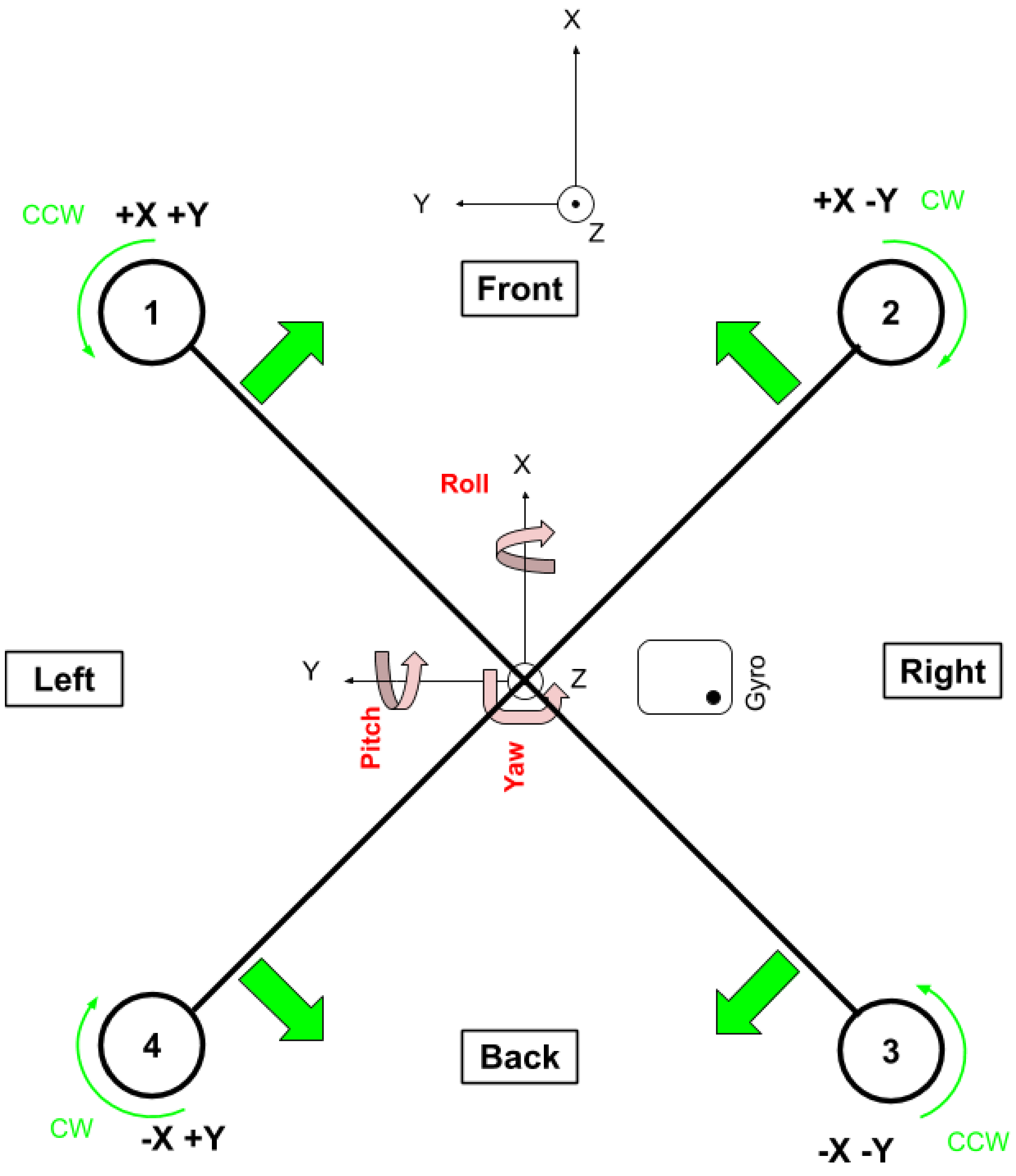
Table of contents

1 Hardware specs

Futaba 7C 2.4GHz Transmitter
R617FS FASST 7-Ch Receiver (FUTL7627)
PPM Frequency : 62.3295 Hz
PPM Period = 16.0437673975 ms
Pulse width from 1000ms to 2000ms

GCC cortex m7 GCC Arm embedded

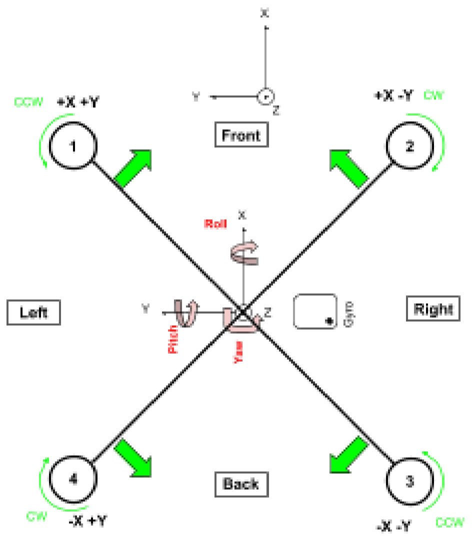
2 Physics:

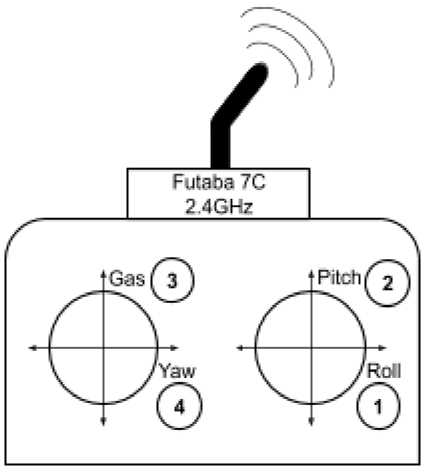

X
Y
Z
CCW
+X +Y
+X -Y
CW
Front
1
2
Roll
X
Left
Y
Z
Gyro
Right
Pitch
Yaw
4
3
Back
CW
-X +Y
-X -Y
CCW

3 Communication

500/3 = 166.6 dps max. rate

Channel 1 = Roll
Channel 2 = Pitch
Channel 3 = Gas
Channel 4 = Yaw





3.1 GAS

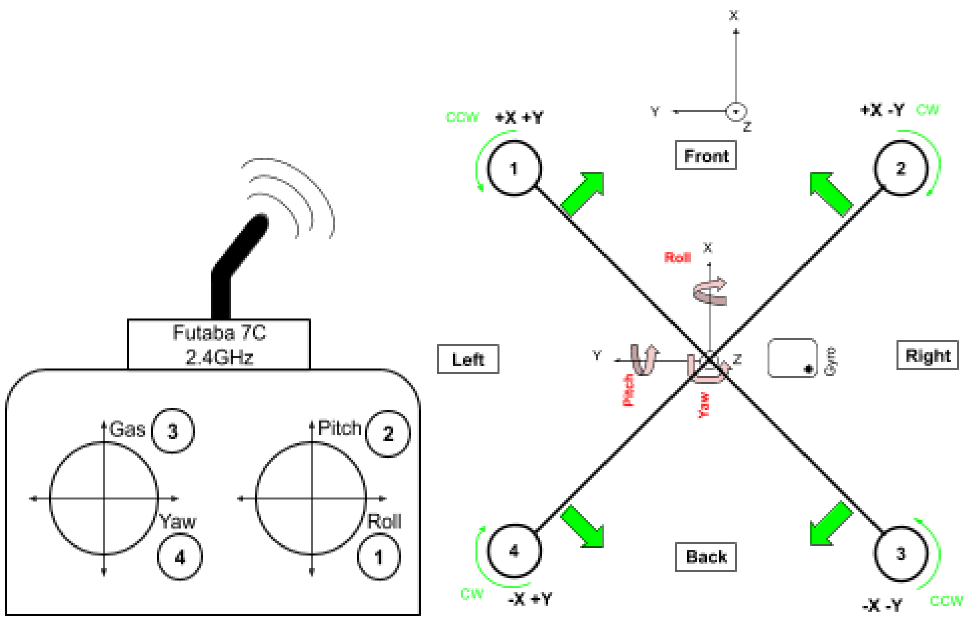
Input (μs)	**Gas ( )**	**Motors**
[1000,1800]	[1000,1800]	1,2,3,4
[1800,2000]	1800	1,2,3,4

3.2 Pitch = channel 2

pitch_input (μs)	**Pitch setpoint (deg/s)**	**Esc contributions**
[1000,1492] = Negative Pitch	(-)[1492 - pitch_input] / 3	(-)+1, (-)+2, (-)-3, (-)-4
[1508,2000] = Positive Pitch	[pitch_input - 1508] / 3	-1,-2,3,4
[1492,1508]	[1500 – 1500] / 3	Final = -1,-2,3,4

Pitch Result

pitch_input (μs)	**Pitch setpoint (deg/s)**	**Pid contributions to each Motor**
[1000,1492] = Negative Pitch	[pitch_input - 1492] / 3	**Final = -1,-2,3,4**
[1508,2000] = Positive Pitch	[pitch_input - 1508] / 3	
[1492,1508]	[1500 – 1500] / 3	

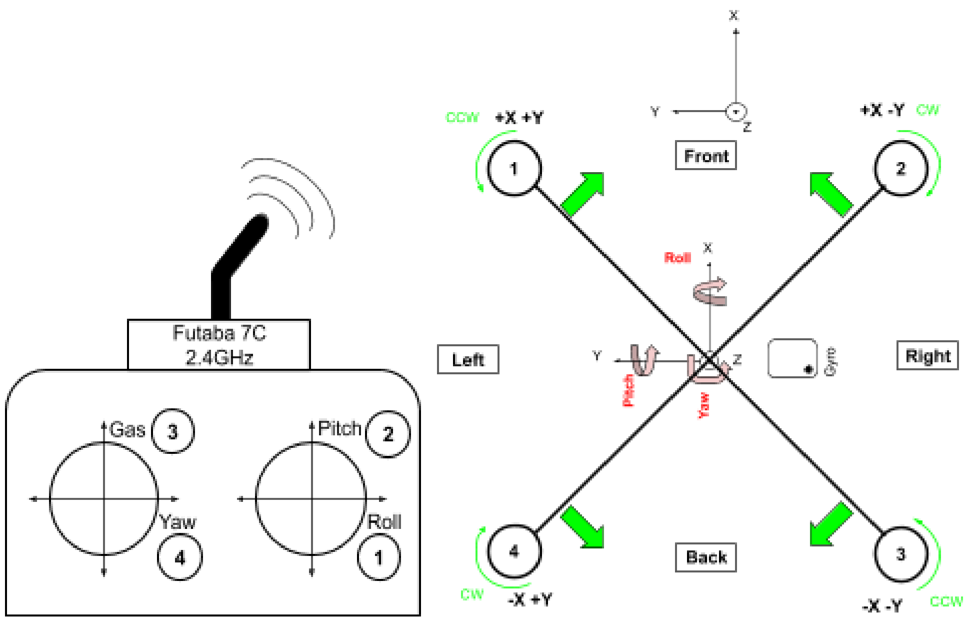


3.3 Roll = channel 1

roll_input (μs)	Roll setpoint (deg/s)	Motors
[1000,1492] = Negative Roll	(-)[1492 - roll_input] / 3	(-)+2, (-)+3, (-)-1, (-)-4
[1508,2000] = Positive Roll	[roll_input - 1508] / 3	+1,+4,-2,-3
[1492,1508]	[1500 – 1500] / 3	Final = +1,+4,-2,-3

Roll Result

roll_input (μs)	Roll setpoint (deg/s)	Pid contributions to each Motor
[1000,1492]	[roll_input -1492] / 3	**Final = +1,+4,-2,-3**
[1508,2000]	[roll_input - 1508] / 3	
[1492,1508]	[1500 – 1500] / 3	



3.4 Yaw = channel 4

yaw_position_setpoint (μs)	**Yaw setpoint (deg/s)**	**Motors**
[1000,1492] = Negative Yaw CCW	(-)[1492 - yaw_input] / 3	(-)+2, (-)+4,-1,-3
[1508,2000] = Positive Yaw CW	[yaw_input - 1508] / 3	+1,+3,-2,-4
[1492,1508]	[1500 - 1500] / 3	Final = +1,+3,-2,-4

Yaw Result

yaw_position_setpoint (μs)	**Yaw setpoint (deg/s)**	**Pid contributions to each Motor**
[1000,1492]	[yaw_input - 1492] / 3	**Final = +1,+3,-2,-4**
[1508,2000]	[yaw_input - 1508] / 3	
[1492,1508]	[1500 - 1500] / 3	

Yaw: Amb nomes motors 2 i 4 actius, gira CCW. Gains = 0;

Why do I need to invert gyro Z output:

Rotating CW motors, makes the Quad turn CCW.
Let's assume I request a: YAW = 2000 , which means Desired Setpoint = ~164
Assuming I just have KP_YAW: desired - measured = 164 -0 = 164
I apply the "extra" 164 to motors +1,+3, which makes me rotate CW (that's what we want if we send full yaw right)
This rotation produces a **negative** measured yaw velocity. Let's say = - 20

In the next loop, the Desired still 164 and the measured -20, so the error has increased. However we are rotating the quad in the CW direction, which is correct. **The solution** is to invert the gyro z axis readout ! :)

This phenomena does not happen for the Roll and Pitch

4 ESC Contributions

esc_1 = gas + pid_output_roll - pid_output_pitch + pid_output_yaw
esc_2 = gas - pid_output_roll - pid_output_pitch - pid_output_yaw
esc_3 = gas - pid_output_roll + pid_output_pitch + pid_output_yaw
esc_4 = gas + pid_output_roll + pid_output_pitch - pid_output_yaw

V2 code:
esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 1 (front-right - CCW)
    esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 2 (rear-right - CW)
    esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 3 (rear-left - CCW)
    esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 4 (front-left - CW)

5 PIDs

A little bit of theory..

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

We can do a stability control loop in position, where the receiver inputs that come from the transmitter sticks feed the desired position setpoint (case 1) or a stability control loop in velocity, which means that the position of the sticks determine the desired angular motion around that axis (case 2).

The term **de(t)/dt** can be obtained using 2 different methods:

Method 1) Subtracting two consecutive e(t) values (1 sensor needed)

Method 2) Feding the current differential of the e(t) measured by a second sensor (2 sensors needed)

5.1 Case 1: Control loop in position

For such implementation we would require inclinometers (first sensor), that would determine the measurement position in a specific axis.

Method 1:
e(t) = Desired Position - Measured Position = Receiver input (degrees) - Inclinometer readout (degrees)
de(t)/dt = {(Desired Position - Measured Position)[i] - (Desired Position - Measured Position)[i-1] }

Method 2:
e(t) = Desired Position - Measured Position = Receiver input (degrees) - Inclinometer readout (degrees)
de(t)/dt = Desired Velocity - Measured Velocity = 0 - gyro readout (degrees/second)
We do not necessarily require the gyro sensor (Method 2), it could also be done with just the inclinometer (Method 1)

That would be a PID control on position, the stick would determine the orientation of the quad and it will hold there.

5.2 Case 2: Control loop in velocity

That's the easiest implementation, since we do not have accelerometers, we will just use the gyroscopes (Method 1 = 1 Sensor needed)

Method 1:
e(t) = Desired velocity - Measured velocity = Receiver input (degrees/second) - gyro readout (degrees/second)
de(t)/dt = e(t)[i] - e(t)[i-1] = {(Desired velocity - Measured velocity) [i] - (Desired velocity - Measured velocity) [i-1] } = [receiver input (i) - gyro input(i)] - [receiver input (i-1) - gyro input (i-1)]

Method 2:
e(t) = Desired Position - Measured Position = Receiver input (degrees) - Inclinometer readout (degrees)

de(t)/dt* = *{Desired velocity - Measured velocity} [i] - {Desired velocity - Measured velocity} [i-1]* =
(Desired velocity[i] - Desired velocity[i-1]) - (Measured velocity[i] - Measured velocity[i-1])
Desired Acceleration - Measured Acceleration = StickChangeRate (arcsec/s^2) - accelerometer Readout (degrees/second) .

This example also proves that method 1 and method 2 are mathematically the same …

That would be a PID control on velocity, the stick would determine the angular speed of the quad, since we do not have an accelerometer nor an inclinometer nor an Aceelerometer, the only possible combination for us to use is **Case 2, Method 1.**

5.3 PID contributions for PHOENIX

The PID controller equation is defined as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

KP · e(t) = **KP ·** (Desired Velocity- Measured Velocity)
KD · de(t)/dt = **KD**· { (Desired Velocity - Measured Velocity) [instant i] - (Desired Velocity - Measured Velocity) [instant i-1] } = **KD**· {(Desired Acceleration - Measured Acceleration) [between instant i and i-1]. = **KD ·** (Desired Velocity - Measured Velocity)

For the KI term we calculate: Instantateous error = (Desired Velocity- Measured Velocity) * dt
KI · accomulated Error = KI · (Instantaneous error + previous errors)
We then add up the instantaneous error to the accomulated so far:
Previous errors += Instantaneous error

5.4 Attitude control code

Case 2 Metod 1 Implementation :

error = reference - reading
- Reference range = [-166.6 , + 166.6] dps
- Reading range = [-500, + 500] dps
- Error Range = [-666.6 , + 666.6] dps

kp_pitch_term : KP * pid_error_temp = KP * (desired_pitch_velocity - measured_pitch_velocity)

ki_pitch_term= KI * pid_error_temp + pid_error_pitch_accomulated
if(ki_pitch_term > pid_max_output){ki_pitch_term = pid_max_pitch }
else if (ki_pitch_term < -pid_max_output){ki_pitch_term = - pid_max_pitch}
kd_pitch_term : pid_error_temp - pid_last_pitch_error_temp;

pid_output_pitch = kp_pitch_term + ki_pitch_term + kd_pitch_term
if(pid_output_pitch >pid_max_pitch){pid_output_pitch = pid_max_pitch}
else if(pid_output_pitch <-pid_max_pitch){pid_output_pitch = -pid_max_pitch}

The same code would apply for roll and yaw...

Loop = 4000 μs

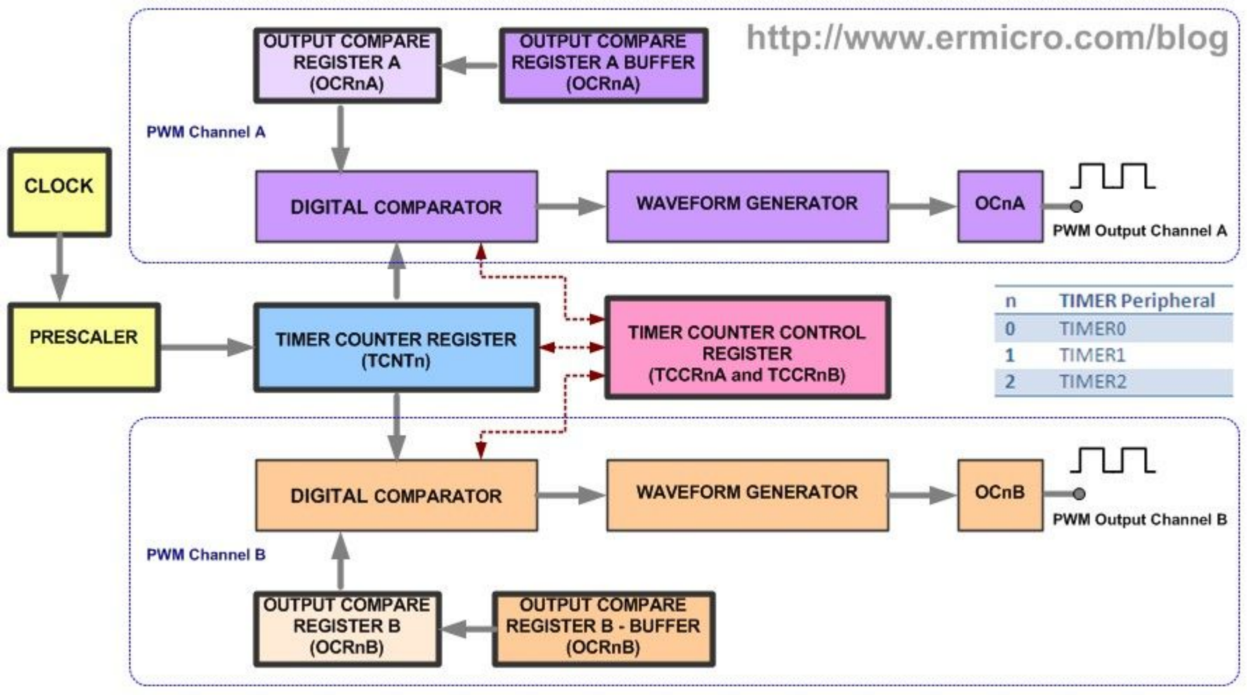
1. Read Gyro angular data
2. Calculations
    a. Read Receiver inputs via Interrupt
    b. Use the previous Receiver inputs
3. Calculate PID outputs for each ESC
    a. Compensate for voltage drop
4. Feed pulse to the ESC → from 1000 to 2000 μs

Additional Functions Required:

- Convert receiver input signals to standardized 1000 - 1500 - 2000 pulses.
    - That requires EEPROM storage of the low center and high values of that channel

PWM

https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM



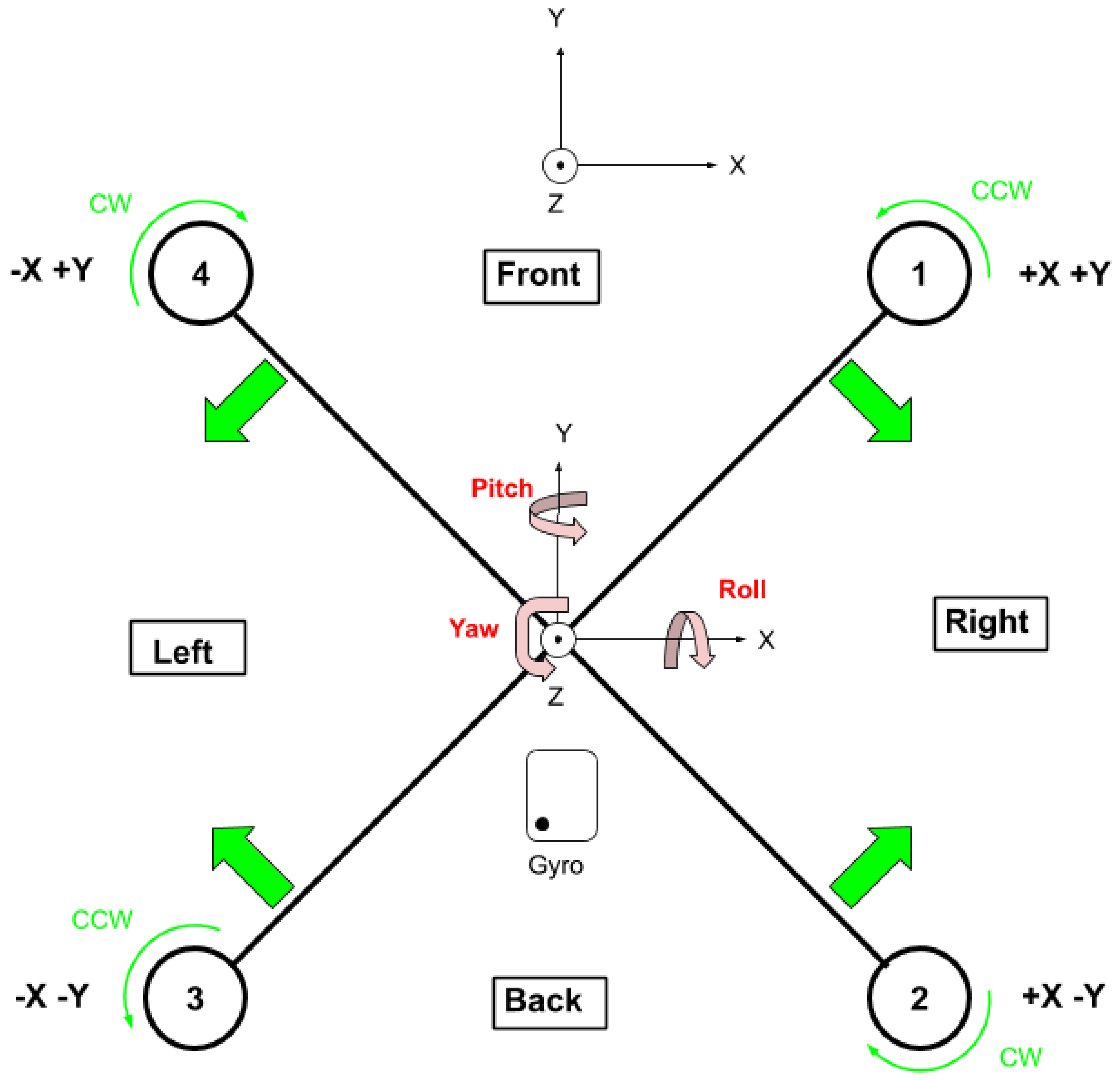
OUTPUT COMPARE REGISTER A (OCRnA)
OUTPUT COMPARE REGISTER A BUFFER (OCRnA)
http://www.ermicro.com/blog
PWM Channel A
CLOCK
DIGITAL COMPARATOR
WAVEFORM GENERATOR
OCnA
PWM Output Channel A
PRESCALER
TIMER COUNTER REGISTER (TCNTn)
TIMER COUNTER CONTROL REGISTER (TCCRnA and TCCRnB)
DIGITAL COMPARATOR
WAVEFORM GENERATOR
OCnB
PWM Output Channel B
PWM Channel B
OUTPUT COMPARE REGISTER B (OCRnB)
OUTPUT COMPARE REGISTER B - BUFFER (OCRnB)

n	TIMER Peripheral
0	TIMER0
1	TIMER1
2	TIMER2

Simplified Atmel AVR ATMega48/88/168/328 Microcontroller PWM Peripheral

Fritzing edit picture

Old Version::


Y
X
Z
CW
-X +Y
4
Front
CCW
1
+X +Y
Y
Pitch
Roll
Yaw
X
Z
Left
Right
Gyro
CCW
-X -Y
3
Back
2
+X -Y
CW