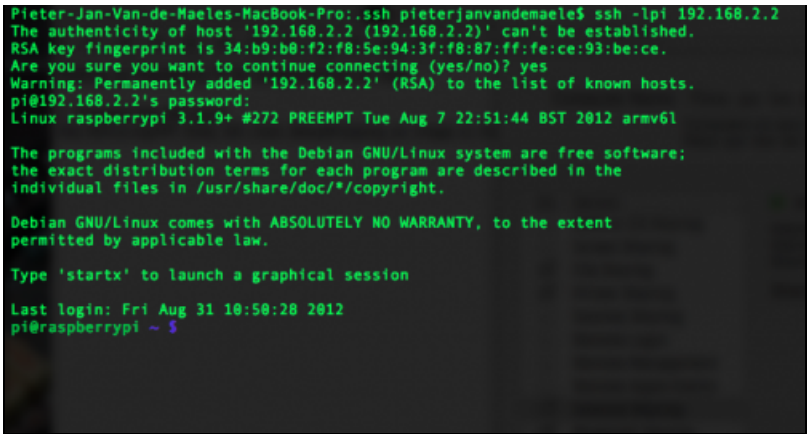
Step 4: Connect to the Raspberry Pi from your mac

Now we know everything we need to know and everything is set up properly, we can disconnect the monitor & keyboard from our Pi (forever). To connect establish the SSH connection, go to the mac terminal and type (you should of course change the IP with the one you found earlier):

```
ssh -lpi 192.168.2.2
```

If it is the first time you connect to this device, you will probably see something like "The authenticity of the host can't be ... blabla". You can answer "yes" to this.

You will be prompted for the password now. If you did not change this in step 2, and you are using the recommended raspbian image, the password is "raspberry". Enter this, and you are now logged in to the Pi! You should see something like this:



Step 5: Setup a basic programming environment

You don't need much to programming from the command line in linux, but you do need a good text editor. I use vim. If this is the first time you work with vim, I recommend taking 10 minutes time to check out a good tutorial. To install vim on your raspberry pi, simply type:

```
sudo apt-get install vim
```

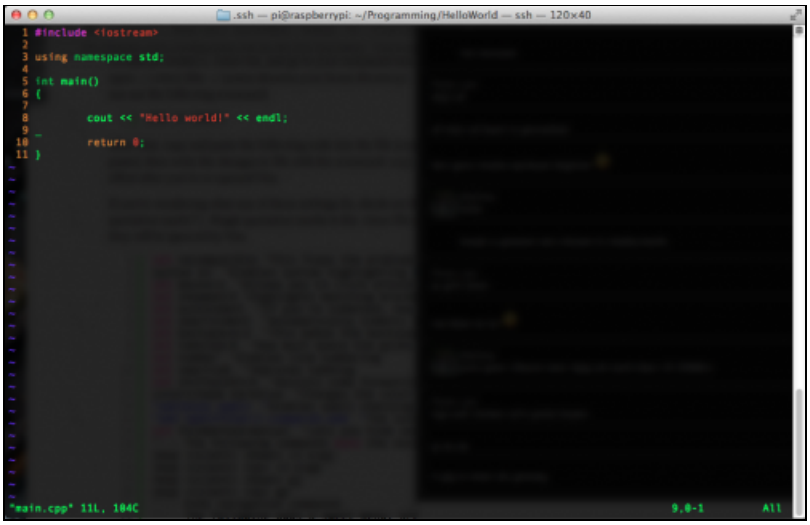
Because I think **syntax highlighting** and **line numbering** are essential programming tools, I want to enable these by default. You can do this by editing (or creating) the .vimrc file in your user directory. We'll do this using vim:

```
vim ~/.vimrc
```

Then add the following two lines:

```
syntax on
set number
```

Now it's time to test our tools! If you write a very simple "Hello World!"-program with vim, it should look like this:



If you name the file "main.cpp", you can compile it with:

```
g++ main.cpp -o HelloWorld
```

If you then run it with:

```
./HelloWorld
```

you can sit back, and enjoy the most simple program in the world!

If you're interested in doing some more advanced programming of your Raspberry Pi, check out my article on Low Level Programming of the Raspberry Pi.

Tags:
Raspberry Pi

Ricardo Anaya
Wed, 05/09/2012 - 10:44
permalink

Hello,

Hello,
I found your article about "Setting up the Raspberry Pi for programming from Mac OS X" very interesting, but I have not been able to connect by ethernet my "Raspberry Pi" with my "Mac Pro". It seems that the configuration to share an "Internet connection" is more complicated than what you've shown.
Can you help me?

My configuration :
- LAN with router and DHCP server (ADSL BOX)
- Mac Pro with "OS X Mountain Lion" and Wifi connection with DHCP client (192.168.0.xx)
- ethernet cross-over cable between "Mac pro" and "Raspberry Pi"

reply

Pieter-Jan
Wed, 05/09/2012 - 12:21
permalink

Can you be more specific

Can you be more specific about the problems you are having with my approach? I have checked this on another computer and it works just fine. Are you sure you are using the correct cables etc.? Thanks

reply

Ricardo Anaya
Thu, 06/09/2012 - 08:48
permalink

Yes, the cable is OK!

Yes, the cable is OK!
When I connect it to the BOX/ADSL, I get an IP address automatically via DHCP: 192.168.0.35.
When I connect it to the Mac, the ethernet connection on the Mac gives me the following IP address 169.254.32.59 with the subnet mask 255.255.0.0, and, of course, I followed the "Step 3". But I don't get an IP address on the "Raspberry Pi"!
Below is a copy of the result of the command "ifconfig" (I masked the MAC address)
pi@raspberrypi ~ $ ifconfig
eth0 Link encap:Ethernet HWaddr xx:xx:xx:xx:xx:xx
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:90 errors:0 dropped:0 overruns:0 frame:0
TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 lg file transmission:1000
RX bytes:11865 (11.5 KiB) TX bytes:7931 (7.7 KiB)

reply

Pieter-Jan
Tue, 11/09/2012 - 03:50
permalink

After I made a short-circuit

After I made a short-circuit on my raspberry pi, I had the same problem as you. I solved it by turning the "internet sharing" option on my mac off and on again. It worked instantly. Let me know if this works for you too. Sorry you had to wait so long for another reply from me. Maybe you have to connect your Raspberry Pi to the mac first, and then enable internet sharing?

reply

Tito
Fri, 28/12/2012 - 16:49
permalink

Any way to get started with

Any way to get started with Pi without ever using an external monitor? All I have are laptops in my home, no external screens. Suggestions welcome :):)

Thanks!

reply

Bjorn Volkers
Fri, 22/03/2013 - 12:24
permalink

Pieter-Jan, you must be from

Pieter-Jan, you must be from Holland. So am i, so will type in Dutch:

Ik heb problemen ervaren.

Ik ben nu bij stap 5. Ik wil VIM installeren op mijn Raspberry Pi door "sudo apt-get install vim" in te typen in de XL Terminal van de Raspberry. Als ik dat doe geeft de Raspberry aan wat hij wil installeren en vervolgens krijg ik de optie om door te gaan of te stoppen door "Y" of "N" in te typen. Als ik "Y" typ geeft ie aan dat pagina "404 not found" melding. Hier loop ik dus vast.

Ik heb evt. screenshots (foto's) gemaakt van de meldingen die ik je mogelijk kan mailen.

Je tutorial is duidelijk en ik waardeer het. Maar ik hoop dat je me verder kunt helpen.

Alvast bedankt!

B

reply

Bjorn Volkers
Sat, 23/03/2013 - 08:22
permalink

I apologize! You're Belgian!

I apologize! You're Belgian!

I finally fixed my previous problem, but now i have another one:

Its step 4. when it asks my password (which is still 'raspberry') in the terminal i can't type! So now im not able to enter the password! Please help!

Thanks in advance,

B

reply

Pieter-Jan
Thu, 28/03/2013 - 07:39
permalink

Hello, sorry for my late

Hello, sorry for my late reply. If debian asks you for a password it will not show the characters you are typing. Just type the word and press enter. It should work.
reply

Raphaël
Thu, 08/08/2013 - 04:40
permalink

Hello Pieter,

Hello Pieter,
After hours of search, I am really glad to find your tutorial. Just what I'm trying to do.
I'm traveling, with a macbook Pro and a Raspberry in my luggage, trying to connect both of them without success through an ethernet cable. (wifi Internet connection).
... But I'm still having the same problem Ricardo Anaya told you, even after your advice of " turning the "internet sharing" option on your mac off and on again ". The IP address is still something like " 169.254.32.59 with the subnet mask 255.255.0.0 ".
The problem also is that I can't connect the Raspberry to another monitor to find out what it's IP is... is there a way to find it through the macbook Pro ? :/
Thank you for your help :) ! ®

reply

Tino Kanngießer
Sun, 24/11/2013 - 17:56
permalink

Hello, does anyone know how I

Hello, does anyone know how I need to configure the Mac keyboard in the Raspberry PI configuration in order to use all keys e.g. to use square brackets, etc.?
Many thanks in advance, Tino

reply

Al
Sun, 24/11/2013 - 23:52
permalink

Hi! I searched for hours on

Hi! I searched for hours on this. Do you know if there is a way to now ssh to the pi via a computer on the wifi network (using the internet sharing)? I have my imac wired to the pi and it works fine with the same configuration as yours. I am wondering (and hoping) if I can cross the subnets, since the pi is in 192.168.2.x, and all other computers are in 192.168.1.x. Your help is greatly appreciated.
Al

reply

Universal/Jamming Gripper Prototype

Submitted by Pieter-Jan on Mon, 03/09/2012 - 12:25

I made my own version of the Universal/Jamming Gripper created by John Ahmend.

You can do this too in 5 minutes! The recourses you need are:

- **Plastic syringe** (ask your pharmacist, he/she will look weird but give it to you if you don't look like a drug addict)
- **Balloon** (a small one)
- **Ground coffee**
- **Duct tape**
- **Piece of cloth/woven tissue** (should act as a filter to let the air pass but not the coffee so keep this in mind)
- **Superglue**

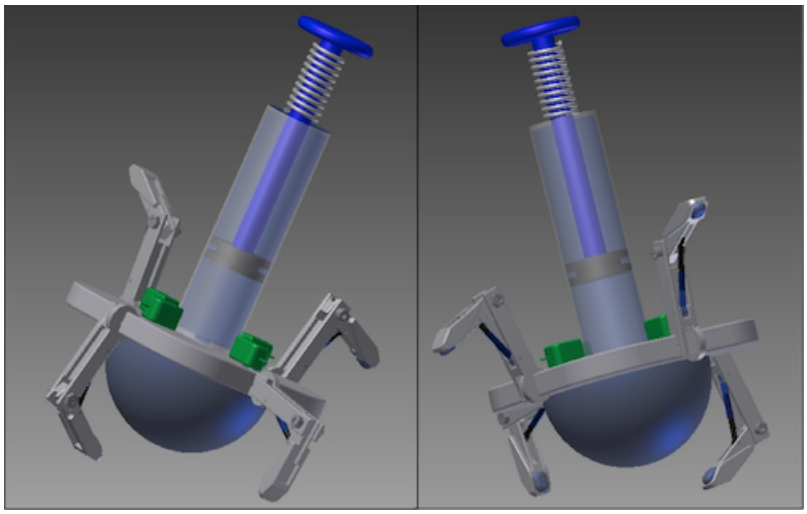
To assemble it you have to go through the following steps:

- **Step 1** - Cutt off the tip of the plastic syringe as cleanly as possible.
- **Step 2** - Cover the hole that is now in the syringe with the piece of hankerchief and glue it reasonably tight with the superglue. Let it dry.
- **Step 3** - Fill the balloon as good as you can with the coffee. This is a difficult step! It might help if you fill the balloon with air a few times so that it stretches.
- **Step 4** - Make sure the syringe is closed so that it does not contain any air. Then pull the neck from the balloon over the syringe, and make sure you don't trap any air. If you do trap air, try to get it out as much as possible as it will drastically decrease the performance of your gripper.
- **Step 5** - Tape the balloon and the syringe together as tightly as possible using the duct tape.

Here is the gripper in action!

The cool thing about this gripper is that you **don't need a pump or anything pneumatic**. So if you could find a way to professionally make this, and actuate the seringe with a **solenoïd**, you can turn this into a very compact and fast device which has all the actuators on board. This would make it fully electrically controllable.

I quickly sketched up an idea I had for this in which I added some extra "fingers" which would make the universal gripper even more universal ;). The first part of the finger would be driven with a small motor and a worm wheel to block it when the motor is not on. The second part part of the finger would be driven by small linear actuators. I never really built this, it's just an idea.



Tags:
Mechanics Grippers Robotics

monterreymAchito
Wed, 05/06/2013 - 11:16
permalink

I love it, very simple. You

I love it, very simple. You should post this on the instructables and enter one of the many open contests.

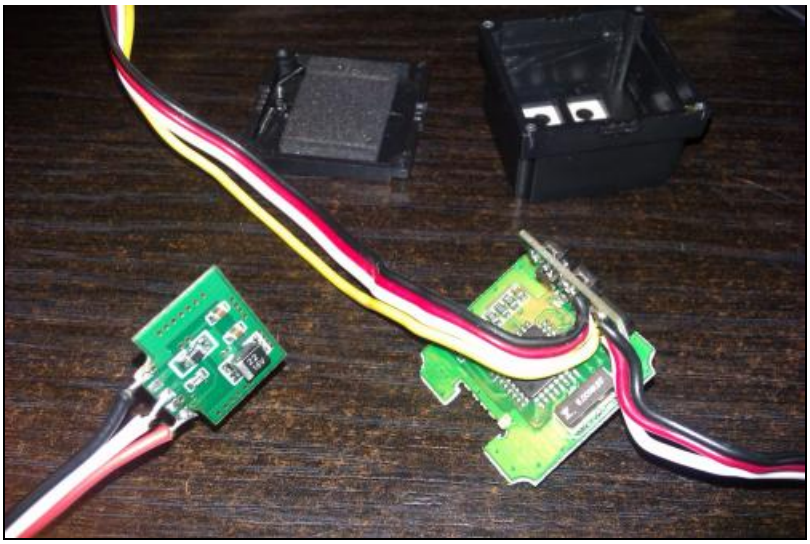
reply

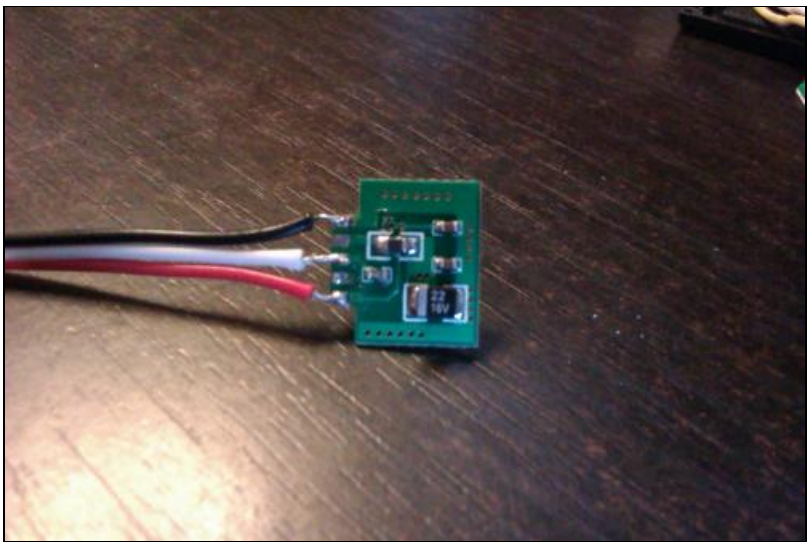
Hacking the Hobbyking HK401B gyroscope module

Submitted by Pieter-Jan on Sat, 21/07/2012 - 10:12

For the quadrocopter project we are doing, we bought a hobbyking HK401B gyro module. This module includes a **one axis analog gyroscope** and a full microcontroller circuit, that already processes the gyroscope data. It sends out the "corrections" the motors should make and can be connected directly to the typical RC-setup.

Because we are not interested in processed data, we decided to **break open** the module, and **desolder the gyroscope**. Here are some pictures of the proces:





As you can see on the pictures, the HK401B consists of **three parts**. The gyro part is the one that has initially **no wires attached to it** (in the pictures above I already added wires though). Once you desolder this part, it is sufficient to add the supply voltage (black and red wires: 5V - check the picture to see which is which) and a third wire for the signal (white). The axis of rotation that the gyro can "sense" is in the same direction of the wires. The following video shows the signal that can be expected:

This datasheet seems like it might be the right one.

Tags:
Quadrocopter

Four (4) independent PWM-signals generated with one PIC

Submitted by Pieter-Jan on Tue, 17/07/2012 - 11:55

Every self-respecting engineer has at least one moment in his or her life when it seems like the right thing to do is **build flying stuff**. In the case of me and a few friends, this means that we are going to do an attempt at creating our own quadrocopter. We quickly started looking around the internet, and placed an order at http://www.hobbyking.com/ (I will describe the contents of this order in a later post once it has shipped to my doorstep).

Because the shipping would take a few days, and another property of the self-respecting engineer is that he does not have any form of patience when it comes to building stuff. We decided that we would already start designing the control circuitry, which we will try to do from scratch. A first step in this is **finding a way to control the speed of the motors**.

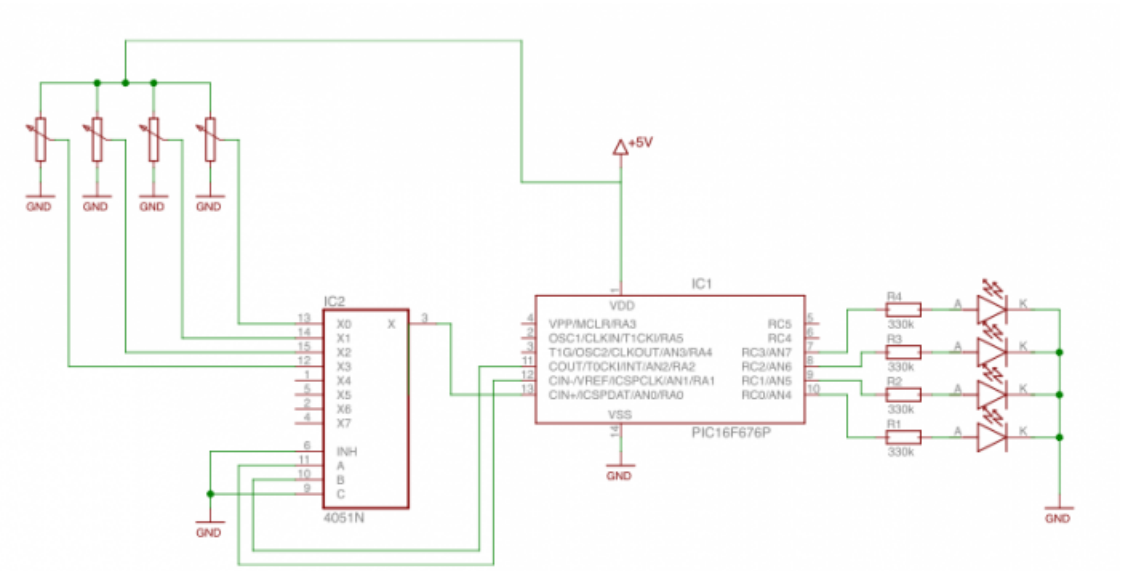
The Signal

Quadrocopters are built with brushless DC motors. A common mistake is to assume that these are conventional DC-motors, and can be driven as such, which is far from true!! Actually, **a brushless DC motor is a permanent magnet AC motor with a block waveform**. I won't go into detail, but this is important stuff! Because of this AC-block waveform we can't just drive this motor with a normal H-bridge, other circuitry is required. It would be very possible to build this ourselves, but the challenge is pretty low and because this is part of the power-circuit, the risk of introducing power-loss is too high. This makes that we use what the industry has provided for us: the **ESC** (Electronic Speed Controller). We can just hook up our motor and our battery to the ESC. If this is driven with a specific PWM signal, the speed of the motor will vary. This is what the PWM signal looks like:

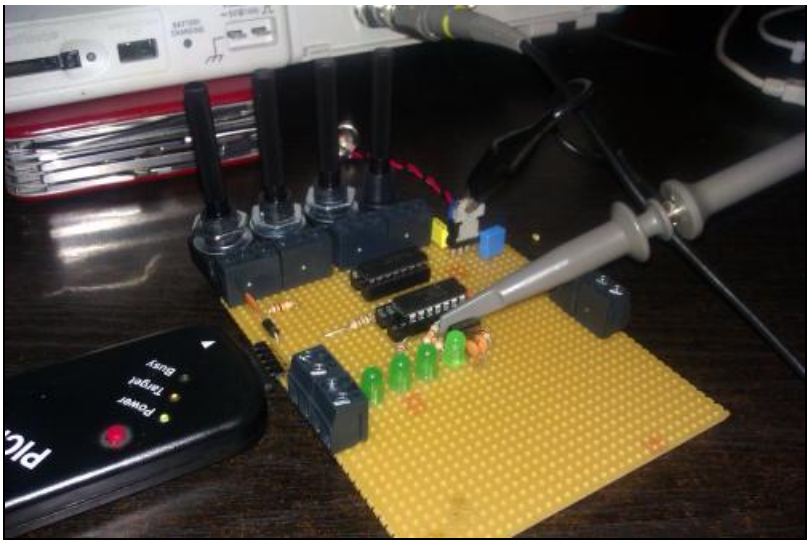
First send a 1 ms pulse, then send the second pulse that controls the speed (max. speed = 1ms, min. speed = 0ms) and then wait until the total period is about 10ms. The waiting time can be longer than 10 ms because it would be hard to do this with RC otherwise. It is only the pulse length that is important. From this information, we know that we have to **generate a PWM-signal of 100Hz (period 10ms) with a duty cycle that is variable between 10% and 20%**. We will have to be able to do this 4 times independently, because a quadrocopter has four motors that have to be able to operate at different speeds (duh!).

Prototype

As the title said, we created this PWM-controller using a PIC microcontroller. The one we chose was a very small one: a **PIC16F684 with 128 bytes of RAM and 2048 words of flash**. Because we don't have any RC-setup yet, the inputs that control the motor speed are 4 potentiometers that are read in with an analog multiplexer (74HCT4051). This saves us a pin on the MCU and makes it a little easier to program the ADC unit properly. Also when we want to add accelerometer or gyro modules later with analog signals, we can easily do this without losing pins (except for one extra adres pin ofcouse!). The full schematic looks like this:



Here's a picture of the prototype:



The four LED's are at the output pins (where the ESC's will be attached to) and give some visual feedback during prototyping (the intensity of a LED changes when the duty cycle is modified, duh!).

We could have chosen a PIC that has 4 independent PWM channels like the PIC18F4x31. Why didn't we? Because we believe that this is one of the simplest parts of making a quadrocopter and it shouldn't have any impact on the choice of your hardware. It's also a nice exercise in efficiency to work with low resources. If we want to cram the full quadrocopter control-software in a PIC one day, we might want to start thinking efficient from day one. The code I provide here is not 100% optimal, but I did my best (let me know if you have any improvements!).

Program Structure

The hardest thing about writing the code was finding a way of outputting the 4 PWM signals with varying duty cycles at the same time . It is possible that motor 1 had to go faster than motor 2 at first (motor 1's pulse is longer than motor 2's pulse), but when the direction ischanged, motor 2 has to go faster than motor 1. This made it a little more awkward to generate the 4 PWM's at the same time. A little creativity though, easily solved this problem by **sorting the pulses from short to long** so that they could be turned off sequentially. Because there are 3 obvious parts to the signal shown above, I decided to **cut my code into three big pieces**. This is how the program was organized:

Part 1: 0 - 1 ms

- Set all output pins to high.
- Read the 4 potentiometers and convert them to digital values using the internal ADC of the PIC. The values are stored in the ADCValues table.
- Sort the ADCValues table from low to high and keep track of which output corresponds to which value (whichMotor).
- Wait until the first millisecond has passed

Part 2: 1 - 2 ms

- Wait until first output pin has to be turned off, and then turn it off.
- Do this for the other 3 pins as well. - Wait until the second millisecond has passed.

Part 3: 2 - 8 ms

- Wait until 8 ms have passed.

The C program

I make extensive use of the internal timers (timer 1) of the microcontroller because this gives a lot of control over the actual time passed, and

you can do other stuff in the meantime instead of just waiting like with a delay function. The time that passes for a value of the TMR1-register can be easily calculated with: time = 1/OscValue * 4 * Prescaler * (65536 - TMR1). A very useful tool for this is the PIC timer calculator. As you can see in the code underneath, the remaining program has to be put in the two while loops. I indicated this with "// Do Something" in the comment.

I make use of the MPLABX IDE and the Hi-Tech C Lite compiler.

```
#include <pic.h>
#include <pic16f690.h>
#include <math.h>

__CONFIG(FOSC_HS & WDTE_OFF & PWRTE_OFF & MCLRE_OFF & CP_OFF);

#define _XTAL_FREQ 20000000

/* Function prototypes */
void ADC(void);
void main(void);

/* Global variables */
unsigned char ADCValues[4];
unsigned char whichMotor[4];
unsigned char help;

static int accData[3], gyrData[3]; // AccX ; AccY ; AccZ ; GyrX; GyrY; GyrZ
//SPid pitchPID, rollPID;
static int pitch = 0, roll = 0;

/* Main program */
void main(void)
{
    unsigned char i, j;
    unsigned char PORTAValue = 0b00000000, PORTCValue = 0b00000000;

    // Init
    TRISA = 0b00000001;     // RA0 input (Analog), RA1 & RA2 output (MUX adres)
    TRISC = 0b00000000;     // Port C output (PWM Signals)

    ADCON0 = 0b00000001;    // AD conversie init
    ADCON1 = 0b00000000;

    PEIE = 1;               // Peripheral interrupt enable
    GIE = 1;                // Global interrupt enable
    TMR1IE = 1;             // Timer 1 interrupt enable
    T0IE = 1;

    TMR1IF = 0;

    T1CON = 0b00000000;     // Timer 1 prescale 1:1

    // Infinite loop
    while(1)
    {
        // ------------------------------------------------------------------ //
        // -------------------------- MS 0 to MS 1 -------------------------- //
        // ------------------------------------------------------------------ //

        // Set all the pins high to start the PWM
        PORTCValue = 0b00001111;
        // Keep track of current state of the ports in a register to walk around
        // the RMW-problem
        PORTC = PORTCValue;

        // Start Timer 1 ms (time = 1/OscValue * 4 * Prescaler * (65536 - TMR1))
        TMR1 = 60536;   // 1 ms with prescaler 1:1 and freq 20MHZ
        TMR1ON = 1;

        // Get ADC value of all input POT'S
        i=0;
        for(i; i < 4; i++)
        {
            // Set MUX adres on RA1 and RA2
```

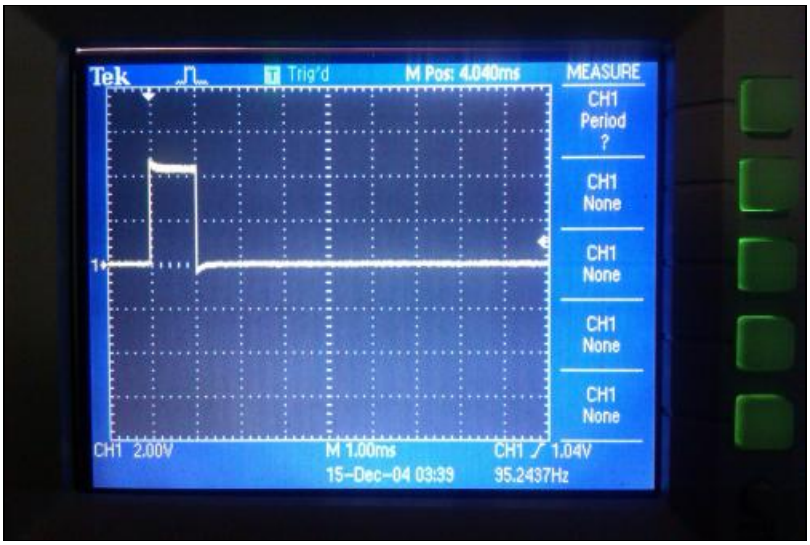
```
        // without affecting current state of PORTA (READ MODIFY WRITE)
        PORTAValue &= 0b11111001;
        PORTAValue |= (i << 1);
        PORTA = PORTAValue;

        __delay_us(1); // Delay here to wait until MUX value is on the pin
        ADC();
        (ADRESH <= 250)? ADCValues[i] = ADRESH : ADCValues[i] = 250;
    }

    // Initialise the whichMotor table
    i = 0;
    for(i; i<4; i++)
        whichMotor[i] = i;

    // Sort the ADC values from low to high
    // (this is the order in which they will be turned off)
    i=0;
    for(i; i<4; i++)
    {
        j = i+1;
        for(j; j < 4; j++)
        {
            if(ADCValues[j] < ADCValues[i])
            {
                help = ADCValues[i];
                ADCValues[i]=ADCValues[j];
                ADCValues[j]=help;

                help = whichMotor[i];   // Keep track of output pins
                whichMotor[i] = whichMotor[j];
                whichMotor[j] = help;
            }
        }
    }

    // Calculate the values that have to be loaded in the timer
    unsigned int delayValues[5];
    i = 0;
    delayValues[0] = 65535 - ADCValues[0]*20;
    for(i; i<3; i++)
    {
        // time = 1/OscValue * 4 * Prescaler * (65536 - TMR1)
        // ADC value is scaled to 1/4 of 1000 (value * 4)
        delayValues[i+1] = 65535 - (ADCValues[i+1] - ADCValues[i])*20;
    }
    delayValues[4] = 60535 + ADCValues[3]*20; // remaining time to 1 ms

    // Wait until the first ms is passed
    while(!TMR1IF)  // Wait until the timer overflows
    {
        // DO something
    }
    TMR1IF = 0;     // Must be cleared in the software
    TMR1ON = 0;

    // ------------------------------------------------------------------- //
    // ------------------------- MS 2 to MS 3 ---------------------------- //
    // ------------------------------------------------------------------- //

    // Turn off the outputs in the right order: ~ is binary invert
    // Wait until first output has to be turned off
    // whichMotor holds the index of the current output pin
    // (bitshift inverted 1 this many places to turn that output off)
    i = 0;
    for (i; i<4; i++)
    {
        TMR1 = delayValues[i];
        TMR1ON = 1;
        PORTCValue &= ~(0b00000001 << whichMotor[i]);
        while(!TMR1IF);
        TMR1ON = 0;
        TMR1IF = 0;
        PORTC = PORTCValue;
    }
```

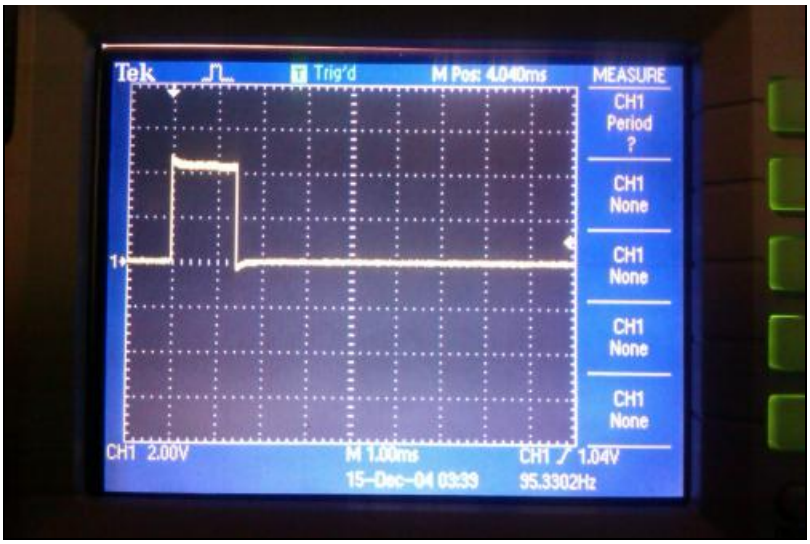
```
            TMR1 = delayValues[4];
            TMR1ON = 1;
            while(!TMR1IF);
            TMR1ON = 0;
            TMR1IF = 0;

           // ------------------------------------------------------------------ //
           // ------------------------- MS 3 to MS 10 -------------------------- //
           // ------------------------------------------------------------------ //

           // Start Timer 8 ms
           TMR1 = 25536;    // 8 ms with prescaler 1:1 and freq 20MHZ
           TMR1ON = 1;
           while(!TMR1IF)
           {
               // Do shit - read sensor or whatever
           }
           TMR1IF = 0;
           TMR1ON = 0;
       }
   }

   void ADC(void)
   {
       GO_DONE = 1;
       while(GO_DONE);
   }
```

Results

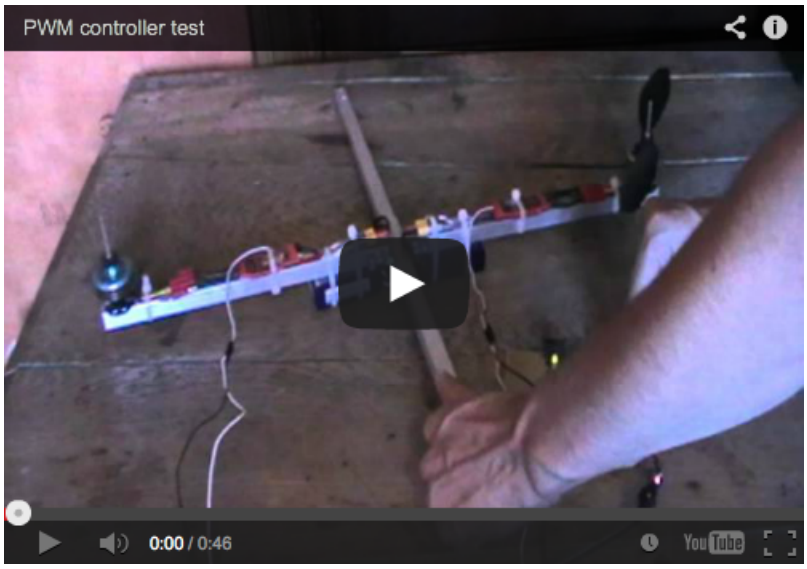
It has to be said that this code is not perfect yet. Because stopping & starting the timers and writing to PORTC all happens outside the timer, a small error is made here. Of course this could be compensated for, but this would make the code more complex, which is not necessairy.

Here are some pictures and a video to show the working circuit.





Here is a video of our PWM-controller driving two motors (I was too lazy to screw on all four motors because I will have to take them off soon anyway ...):



Tags:
Quadrocopter Microcontrollers

« first ‹ previous 1 **2** 3 next › last »