How does a Quadrotor fly? A journey from physics, mathematics, control systems and computer science towards a “Controllable Flying Object”

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
`santoro@dmi.unict.it`

Keynote - L.A.P. 1 Course - Jan 10, 2014

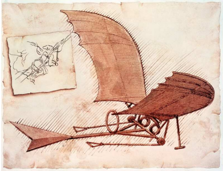
Overview

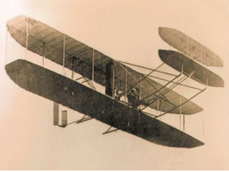
1. Why Multi-rotors?
2. Structure and Physics of a Quadrotor
3. From Analysis to Driving:
   How can I impose a movement to my quadrotor?
4. The ideal world and the real world:
   Why we need Control Systems Theory!
5. Rates and Angles:
   Could I control the *attitude*?
6. What about Altitude or GPS control?

Part I

Why Multi-rotors?

Flying Machines





- **"To fly"** has been one of the dreams of the humans
- But the story tells that building flying machines is not easy!
- A basic and common component: **the wing**
- Two kind of "flying machines" (excluding rockets and balloons):
    1. Fixed wing, i.e. airplanes
    2. Rotating wing, i.e. helicopters

Design and Implementation problems

Airplanes (fixed wing)

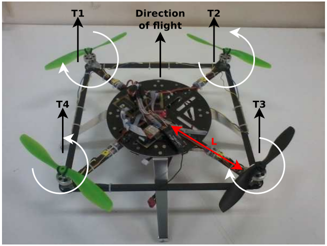
- Wing profile and shape
- Wing and stab size/area
- Wing load
- Position of the COG
- Motion is achieved by driving (mechanically) the mobile surfaces (aleirons, rudder, elevator)

Helicopters (rotating wing, VTOL)

- Size and structure of the rotor
- Mechanical system to control motion inclination
- Yaw balancing system for the rotor at tail
- Position of the COG
- Motion is achieved by (mechanically) changing the inclination of the rotor and the pitch of the rotor wings

Multi-rotors ...

- are mechanically simple: they have *n* motors and *n* propellers
- do not require complex mechanical parts to control the flight
- can fly and move only by changing motor speed
- are controlled only by a electronic-/computer-based system

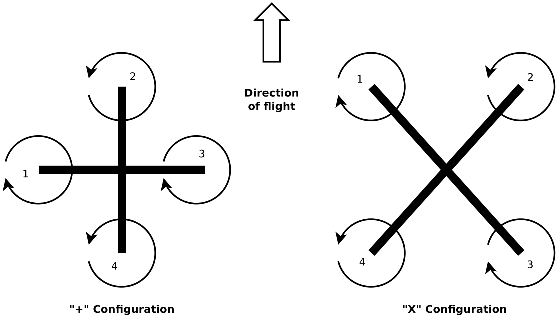


Building them is simple!!

Part II

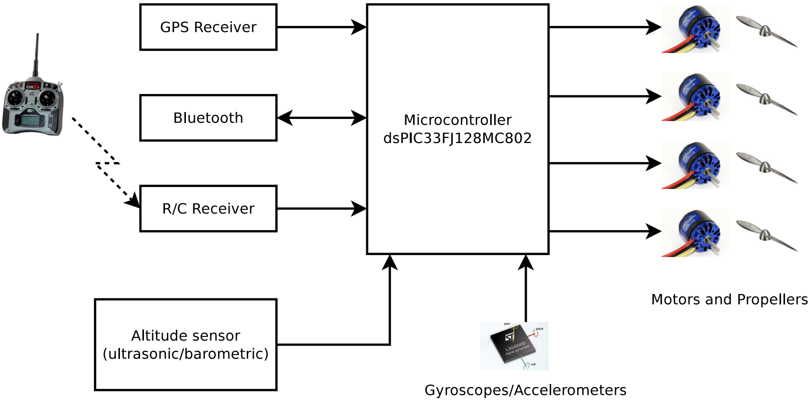
Structure and Physics of a Quadrotor

Structure of a Quadrotor (Mechanics)

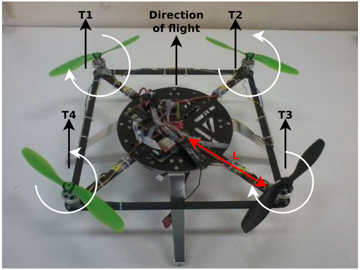


- Four **equal** propellers generating four thrust forces
- Two possible **configurations**: "$+$" and "$\times$"
- Propellers 1 and 3 rotates CW, 2 and 4 rotates CCW
- Required to compensate the *action/reaction effect* (Third Newton's Law)
- Propellers 1 and 3 have opposite **pitch** w.r.t. 2 and 4, so all thrusts have the **same direction**

Structure of a Quadrotor (Electronics)

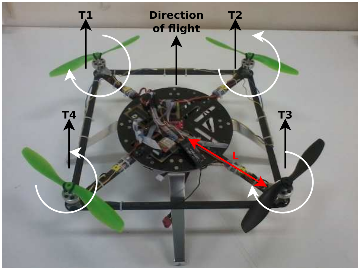


Forces and Rotation speeds



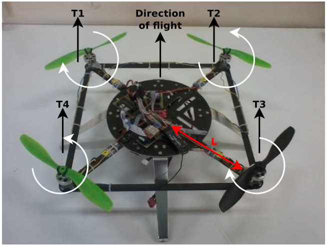
- $\omega_1, \omega_2, \omega_3, \omega_4$: **rotation speeds** of the propellers
- T_1, T_2, T_3, T_4: **forces** generated by the propellers
- $T_i \propto \omega_i^2$: on the basis of propeller shape, air density, etc.
- m: mass of the quadrotor
- mg: weight of the quadrotor

Moments



- M_1, M_2, M_3, M_4: **moments** generated by the forces
- $M_i = L \times T_i$

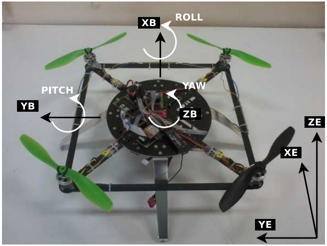
Hovering Condition (Equilibrium)



1. **Equilibrium of forces**: $\sum_{i=1}^{4} T_i = -mg$
2. **Equilibrium of directions**: $T_{1,2,3,4} || g$
3. **Equilibrium of moments**: $\sum_{i=1}^{4} M_i = 0$
4. **Equilibrium of rotation speeds**: $(\omega_1 + \omega_3) - (\omega_2 + \omega_4) = 0$

Violating one (or more) of these conditions implies to impose a certain movement to the quadrotor

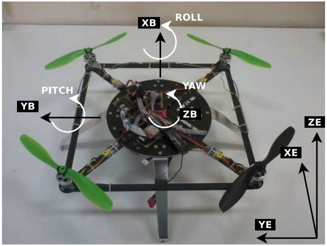
Reference Systems



There are two reference systems:

1. The **inertial** reference systems, i.e. the Earth frame (x_E, y_E, z_E)
2. The **quadrotor** reference system, i.e. the Body frame (x_B, y_B, z_B)

Euler Angles

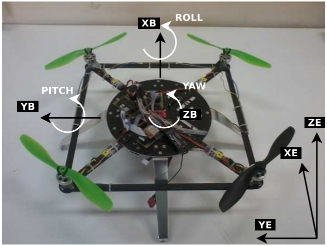


Three angles (ϕ, θ, ψ) define the transformation between the two systems:

- **Roll**, ϕ: angle of rotation along axis $x_B || x_E$
- **Pitch**, θ: angle of rotation along axis $y_B || y_E$
- **Yaw**, ψ: angle of rotation along axis $z_B || z_E$

They are called **Euler Angles**

Angular Speeds

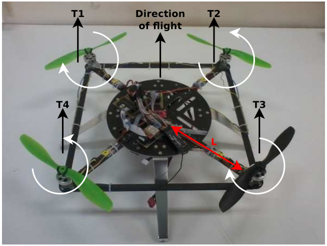


The derivative of (ϕ, θ, ψ) w.r.t. time are the **angular/rotation speeds** $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ of the system:

- $\dot{\phi}$, Roll rate
- $\dot{\theta}$, Pitch rate
- $\dot{\psi}$, Yaw rate

From Analysis to Driving:
How can I impose a movement to my quadrotor?

Hovering Condition (Equilibrium)



1. **Equilibrium of forces**: $\sum_{i=1}^{4} T_i = -mg$
2. **Equilibrium of directions**: $T_{1,2,3,4} || g$
3. **Equilibrium of moments**: $\sum_{i=1}^{4} M_i = 0$
4. **Equilibrium of rotation speeds**: $(\omega_1 + \omega_3) - (\omega_2 + \omega_4) = 0$

As a consequence:

- $\dot{\phi} = 0 \quad \dot{\theta} = 0 \quad \dot{\psi} = 0$
- $\phi = 0 \quad \theta = 0 \quad \psi = 0$

Going Up and Down

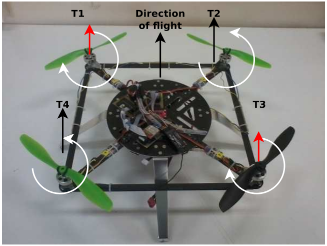
1. **No equilibrium of forces**: $\sum_{i=1}^{4} T_i \neq -mg$
2. **Equilibrium of directions**: $T_{1,2,3,4} || g$
3. **Equilibrium of moments**: $\sum_{i=1}^{4} M_i = 0$
4. **Equilibrium of rotation speeds**: $(\omega_1 + \omega_3) - (\omega_2 + \omega_4) = 0$

By increasing/decreasing the rotation speed of **all** the propellers we can:

- **Go Up**: $\sum_{i=1}^{4} T_i > -mg$
- **Go Down**: $\sum_{i=1}^{4} T_i < -mg$

Euler angles and rates remain 0

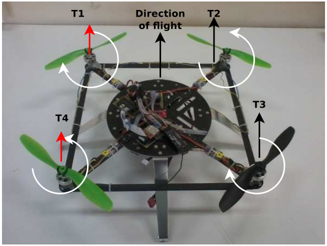
Yaw Rotation



1. **Equilibrium of forces**: $\sum_{i=1}^{4} T_i = -mg$
2. **Equilibrium of directions**: $T_{1,2,3,4} || g$
3. **Equilibrium of moments**: $\sum_{i=1}^{4} M_i = 0$
4. **No equilibrium of prop speeds**: $(\omega_1 + \omega_3) - (\omega_2 + \omega_4) \neq 0$

As a consequence:

$\dot{\psi} = k_Y((\omega_1 + \omega_3) - (\omega_2 + \omega_4)) \qquad \psi = \int \dot{\psi} dt$



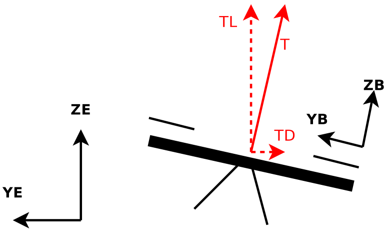
No equilibrium of moments: $\sum_{i=1}^{4} M_i \neq 0$
... by unbalancing propeller speeds as:

$$(\omega_1 + \omega_4) - (\omega_2 + \omega_3) \neq 0$$

As a consequence:

- $\dot{\phi} = k_R((\omega_1 + \omega_4) - (\omega_2 + \omega_3)) \qquad \phi = \int \dot{\phi} dt$
- **No equilibrium of directions**: $T_{1,2,3,4}$ not parallel to g

Roll Rotation and Translated Flight

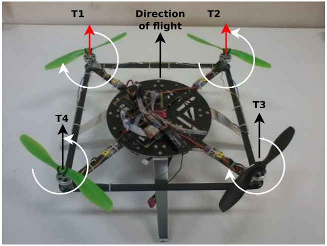


Total thrust $T = \sum_{i=1}^{4} T_i$ is decomposed in:

- **Lift Force**: $T_L = T \cos\phi$
- **Drag Force**: $T_D = T \sin\phi$

Avoiding diving implies $T_L = T \cos\phi = -mg$ thus in **translated flight** we need more power w.r.t. **hovering** or **yawing**.

Pitch Rotation



No equilibrium of moments: $\sum_{i=1}^{4} M_i \neq 0$

... by unbalancing propeller speeds as:

$$(\omega_1 + \omega_2) - (\omega_3 + \omega_4) \neq 0$$

As a consequence:

- $\dot{\theta} = k_P((\omega_1 + \omega_2) - (\omega_3 + \omega_4)) \qquad \theta = \int \dot{\theta} dt$
- Also in this case the total thrust is decomposed thus we need more power w.r.t. **hovering** or **yawing**.

Equations of Movement

We assume a common factor of proportionality k and $F = \sqrt{T}$ (we will see that such an assumption is not a problem!):

$$\begin{array}{rclcl}
\dot{\phi} & = & k((\omega_1 + \omega_4) - (\omega_2 + \omega_3)) & = & k\omega_1 - k\omega_2 - k\omega_3 + k\omega_4 \\
\dot{\theta} & = & k((\omega_1 + \omega_2) - (\omega_3 + \omega_4)) & = & k\omega_1 + k\omega_2 - k\omega_3 - k\omega_4 \\
\dot{\psi} & = & k((\omega_1 + \omega_3) - (\omega_2 + \omega_4)) & = & k\omega_1 - k\omega_2 + k\omega_3 - k\omega_4 \\
F & = & k((\omega_1 + \omega_2 + \omega_3 + \omega_4)) & = & k\omega_1 + k\omega_2 + k\omega_3 + k\omega_4
\end{array}$$

or, using matrices:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ F \end{pmatrix} = \begin{pmatrix} k & -k & -k & k \\ k & k & -k & -k \\ k & -k & k & -k \\ k & k & k & k \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix}$$

Equations of Movement

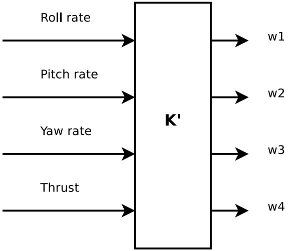
$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ F \end{pmatrix} = \begin{pmatrix} k & -k & -k & k \\ k & k & -k & -k \\ k & -k & k & -k \\ k & k & k & k \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix} = K \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix}$$

This equation gives the **angular velocities** of the quadrotor, given the speed of the **propellers**.

But if we want to **control** the quadrotor we must understand *how to set* ω_i in order to impose a certain rotation rate of axis in the body frame.

Controlling Roll, Pitch and Yaw Rates, and Total Thrust

$$\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix} = K^{-1} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ F \end{pmatrix} = \begin{pmatrix} k & k & k & k \\ -k & k & -k & k \\ -k & -k & k & k \\ k & -k & -k & k \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ F \end{pmatrix}$$

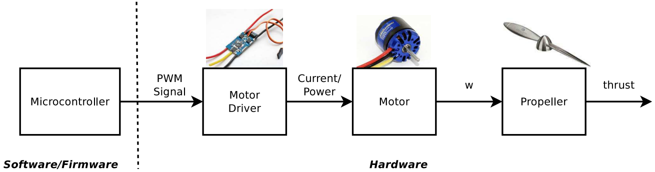


Part IV

The ideal world and the real world:
Why we need Control Systems Theory!

Can we really set the rotation rate of propellers??

Motor/Propeller Driving Schema

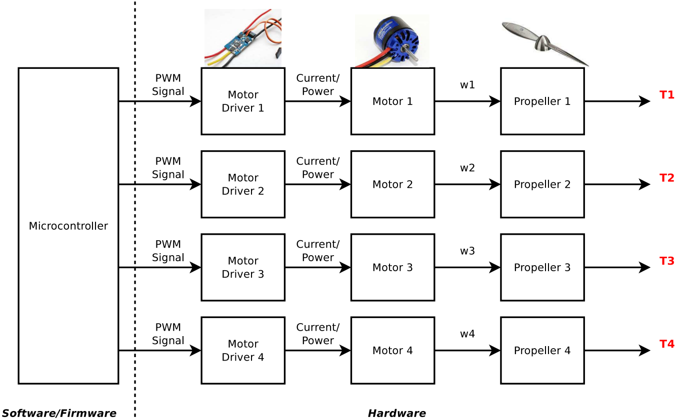


Drivers, motors and propellers are chosen to be of the same type for the four arms.
Software (firmware) controls PWM, but ...

1. Are the drivers really all the same?
2. Are the motors really all the same?
3. Are the propellers really all the same?
4. Is the COG placed at the center of the quadrotor?

The answer is: **In general, No!!**

Motor/Propeller Driving Schema



Same PWM signals applied **different driver/motor/propeller** chains provoke **different thrust forces**, even if the components are of the same type!

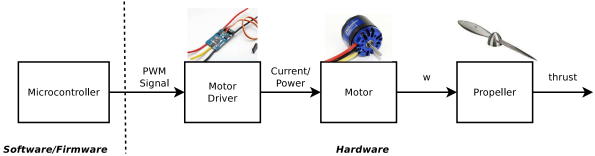
The "Real world" effect

Problem

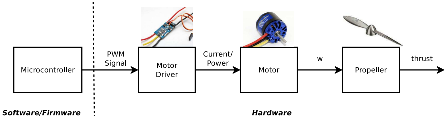
We need to set ω_i by

$$\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix} = K^{-1} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ F \end{pmatrix}$$

but we don't have a direct control on ω_i and propeller thrust



The Mathematician/Physicists Solution



Solution ??

Let's characterize **each driver/motor/propeller chain** and derive the functions:

$$T_i = f_i(PWM_i)$$

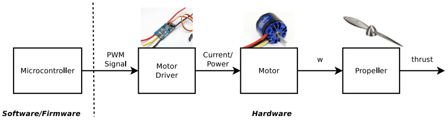
Then, let's invert the functions:

$$PWM_i = f_i^{-1}(T_i)$$

But...

- Characterization is not so easy
- If we change a component, we must repeat the process
- There are unpredictable variables, e.g. air density, wind, etc.

The Computer Scientist/Engineer Solution



Solution ??

Let's sperimentally tune:

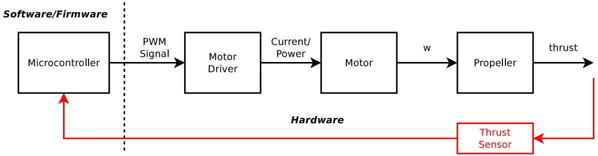
- an offset for each channel
- a gain for each channel

until the system behaves as expected!

But...

- Tuning is not so easy
- If we change a component, we must repeat the process
- There are unpredictable variables, e.g. air density, wind, etc.

The Control System Engineer Solution



Solution!!!! Use feedback!

1. Measure your variable through a **sensor**
2. Compare the measured value with your desired **set point**
3. Apply the correction to the system on the basis of the **error**
4. Go to 1

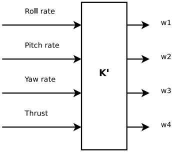
- Tuning is easy and, if the controller is properly designed ...
- it works no matter the components
- it works also in the presence of uncontrollable variables, e.g. air density, wind, etc.

Our measures:

- **Actual** angular velocities on the three axis $(\dot{\phi}_M, \dot{\theta}_M, \dot{\psi}_M)$
- **They are measured through a 3-axis gyroscope!**

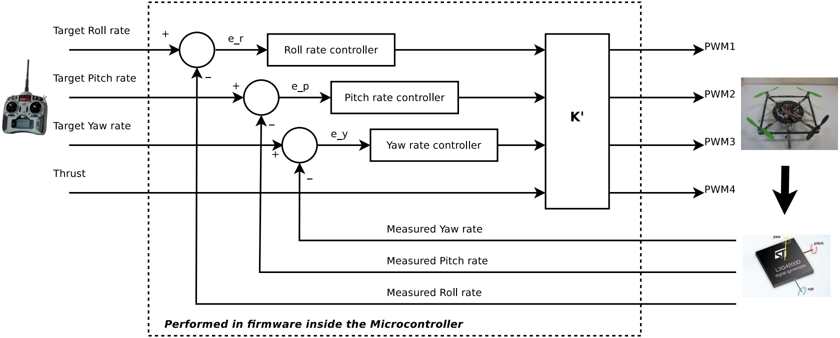
Our set-points:

- **Desired** angular velocities on the three axis $(\dot{\phi}_T, \dot{\theta}_T, \dot{\psi}_T)$
- They are given through the remote control



Using Feedback to Control the Quadrotor

The overall schema of the feedback controller is:



Using Feedback to Control the Quadrotor

Algorithmically

while *True* **do**

On ΔT timer tick ;

$(\dot{\phi}_T, \dot{\theta}_T, \dot{\psi}_T, F)$ = sample_remote_control();

$(\dot{\phi}_M, \dot{\theta}_M, \dot{\psi}_M)$ = sample_gyro();

$e_{\dot{\phi}} := \dot{\phi}_T - \dot{\phi}_M; \quad e_{\dot{\theta}} := \dot{\theta}_T - \dot{\theta}_M; \quad e_{\dot{\psi}} := \dot{\psi}_T - \dot{\psi}_M;$

$C_{\dot{\phi}}$:=roll_rate_controller($e_{\dot{\phi}}$);

$C_{\dot{\theta}}$:=pitch_rate_controller($e_{\dot{\theta}}$);

$C_{\dot{\psi}}$:=yaw_rate_controller($e_{\dot{\psi}}$);

$(pwm_1, pwm_2, pwm_3, pwm_4)^T := K^{-1}(C_{\dot{\phi}_T}, C_{\dot{\theta}_T}, C_{\dot{\psi}_T}, F)^T;$

send_to_motors($pwm_1, pwm_2, pwm_3, pwm_4$);

end

Algorithmically

while *True* **do**

On ΔT timer tick ;

$(\dot\phi_T, \dot\theta_T, \dot\psi_T, F)$ = sample_remote_control();

$(\dot\phi_M, \dot\theta_M, \dot\psi_M)$ = sample_gyro();

$e_{\dot\phi} := \dot\phi_T - \dot\phi_M;\quad e_{\dot\theta} := \dot\theta_T - \dot\theta_M;\quad e_{\dot\psi} := \dot\psi_T - \dot\psi_M;$

$C_{\dot\phi}$:=roll_rate_controller($e_{\dot\phi}$);

$C_{\dot\theta}$:=pitch_rate_controller($e_{\dot\theta}$);

$C_{\dot\psi}$:=yaw_rate_controller($e_{\dot\psi}$);

$(pwm_1, pwm_2, pwm_3, pwm_4)^T := K^{-1}(C_{\dot\phi_T}, C_{\dot\theta_T}, C_{\dot\psi_T}, F)^T;$

send_to_motors($pwm_1, pwm_2, pwm_3, pwm_4$);

end

The key is in the controllers!!

The P.I.D. Controller

The most common used controller type is the **Proportional-Integral-Derivative** controller, represented by the following function:

PID Function

C := xxx_rate_controller(e);

That is:

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

In a discrete world (at k^{th} sampling instant):

$$C(k) := K_p e(k) + K_i \sum_{j=0}^{k} e(j)\, \Delta T + K_d \frac{e(k) - e(k-1)}{\Delta T}$$

The P.I.D. Controller

PID Function

$$C(k) := K_p e(k) + K_i \sum_{j=0}^{k} e(j)\,\Delta T + K_d \frac{e(k) - e(k-1)}{\Delta T}$$

Constants K_p, K_i, K_d regulate the behaviour of the controller:

- K_p drives the short-term action
- K_i drives the long-term action
- K_d drives the action on the basis of the “error trend”

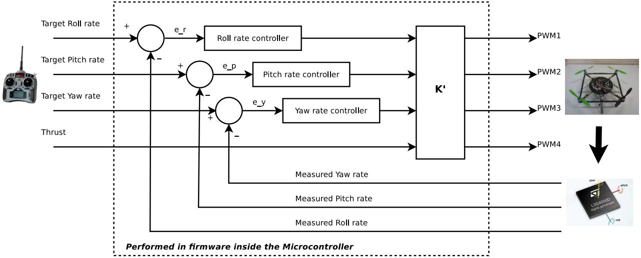
Constants K_p, K_i, K_d are tuned:

- Using a specific tuning method (Ziegler-Nichols)
- Sperimentally by means of “trial-and-error”

Part V

Rates and Angles:
Could I control the *attitude*?

Rates are not Angles



The above schema controls **rates**:

- suppose a roll angle of $\phi = 10^o$
- but no roll rotation (rate), i.e. $\dot{\phi} = 0$
- and no roll rotation command (sticks set to center)
- $\Rightarrow$ the quadrotor is **not** horizontal and performs a **translated flight**

Could we control *angles* instead of *rates*?

Measuring Angles (instead of Rates): Gyros

First we must **measure** euler angles (ϕ, θ, ψ)!
We could do this by using **Gyroscopes**, **Accelerometers**, **Magnetometers**, but...

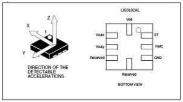
Gyroscopes measure *angular velocities* which can be **integrated** in order to derive the angle $\alpha(t) = \int_0^t \dot{\alpha}(\tau) d\tau$, but:

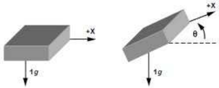
- Numeric integration is affected by approximation errors
- Gyroscopes are affected by an *offset*, i.e. they give non-zero value when the measure should be zero
- Such an *offset* is not constant over time and depends on the temperature

The estimated angle **is not** reliable!

Measuring Angles: Accelerometers

An accelerometer is a sensor measuring the acceleration over the three axis (a_x, a_y, a_z).





- If the sensor is **static** sensed values are the **projections** of g vector in the sensor reference system
- Two functions (using *arctan*) determines **pitch** and **roll**:
  $\phi = \tan^{-1} \frac{-a_y}{-a_z}$
  $\theta = \tan^{-1} \frac{a_x}{\sqrt{a_y^2 + a_z^2}}$
- But if the object is moving (e.g. shaking) other accelerations appear

The computed angles **are not** reliable!

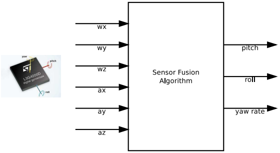
Measuring Angles: Two sensors, No reliability!

- **Gyros**
  - Drift
  - Approximate discrete integration
- **Accelerometers**
  - Precise only if sensor is not “shaking”

We have **two different source** of the **same** information which are affected by **two different error** types.

We can use **both** measures by *fusing* them in order to adjust the error and obtain a reliable information.

Sensor Fusion



Basic Algorithm

while *True* **do**

  On ΔT timer tick ;

  $(\dot\phi, \dot\theta, \dot\psi)$ = sample_gyro();

  (a_x, a_y, a_z) = sample_accel();

  $(\phi, \theta, \psi) = (\phi, \theta, \psi) + \Delta T(\dot\phi, \dot\theta, \dot\psi)$;

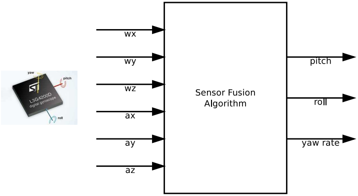
  $\hat\phi = \tan^{-1}(-a_y / - a_z)$;

  $\hat\theta = \tan^{-1}(a_x / \sqrt{a_y^2 + a_z^2})$;

  $(\phi, \theta, \psi) = \mathit{fusion\_filter}(\phi, \theta, \psi, \hat\phi, \hat\theta)$;

end

Sensor Fusion: Algorithms



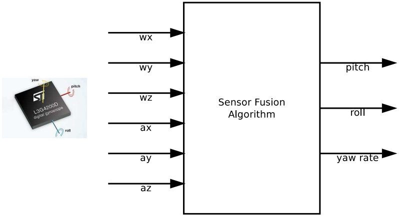
The key is the **filter function**!

- DCM (Direction Cosine Matrix)
- Complementary filters
- **Kalman filters**

Basic idea:

- Derive an **error function** $e(t) = real(t) - estimated(t)$
- Design a **controller** able to guarantee $\lim_{t \to \infty} e(t) = 0$

Sensor Fusion: Algorithms



High computational load due to:

- Rotations in the 3D space
- Matrix calculations

May we reduce the load?

Representing Rotations in 3D

Direction Cosine Matrix

$$DCM = \begin{pmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{pmatrix}$$

$$s = \sin, \quad c = \cos$$

This matrix is re-computed at each iteration!!

Rotating a vector $v = (x, y, z)$ implies the product $DCM \cdot v$.

Representing Rotations in 3D

Quaternions

A **quaternion** is a complex number with one real part and three imaginary parts:

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

$$\mathbf{i}, \mathbf{j}, \mathbf{k} = imaginary \;\; units$$

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

While **Complex numbers** can be used to represent **rotations in 2D**, **Quaternions** can be used to represent **rotations in 3D**.

Rotations in 3D and Quaternions

- Transformations from Euler angles to quaternion exist:

$$q \rightarrow (\phi, \theta, \psi)$$

$$(\phi, \theta, \psi) \rightarrow q$$

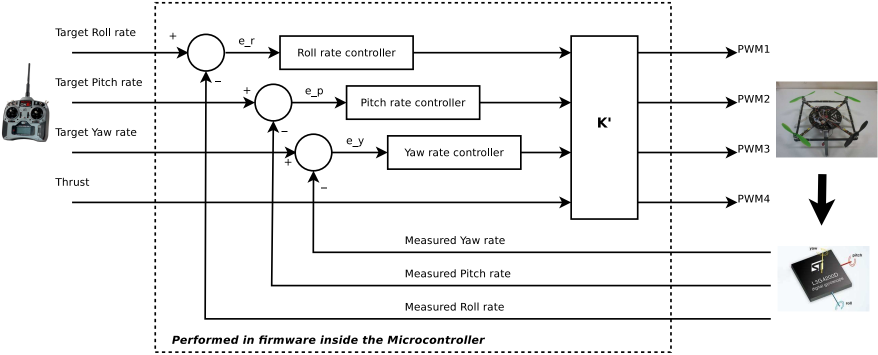
- Rotating a vector v using a quaternion implies the product $q\overline{v}q^*$ where q^* is the conjugate of q and $\overline{v} = \{0, v_x, v_y, v_z\}$.
- The overall fusion algorithm can be written using quaternion algebra, thus avoiding continuous *sin*, *cos* calculation.
- Quaternions avoid **gimbal lock!**
- The attitude can be easily obtained by using:

$$q \rightarrow (\phi, \theta, \psi)$$

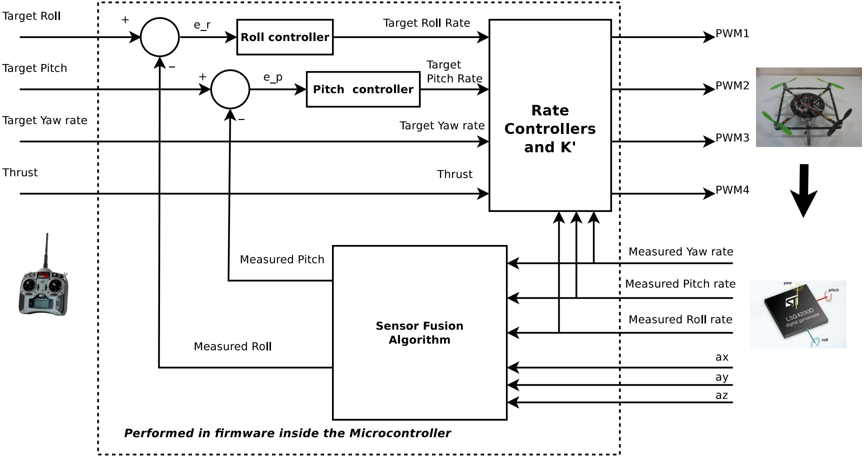
So far so good: Controlling attitude

- Attitude control is achieved using (once again) *feedback controllers*.
- We set the **Target (desired) Attitude** $(\phi_T, \theta_T, \dot{\psi}_T)$ from remote controller.
- **Current quad attitude** $(\phi_M, \theta_M, \dot{\psi}_M)$ is computed using sensor fusion.
- The error signals (differences) are sent to PID controllers whose output are the **target rates** for rate controllers.
- The basic model is "*cascading controllers*": **attitude controllers** which drives **rate controllers**.

Let’s remind the schema of Rate Controllers



Complete Attitude Controller



Control "loops": Requirements

- Two control loops in the schema
  - rate control (inner);
  - attitude control (outer);
- Attitude control "drives" rate control, thus rate control must have *"enough time"* to reach the desired target.
- Loops must have different **dynamics**, i.e. **sampling time**
- T_r = rate control sampling time
- T_a = attitude control sampling time
- $T_a >> T_r$, $T_a = nT_r$, $n \in \mathcal{N}, n > 1$
- In our quad: $T_r = 5ms$, $T_a = 50ms$

Finally, the overall algorithm

while *True* **do**

- On T_r timer tick ;
- $(\dot{\phi}_M, \dot{\theta}_M, \dot{\psi}_M)$ = sample_gyro();
- (a_x, a_y, a_z) = sample_accel();
- $(\phi_M, \theta_M) = fusion\_filter(\dot{\phi}_M, \dot{\theta}_M, \dot{\psi}_M, a_x, a_y, a_z)$;
- **if** *after N loops* **then**
  - $(\phi_T, \theta_T, \dot{\psi}_T, F)$ = sample_remote_control();
  - $\dot{\phi}_T :=$roll_controller(ϕ_M, ϕ_T);
  - $\dot{\theta}_T :=$pitch_controller(θ_M, θ_T);
- **end**
- $C_{\dot{\phi}} :=$roll_rate_controller$(\dot{\phi}_M, \dot{\phi}_T)$;
- $C_{\dot{\theta}} :=$pitch_rate_controller$(\dot{\theta}_M, \dot{\theta}_T)$;
- $C_{\dot{\psi}} :=$yaw_rate_controller$(\dot{\psi}_M, \dot{\psi}_T)$;
- $(pwm_1, pwm_2, pwm_3, pwm_4)^T := K^{-1}(C_{\dot{\phi}_T}, C_{\dot{\theta}_T}, C_{\dot{\psi}_T}, F)^T$;
- send_to_motors$(pwm_1, pwm_2, pwm_3, pwm_4)$;

Part VI

What about Altitude or GPS control?

Let’s repeat the schema!

Do you need another kind of control? Repeat the schema!

- Identify your source of measure m
- Identify your target t
- Identify the variables to drive v
- Identify the sampling time
- Use a (PID) controller $v = pid(t, m)$

Altitude Control

- H_T = our target height
- H_M = measured height (from a sensor)
- F = output variable to control (desired thrust)
- MT_r = altitude control sampling time, $M > N$

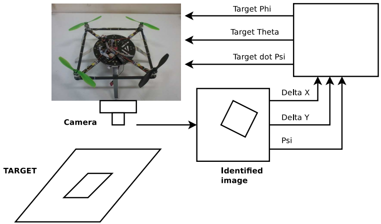
```
while True do
    On T_r timer tick ;
    ...;
    if after M loops then
        H_M = sample_altitude_sensor();
        F :=altitude_controller(H_M, H_T);
    end
    ...
end
```

GPS Control

- Lat_T, Lon_T = our target position
- Lat_M, Lon_T = measured position (from a GPS sensor)
- ϕ_T, θ_T = target variables to control (desired pitch and roll)
- GT_r = GPS control sampling time, $G > N$

while *True* **do**
  On T_r timer tick ;
  ...;
  if *after G loops* **then**
    (Lat_M, Lon_M) = sample_gps();
    ϕ_T :=gps_lon_controller(Lon_M, Lon_T);
    θ_T :=gps_lat_controller(Lat_M, Lat_T);
  end
  ...
end

Note: for a proper GPS navigation, a compass (with related yaw control) is mandatory



while *True* **do**

  On T_r timer tick ;

  ...;

  if *after H loops* **then**

    $(\Delta X, \Delta Y, \Delta\psi)$ = identify_target_with_camera();

    ϕ_T :=x_controller(ΔX);

    θ_T :=y_controller(ΔY);

    $\dot{\psi}_T$:=heading_controller($\Delta\psi$);

  end

Conclusions

It seems easy ....

... but, where is the trick?

- **Are sensors reliable?**
  - Sometimes, NO!
  - Noise due to mechanical vibrations (MEMS-IMU to be filtered by applying **Fourier analysis**)
  - False positives due to wiring problems (Magnetometers, ADC, etc.)
- **Are execution platforms reliable?**
  - Check it!
  - Controllers need **precise** (**real-time**) timing
  - DO NOT Windows to stabilize your quad!!!
  - You can try with RT-Linux
- **Is PID Tuning really easy?**
  - NO! You must learn it!
  - ... and be sure to have a large set of propellers!!
- **Are all those things fun?**
  - OF COURSE!!!! ⌣̈

Will Multi-rotors be the future of personal transportation systems?

Where do I park my multi-rotor??

Demonstration Flight

First prototype: PROBLEMS!!!

- **DIY is fun but ...**
  - The frame is not well balanced... but the control will do the job
  - Too many vibrations (many of them suppressed using Chebyshev filters)
  - Wrong choice of motors (specs report a thurst of 400gr each, but ...)
- **Wiring/Electronics problems**
  - Current spikes reset the ultrasonic sensor
  - I2C sometimes locks (a watchdog intervenes and turn-off motors)
- **Firmware problems**
  - Still working on the sensor fusion algorithm, since it is not satisfactory (we want more stability...)

How does a Quadrotor fly?
A journey from physics, mathematics, control systems and computer science towards a “Controllable Flying Object”

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Keynote - L.A.P. 1 Course - Jan 10, 2014