AREcopter - Arduino Flight Controller

Federico Giugni

31 dicembre 2024

Indice

1 Introduzione

Un quadricottero è un tipo di velivolo a quattro rotori, comunemente usato per applicazioni civili, industriali e di ricerca grazie alla sua semplicità costruttiva e versatilità. La particolarità di un quadricottero risiede nel fatto che, a differenza di un aereo tradizionale, non possiede superfici alari che garantiscano portanza e stabilità aerodinamica. Tutto il controllo del volo è demandato a un sistema di controllo numerico che regola costantemente la velocità dei quattro rotori per mantenere l'equilibrio e rispondere ai comandi.

Il cuore di un quadricottero è rappresentato dal *Flight Controller*, che monitora i dati dai sensori, interpreta i comandi del pilota e controlla i motori per stabilizzare il drone e consentirne il movimento. Esso utilizza una combinazione di sensori come IMU (Inertial Measurement Unit) per raccogliere dati di orientamento e velocità angolare. Le principali funzionalità di un Flight Controller includono la lettura dei comandi dal ricevitore, la lettura dei dati di volo dai sensori, l'elaborazione dei dati per il calcolo della potenza da applicare ai motori e il controllo dei segnali inviati agli ESC per regolare la velocità dei motori.

Un sistema di controllo PID (Proporzionale, Integrale, Derivativo) è una tecnica ampiamente utilizzata nell'ambito dell'ingegneria per controllare sistemi dinamici. Il controllo PID calcola una risposta basata sulla somma di tre componenti:

- **Componente Proporzionale** ($K_p \cdot e(t)$): Proporzionale all'errore attuale, consente di correggere rapidamente la deviazione.
- **Componente Integrale** ($K_i \cdot \int_0^t e(\tau)\, d\tau$): Considera l'errore accumulato nel tempo per eliminare deviazioni persistenti.
- **Componente Derivativa** ($K_d \cdot \frac{de(t)}{dt}$): Reagisce ai cambiamenti improvvisi dell'errore per migliorare la stabilità.

Le equazioni principali del PID sono le seguenti:

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

dove $e(t)$ rappresenta l'errore.

Nell'ambito aeronautico, viene spesso adottato un approccio avanzato basato sul PID a due livelli. In questo schema, il primo livello gestisce la stabilizzazione angolare (pitch, roll, yaw), calcolando l'errore angolare tra l'assetto attuale e quello desiderato. Il secondo livello utilizza l'output del primo livello per regolare la velocità angolare e stabilizzare ulteriormente il volo. Questo sistema è particolarmente efficace per ottenere un controllo preciso e reattivo in applicazioni come il volo dei quadricotteri. Un diagramma esplicativo è illustrato in Figura 1.

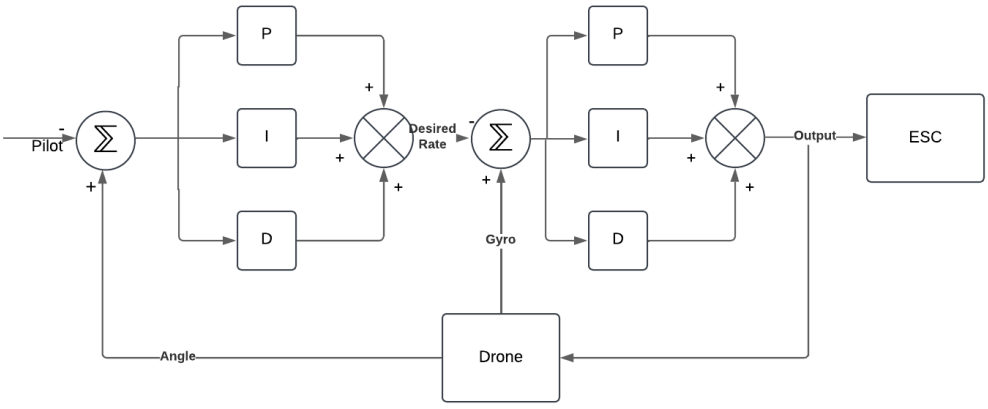

Figura 1: Diagramma del Controllo PID a Due Livelli

2 Descrizione del Sistema

2.1 Specifiche Hardware

Il sistema utilizza i seguenti componenti hardware:

- **Microcontrollore**: Arduino Nano con AMEGA328P.
- **Motori e Controllori**: Quattro motori brushless da 1000KV controllati tramite ESC da 30A.
- **IMU**: Adafruit BNO055 per l'orientamento e la stabilizzazione.
- **Batteria**: Batteria LiPo 3S.
- **Frame**: Telaio F450.
- **PBD**: Power Distribution Board per alimentare ESC, Arduino e ricevitore.

2.2 Collegamenti Hardware

I pin utilizzati sono descritti nella Tabella 1:

Componente	**Pin Arduino**	**Motivazione**
Ricevitore IA6B	3, 4, 2, 8	Input PWM (interrupt supportati)
ESC	5, 6, 9, 10	Output PWM per controllo motori
IMU BNO055	SDA, SCL	Comunicazione I2C
LED	11, 12	Indicazioni visive di stato

Tabella 1: Collegamenti hardware principali

Il sistema è alimentato dalla batteria LiPo 3S tramite la PBD, che distribuisce l'energia agli ESC e all'Arduino. Quest'ultimo alimenta il ricevitore e altri componenti a basso consumo.

3 Software

3.1 Architettura del Software

L'architettura software della piattaforma è basata sui principi della programmazione orientata agli oggetti. Ogni componente principale è rappresentato da una classe che incapsula sia i dati sia le operazioni ad esso correlate. Questo approccio consente di astrarre le complessità hardware, mantenendo il codice modulare, scalabile e di facile manutenzione.

La struttura principale è organizzata attorno alla classe `Drone`, che agisce come punto centrale di coordinamento. Le funzionalità principali della piattaforma sono distribuite tra le seguenti classi:

- `IA6B`: Per la gestione del ricevitore e la decodifica dei segnali PWM provenienti dal pilota.

- `BNO055`: Per la gestione dell'unità IMU, che fornisce dati di orientamento (pitch, roll, yaw) e velocità angolare.
- `ESC`: Per il controllo degli Electronic Speed Controllers, responsabili della velocità dei motori.
- `FlightController`: Per il calcolo dei segnali PID, che garantiscono la stabilizzazione e il controllo preciso del volo.
- `LED`: Per la gestione dello stato visivo del sistema, come segnalazioni operative e di errore.

Ogni classe è progettata per interfacciarsi direttamente con il componente hardware associato, gestendo autonomamente la comunicazione hardware/software. Questo design facilita l'estensibilità, permettendo l'aggiunta di nuove funzionalità o componenti con un impatto minimo sulle altre parti del sistema. Inoltre, la separazione delle responsabilità rende il codice più leggibile e testabile.

3.2 Implementazione

3.2.1 La Classe Drone

La classe `Drone` funge da centro di controllo per la gestione del drone. Di seguito i dettagli degli attributi più rilevanti:

- **Oggetti di Gestione Hardware**:
  - `ESC motors[NUM_ESCs]`: Array di oggetti `ESC` che rappresentano i motori del drone.
  - `IA6B receiver`: Istanza per la gestione del ricevitore IA6B.
  - `BNO055 imu`: Gestisce la IMU per i dati di orientamento e velocità angolare.
  - `LED ledRed` e `LED ledGreen`: Indicatori visivi dello stato del sistema.
- **Dati Operativi**:
  - `SystemState state`: Stato attuale del drone (`DISARMED`, `ARMED`, ecc.).
  - `FlightData angleData`, `FlightData gyroData`: Dati di orientamento e velocità angolare.
  - `PilotData pilotData`: Input di controllo ricevuti dal pilota.
  - `ControlDataType pilotInput`: Tipo di comando ricevuto dal pilota (`START_INPUT`, `STOP_INPUT`, ecc.).
- **Controllo del Volo**:
  - `FlightController flightController`: Istanza del controllore di volo che implementa logiche PID a due livelli.
  - `ESCData escData`: Valori dei segnali PWM per i motori.

3.2.2 La Classe FlightController

La classe `FlightController` implementa i controlli PID per garantire la stabilità del drone:

- **PID di Stabilizzazione Angolare**:
  - `PIDControl pidPitchAngle`, `PIDControl pidRollAngle`, `PIDControl pidYawAngle`: Stabilizzano rispettivamente gli assi pitch, roll e yaw.
- **PID di Stabilizzazione del Giroscopio**:
  - `PIDControl pidPitchGyro`, `PIDControl pidRollGyro`, `PIDControl pidYawGyro`: Stabilizzano la velocità angolare.

3.2.3 Metodi di Lettura tramite Interrupt

I metodi di lettura tramite interrupt sono stati implementati per superare le limitazioni della classica lettura PWM basata su `pulseIn`. Inizialmente, il metodo `pulseIn` veniva utilizzato per leggere i segnali di controllo provenienti dal ricevitore IA6B. Tuttavia, chiamando `pulseIn` quattro volte per lettura, si osservavano ritardi significativi, fino a 80 ms, per ogni ciclo del loop principale.

Per risolvere questo problema, si è sfruttata la capacità dei pin 2 e 3 di supportare interrupt hardware (channel interrupt), delegando a essi la gestione dei segnali PWM per due canali principali. Per gli altri due canali, `pitch` e `yaw`, si è scelto di intervallare ciclicamente le letture utilizzando `pulseIn`. Questa combinazione ha permesso di ridurre significativamente i ritardi causati da `pulseIn`, rendendoli trascurabili, pur mantenendo un segnale stabile e affidabile.

Nel modulo `PWM.cpp`, la funzione `ISR_generic` gestisce i segnali PWM tramite interrupt, calcolando la durata del segnale alto o basso e aggiornando i valori in tempo reale:

```
void ISR_generic(byte isr)
{
    unsigned long now = micros();
    bool state_now = digitalRead(isr_pin[isr]);
    if (state_now != isr_last_state[isr])
    {
        if (state_now == isr_trigger_state[isr])
        {
            isr_timer[isr] = now;
        }
        else
        {
            isr_value[isr] = (unsigned int)(now - isr_timer[isr]);
            isr_age[isr] = now;
        }
        isr_last_state[isr] = state_now;
    }
}
```

Nel modulo `IA6B.cpp`, il metodo `read()` utilizza sia i valori acquisiti tramite interrupt per `throttle` e `roll`, sia quelli acquisiti ciclicamente tramite `pulseIn` per `pitch` e `yaw`. Questo approccio ibrido combina l'efficienza degli interrupt con la semplicità della lettura PWM tradizionale:

```
bool IA6B::read()
{
    if ((raw_data.throttle = pwm_throttle.getValue()) == 0)
        isValid[THROTTLE] = false;
    if ((raw_data.roll = pwm_roll.getValue()) == 0)
        isValid[ROLL] = false;
    if (cycleCounter % PULSEIN_READ_CYCLE == 0)
    {
        if (readPitchNext)
        {
            if ((raw_data.pitch = pulseIn(pin.pitch, HIGH, 30000)) == 0)
                isValid[PITCH] = false;
            readPitchNext = false;
        }
        else
        {
            if ((raw_data.yaw = pulseIn(pin.yaw, HIGH, 30000)) == 0)
                isValid[YAW] = false;
            readPitchNext = true;
        }
    }
}
```

3.2.4 Conversione dei Dati

Nel modulo `Utils.cpp`, sono presenti due funzioni chiave per la conversione dei dati:

- `digital_to_pwm`: Converte un valore digitale in un segnale PWM tramite una mappatura lineare. Garantisce che il risultato sia compreso nell'intervallo valido.
- `pwm_to_digital`: Esegue la conversione inversa, mappando un valore PWM in un intervallo digitale definito.

3.2.5 Diagramma del Ciclo Principale del Drone

Il ciclo principale del drone segue la sequenza:

1. Lettura dei dati del ricevitore `IA6B`.
2. Lettura dei dati della IMU `BNO055`.
3. Interpretazione e Validatura dei dati.
4. Aggiornamento dello stato dei LED.
5. Calcolo dei segnali PID tramite il `FlightController`.

6. Aggiornamento dei segnali PWM agli ESC.

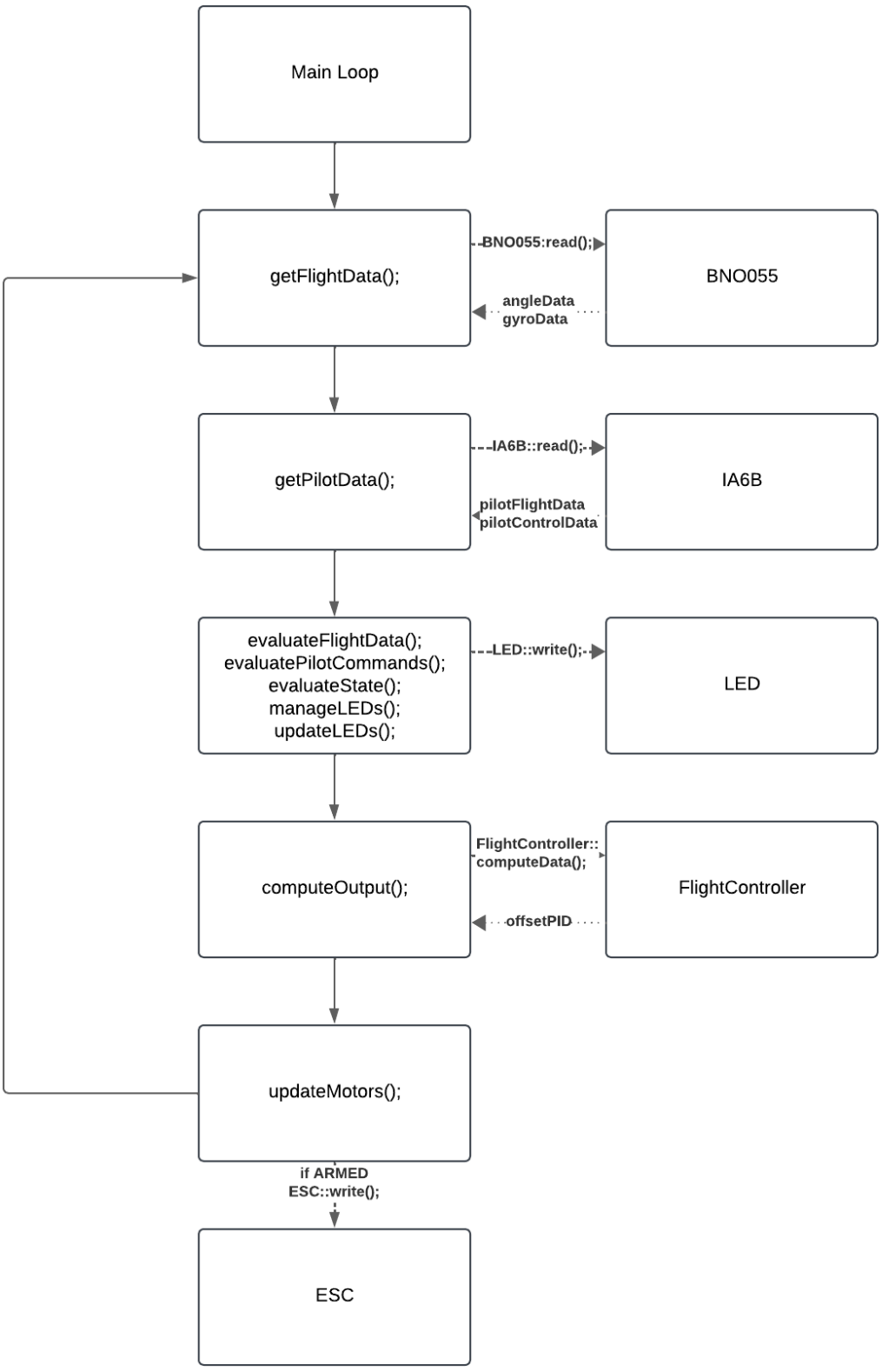

Figura 2: Diagramma del Ciclo Principale del Drone

3.3 Validazione del Software

La validazione ha avuto inizio testando le singole classi in un ambiente controllato. Ciascun componente è stato istanziato nel ciclo principale di uno script Arduino separato e testato individualmente per garantire la corretta comunicazione tra l'Arduino e i dispositivi hardware associati. Ad esempio:

- Il modulo `IA6B` è stato verificato leggendo i segnali PWM del ricevitore e analizzando i valori trasmessi in seriale.
- Il modulo `BNO055` è stato testato per garantire che i dati di orientamento e velocità angolare fossero accurati e coerenti.
- Gli ESC sono stati calibrati e controllati per verificare che i segnali PWM inviati dal microcontrollore corrispondessero alle specifiche impostate.

Successivamente, sono stati validati i sistemi di conversione dei dati e i risultati delle formule implementate, come quelle relative ai PID e alle mappature PWM. Utilizzando gli strumenti di debugging, sono stati analizzati gli output di queste funzioni in condizioni controllate.

Infine, per verificare il comportamento del sistema integrato, il drone è stato mosso manualmente, e i dati di volo trasmessi via seriale al PC sono stati analizzati. Questa fase ha confermato il corretto funzionamento degli algoritmi di controllo del volo, mostrando la coerenza tra i comandi del pilota, i dati dei sensori e le azioni correttive generate dai PID.

Questo approccio sistematico ha permesso di identificare e correggere i difetti in modo iterativo, garantendo la stabilità e l'affidabilità del software prima dei test di volo reale.

4 Calibrazione e Testing

La calibrazione del sistema di controllo è stata effettuata tramite un approccio sperimentale, utilizzando una piattaforma di test controllato. Questa piattaforma, progettata appositamente, consiste nel legare il quadricottero per le due estremità anteriori e posteriori con cavi tesi, consentendone la rotazione su un singolo asse (rollio). Poiché il drone è simmetrico sui quattro quadranti, i dati raccolti sono stati considerati validi anche per l'asse del beccheggio.

4.1 Calibrazione del Controllo PID

La calibrazione è stata suddivisa in due fasi principali, corrispondenti ai due livelli di controllo PID: velocità angolare e angolo.

4.1.1 Calibrazione del PID sulla Velocità Angolare

Il primo passo è stato calibrare il controllo PID sulla velocità angolare, per insegnare al drone "come ruotare". Il procedimento è stato il seguente:

- Si è utilizzato come velocità angolare desiderata l'input fornito dal tester.
- I termini I (integrale) e D (derivativo) sono stati inizialmente azzerati, concentrandosi solo sul termine P (proporzionale).

- Incrementando gradualmente il valore di P, si è raggiunto un equilibrio in cui il drone rispondeva velocemente agli input, ma senza reazioni eccessivamente reattive.
- Successivamente, il termine D è stato introdotto gradualmente per smorzare eventuali oscillazioni o risposte eccessive ai cambiamenti di input.
- Infine, il termine I è stato aumentato fino a compensare gli errori persistenti nel tempo (drift), garantendo una velocità angolare nulla in assenza di input.

4.1.2 Calibrazione del PID sull'Angolo

Una volta calibrato il controllo della velocità angolare, si è passati al controllo sull'angolo, necessario per insegnare al drone "dove ruotare". Questo livello di controllo ha seguito lo stesso approccio metodico:

- Gli input desiderati sono stati forniti tramite il sistema standard (comandi del pilota e dati IMU).
- Si è proceduto con l'incremento graduale del termine P, seguito dall'introduzione controllata di D e, infine, dall'aggiunta di I per compensare errori persistenti.

Questa fase di calibrazione si è conclusa quando il drone era in grado di:

- Rispondere correttamente agli input del pilota.
- Mantenere un'angolazione di 0° (orizzontale) in assenza di input.
- Ristabilire la stabilità anche se disturbato da fattori esterni.

4.2 Applicazione dei Parametri Calibrati e Test di Volo

I parametri PID calibrati per l'asse del rollio sono stati applicati anche all'asse del beccheggio, sfruttando la simmetria del drone. Prima di procedere ai test di volo, sono stati condotti ulteriori controlli per verificare la coerenza del comportamento del sistema.

Durante i primi test di volo, il drone ha mostrato un'eccellente stabilità, rispondendo prontamente ai comandi del pilota. In questa fase, i valori PID per l'asse dell'imbardata sono stati calibrati seguendo lo stesso approccio descritto in precedenza.

La calibrazione complessiva ha garantito che il drone fosse in grado di mantenere stabilità e precisione durante il volo, anche in condizioni variabili o in presenza di perturbazioni esterne.

5 Codice Sorgente

Il codice completo del progetto è disponibile nella repository GitHub al seguente indirizzo:

`https://github.com/ZeroEra99/AREcopter_Arduino_FlightController`

6 Conclusioni

Il progetto si è rivelato estremamente impegnativo, specialmente per quanto riguarda la comprensione della fisica e delle dinamiche di volo del quadricottero. Le limitazioni intrinseche del controllore Arduino hanno inoltre rappresentato una sfida aggiuntiva, richiedendo un'ottimizzazione accurata delle risorse e delle logiche di controllo.

Nonostante le difficoltà, è stato estremamente soddisfacente vedere il drone volare utilizzando un software completamente scritto da zero.

L'esperienza ha anche avuto un valore educativo significativo, permettendo di osservare concretamente le relazioni tra classi, moduli e sezioni di un sistema hardware e software integrato.