



**Francisco Freitas
Francisco Martins**

**Controlo de voo e monitorização remota de um
drone**

Turma P4, Projeto 4



**Francisco Freitas
Francisco Martins**

Controlo de voo e monitorização remota de um drone

Turma P4, Projeto 4

Relatório da unidade curricular de Projeto em Sistemas de Automação do Mestrado em Engenharia Mecânica, realizada sob orientação do professor José Paulo Santos.

Palavras-chave

drone, controlo, *NodeRed*, *ESP32*

Resumo

Este projeto teve como objetivo desenvolver e testar um sistema de estabilização para um drone, baseado na leitura de um acelerómetro e giroscópio integrados no MPU6500. Os dados dos sensores foram processados por um filtro de Kalman, permitindo estimar os ângulos de roll e pitch em tempo real. Com base nessas estimativas, foi implementado um controlador PID para ajustar a velocidade dos motores via ESCs. A afinação dos ganhos e parâmetros do filtro foi realizada através de uma interface gráfica em Node-RED, com comunicação via MQTT para um ESP32. Devido a limitações da fonte de alimentação inicialmente utilizada (células INR21700-50ME), que não garantiam corrente suficiente para voo livre, os testes foram conduzidos num sistema de pêndulo. Esta montagem permitiu no entanto validar o comportamento do algoritmo de estabilização, confirmando a viabilidade do controlo.

Repositório Github

<https://github.com/FSF-Toco/PSA-2025-Drone>

Conteúdo

1	Introdução	1
2	Objetivos	1
3	Esquema de Funcionamento	1
4	Funcionamento do drone	1
4.1	Voo Estacionário	1
4.2	Movimentação Frente/Trás e Esquerda/Direita	3
4.3	Auto-rotação (<i>Yaw</i>)	3
5	Especificações dos componentes	3
5.1	MPU6500	3
5.2	ESP32-WROOM	3
5.3	BLHeli	4
6	Leitura do Acelerômetro MPU6500	4
7	Acionamento dos Motores BLDC	5
8	Manipulação dos motores através do acelerômetro	6
8.1	Filtro de kalman	8
8.2	Implementação de ganhos	9
9	Implementação em NodeRed	10
10	Montagem Final	12
11	Dificuldades e soluções	13
12	Discussão de Resultados	14
13	Conclusão	14

Lista de Figuras

1	Esquema para a montagem de testes no laboratório	2
2	Esquema de comunicação entre componentes	2
3	Movimento frente/trás	3
4	Movimento esquerda/direita	3
5	Auto-rotação	3
6	Acelerômetro MPU6500	4
7	ESP32 WROOM	4
8	Montagem do ESP32 e MPU6500	5
9	Código para teste do acelerômetro (parte 1)	5
10	Formulas para cálculo da Inclinação	6
11	Código para teste do acelerômetro (parte 2)	6
12	Código para teste dos ESC	7
13	Implementação do filtro de Kalaman	8
14	Interface Node-Red	10
15	Blocos Node-Red	11
16	Esquema Elétrico Final	12
17	CAD do suporte das células	13

1 Introdução

No âmbito da cadeira de projetos em sistemas de automação, tem-se como objetivo desenvolver código para controlar um drone, garantindo a sua estabilização através do uso de sensores para detectar inclinações. Outro objetivo será a utilização do software Node-RED para nos auxiliar a monitorizar e controlar certos aspectos do drone. Assumindo que estes objetivos são concluídos, planeia-se também desenvolver algoritmos para movimentos horizontais do drone.

De modo a testar o controlo dos motores relativamente aos valores lidos pelo acelerómetro, foi utilizado um Esp32-Wroom ([Systems, 2023]), um sensor MPU6500 ([InvenSense, 2014]), quatro ESC da serie BLHeli ([Robotics, 2018]), e quatro motores BLDC do modelo X-ing. Nesta fase inicial foi efetuada a certificação do funcionamento destes componentes, assim como o planeamento dos sistemas de comunicação que vão ser utilizados.

2 Objetivos

1. Realizar a leitura do sensor MPU6500 através do protocolo I2C.
2. Calibrar os dados do acelerómetro com o filtro de Kalman.
3. Influenciar os motores com os dados do MPU6500.
4. Implementar ganhos para compensar o comportamento dos motores.
5. Criar uma interface com recurso a NodeRed e SQL para obter dados em tempo real e facilitar o controlo.

3 Esquema de Funcionamento

Foram elaborados dois esquemas para planear o projeto, a fig.1, com o objetivo de a mapear a interação entre os diversos dispositivos e componentes que compõem este projeto e a fig.2, um esquema *uml* que pretende explicar a lógica de comunicação entre estes componentes em sequência.

4 Funcionamento do drone

Antes de arranjar soluções para o problema proposto neste projeto, primeiramente é necessário compreender os movimentos que um drone do tipo *multi-rotor* é capaz de fazer ([Herrero, 2017]).

4.1 Voo Estacionário

Quando em voo estacionário o drone é capaz de manter uma posição e altitude constantes, sem subir nem descer, mantendo-se paralelo ao solo. Ser capaz de garantir o voo estacionário quando o drone está sujeito a situações adversas, como ventos fortes, é um dos principais focos deste projeto.

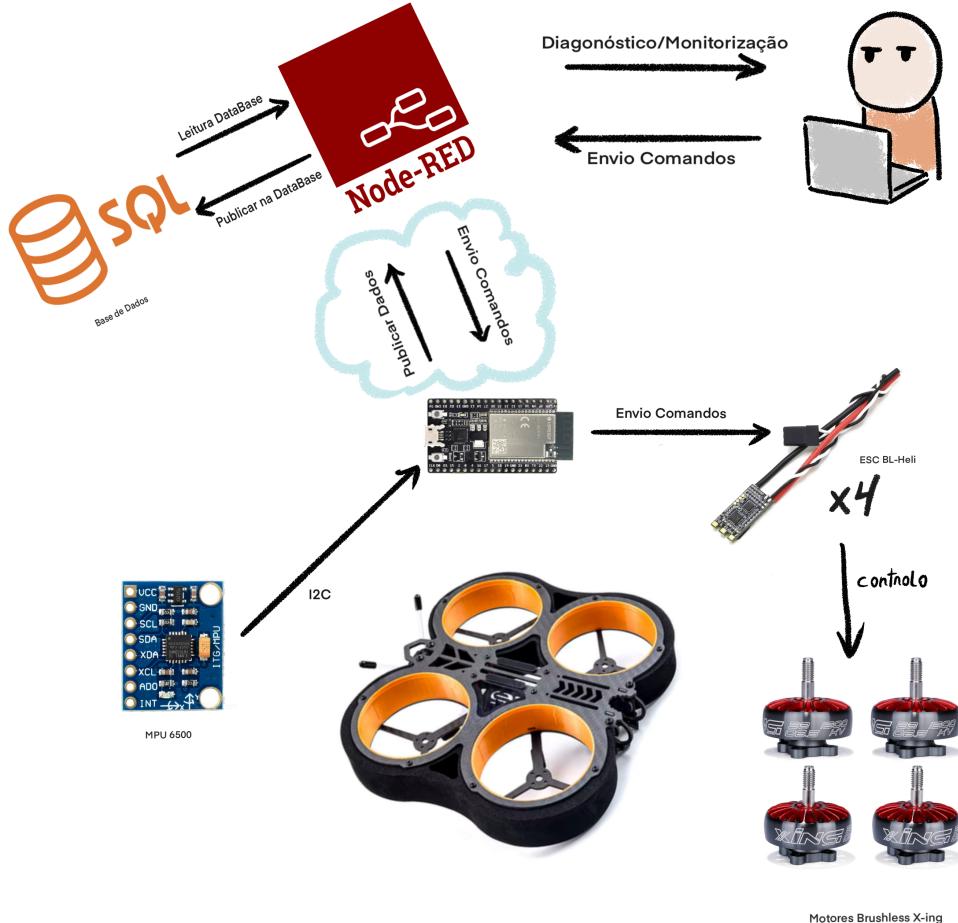


Figura 1: Esquema para a montagem de testes no laboratório

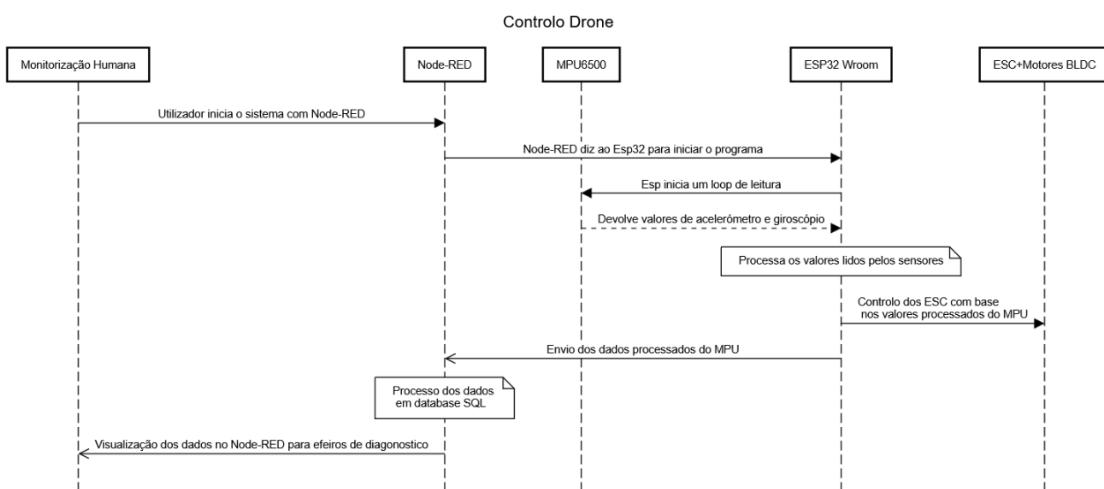


Figura 2: Esquema de comunicação entre componentes

4.2 Movimentação Frente/Trás e Esquerda/Direita

O movimento do drone para as quatro direções ocorre de uma simples forma. Quando definida a direção e sentido do movimento pretendido, os motores do lado oposto rodam com mais velocidade angular, permitindo que o drone se incline para a direção que o utilizador pretende que ele vá.

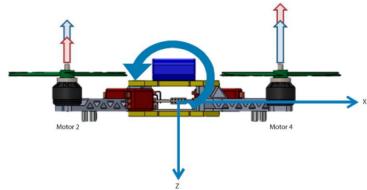


Figura 3: Movimento frente/trás

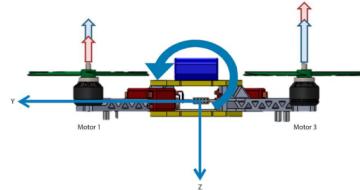


Figura 4: Movimento esquerda/direita

4.3 Auto-rotação (Yaw)

Num movimento de auto-rotação o drone roda em torno do eixo normal ao plano das hélices. Este movimento é obtido por fazer rodar os motores do primeiro e terceiro quadrante (sentido positivo) mais rápido que os do segundo e quarto quadrante (sentido negativo), ou vice-versa.

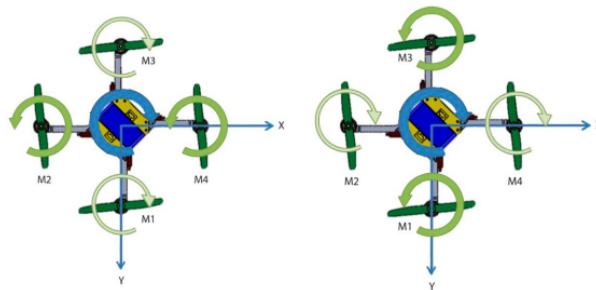


Figura 5: Auto-rotação

5 Especificações dos componentes

5.1 MPU6500

O MPU6500 (semelhante ao MPU9250) foi o sensor utilizado para este projeto, é um sensor amplamente utilizado em projetos de eletrónica e robótica, fig.6. Este componente integra 2 sensores diferentes: um acelerómetro de 3 eixos e um giroscópio de 3 eixos, para leitura das acelerações gravíticas e velocidades angulares, respetivamente. Para os fins deste projeto, ainda só foi utilizado na sua capacidade de acelerómetro.

5.2 ESP32-WROOM

O Esp32-Wroom (fig.7) foi escolhido para este projeto pela sua conveniência. É versátil, funciona com 3.3V, o que é suficiente para a alimentação dos sensores, tem os pinos SCL

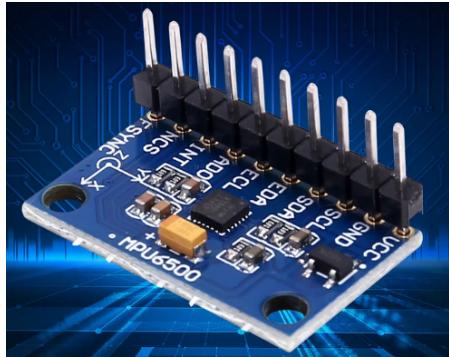


Figura 6: Acelerômetro MPU6500

e SDA, para comunicar usando I2C, e finalmente possui pinos com capacidade PWM, necessários para acionar os ESC. Concluindo, era um componente com a capacidade de cumprir a tarefa pretendida e com capacidades de *wifi* do ESP32, o que permite a publicação direta para o Node-RED.

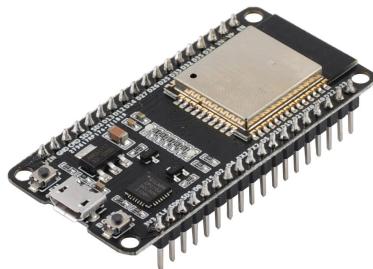


Figura 7: ESP32 WROOM

5.3 BLHeli

Os ESC utilizados são da série BLHeli, e são feitos para motores *brushless*, e estes são especificamente usados maioritariamente no controlo de drones. Este modelo em específico, funciona quando alimentado com 7,4 a 21 Volts. Para este projeto os 4 ESCs seriam alimentados com uma fonte de 12 volts (inicialmente).

6 Leitura do Acelerômetro MPU6500

O esquema elétrico desenvolvido para este sensor estabelece a ligação entre o ESP32 e o MPU6500, através da comunicação I2C. Os sinais SDA (*data*) e SCL (*clock*), foram ligados respetivamente aos pinos GPIO21 e GPIO22 do ESP32. A alimentação do MPU6500 é assegurada através da ligação dos pinos VCC e GND ao ESP32, fornecendo 3.3V (fig.8).

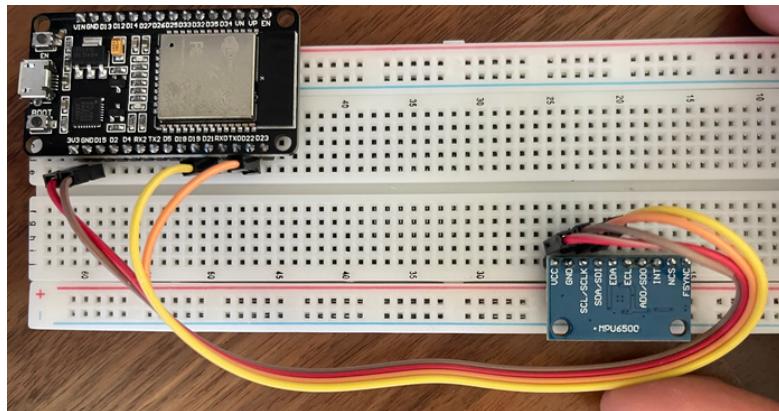


Figura 8: Montagem do ESP32 e MPU6500

Para facilitar o uso do protocolo I₂C na leitura deste componente, utilizou-se a biblioteca `MPU9250_asukiaaa.h` (compatível com o MPU-6500), a família de bibliotecas asukiaaa é utilizada para simplificar o processo de leitura de diversos sensores. A biblioteca `Wire` é usada para definir os pinos do ESP32 associados ao SCA e ao SCL.

Inicialmente foi elaborado um código de teste para verificar o correto funcionamento do sensor. A fig.9 mostra o uso da biblioteca asukiaaa.h para fazer a leitura dos valores do acelerômetro, assim como a sua iniciação.

```

1  #include <Wire.h>
2  #include <MPU9250_asukiaaa.h>
3
4  // A biblioteca do MPU9250 funciona no MPU6500
5  MPU9250_asukiaaa mySensor;
6
7  void setup() {
8      Serial.begin(115200);
9      Wire.begin();
10
11     mySensor.setWire(&Wire);
12     mySensor.beginAccel(); // Iniciar o aaccelerometro
13
14     delay(100);
15     Serial.println("MPU6500 (accelerometer) initialized!");
16 }
```

Figura 9: Código para teste do acelerômetro (parte 1)

Para concluir a verificação do funcionamento do acelerômetro foi elaborado um simples código onde para cada eixo do sensor foi criada uma variável da aceleração. Através das fórmulas representadas na fig.10, foram calculados os ângulos de inclinação, *Roll* e *Pitch* do sensor, como demonstra na fig.11.

7 Acionamento dos Motores BLDC

O acionamento dos motores é bastante simples, assumindo que o ESC é inicializado corretamente. As velocidades são controladas através de sinais PWM, que com a biblioteca

$$\theta = \tan^{-1} \left(\frac{A_{x,out}}{\sqrt{A_{y,out}^2 + A_{z,out}^2}} \right)$$

$$\psi = \tan^{-1} \left(\frac{A_{y,out}}{\sqrt{A_{x,out}^2 + A_{z,out}^2}} \right)$$

$$\phi = \tan^{-1} \left(\frac{\sqrt{A_{x,out}^2 + A_{y,out}^2}}{A_{z,out}} \right)$$

Figura 10: Formulas para cálculo da Inclinação

```

17 void loop() {
18     mySensor.accelUpdate(); // Lê os dados da aceleração gravitacional
19
20     float ax = mySensor.accelX(); // Aceleração em X
21     float ay = mySensor.accelY(); // Aceleração em Y
22     float az = mySensor.accelZ(); // Aceleração em Z
23
24     // Conversão da aceleração para ângulos
25     float pitch = 180 * atan2(ax, sqrt(ay*ay + az*az))/PI;
26     float roll = 180 * atan2(ay, sqrt(ax*ax + az*az))/PI;
27
28     Serial.print("Roll(Y): ");
29     Serial.print(roll);
30     Serial.print("\n");
31     Serial.print("°, Pitch(X): ");
32     Serial.print(pitch);
33     Serial.println("°");
34
35     delay(200);
36 }
```

Figura 11: Código para teste do acelerómetro (parte 2)

ESP32Servo.h podem ser enviados com o simples comando ”esc.writeMicroseconds()”, o valor em microssegundos a usar deve estar numa gama de valores definidos no setup, usando a função ”esc.attach()”, nesta função deve ser especificado o pino do ESP32 a utilizar para enviar os sinais PWM, o valor máximo de comprimento de onda em microssegundos, assim como o seu valor máximo. Segundo a ficha técnica do ESC, quaisquer valores entre 1000 e 2000 poderiam ser escolhidos, sendo o único requisito que a diferença entre os dois extremos fosse superior a 140, para simplificar eu simplesmente usei 1000 como mínimo e 2000 como máximo. Estes ESC podem também ser usados em motores com dois sentidos, tendo isto em conta, valores acima de 1500 correspondem a uma velocidade num sentido, e valores abaixo de 1500, seriam velocidades no sentido contrário. Como o motor a usar neste projeto só possui rotação num sentido, qualquer valor abaixo de 1500 corresponde à paragem do motor. Para calibrar o ESC, a documentação fornece detalhes sobre como interpretar o seu feedback de modo a iniciá-lo corretamente, após algumas tentativas a analisar os sons que o ESC fornecia como *feedback*. O método que se mostrou consistente foi o de enviar uma onda de 2000 microssegundos, e esperar um intervalo decente de tempo, sendo este o valor máximo, e de seguida enviar um sinal de 1000 microssegundos, o valor mínimo, seguido de um segundo intervalo de espera. A fig.12 representa um código que usei para testar os motores com este método, o código inicializa o motor como descrito, e após isso no loop deixa-o num estado aberto a ser controlado livremente através de comandos simples. Para os efeitos deste projeto, a fonte de alimentação a que tivemos acesso foram as células de lítio intr21700-50me 3.6V (ref), estas serviram para testar os motores numa fase inicial, mas levaram-nos a certos problemas no que toca à montagem final do drone.

8 Manipulação dos motores através do acelerómetro

A estabilidade de voo do drone desenvolvido neste projeto assenta na capacidade de processar continuamente os dados fornecidos pelos sensores inerciais — acelerómetro e giroscópio integrados no MPU6500 — e de, a partir deles, gerar sinais de controlo adequados para os motores. Para isso, foi necessário implementar um filtro de Kalman que combinasse as leituras ruidosas e complementares destes sensores, produzindo estimati-

```
#include <ESP32Servo.h>

const int pinESC = 4; // ESP pin connected to the ESC signal

Servo esc;

void setup() {
    Serial.begin(115200);
    pinMode(pinESC, INPUT);

    esc.attach(pinESC, 1000, 2000); // Attach the ESC to the pin, setting PWM range
    delay(3000); // Wait 3 seconds for the ESC to initialize

    // Step 1: Send max throttle signal
    esc.writeMicroseconds(2000); // Max throttle (full speed)
    Serial.println("2000ms");
    delay(3500); // Hold the max throttle for 3 seconds (ESC stores max throttle)

    // Example: Run at medium speed
    esc.writeMicroseconds(1000); // Adjust value as needed for your application
    Serial.println("1000ms");
    delay(3500);

    delay(5000);
}

void loop() {
    esc.writeMicroseconds(1200);
    Serial.println("1200ms");
    delay(2000); // Wait for 2 seconds
}
```

Figura 12: Código para teste dos ESC

vas fiáveis dos ângulos de *roll* e *pitch*. Com base nesses valores, aplicou-se um controlador *PID* para corrigir desvios de orientação em tempo real, ajustando a resposta dos ESCs. Este processo de fusão sensorial e controlo em malha fechada foi essencial para garantir o equilíbrio dinâmico do drone durante o voo.

8.1 Filtro de kalman

Nesta fase do projeto, os dados provenientes do acelerómetro e do giroscópio integrados no MPU6500 foram processados através de um filtro de Kalman (figura 13). Este algoritmo realiza uma fusão sensorial, combinando a velocidade angular, suscetível a deriva cumulativa, com a aceleração gravítica predominantemente afetada por ruído de alta frequência, de forma a obter uma estimativa mais robusta e estável dos ângulos de *roll* e *pitch*.

```
rollKalman.setQangle(0.01f); rollKalman.setQbias(0.003f); rollKalman.setRmeasure(0.01f);
pitchKalman.setQangle(0.01f); pitchKalman.setQbias(0.003f); pitchKalman.setRmeasure(0.01f);
rollKalman.setAngle(roll_init);
pitchKalman.setAngle(pitch_init);
```

Figura 13: Implementação do filtro de Kalaman

No filtro de Kalman implementado pela biblioteca “Kalman.h” as três variáveis fundamentais são *setQangle*, *setQbias* e *setRmeasure*. Estas permitem a configuração de como o filtro pondera o ruído e a incerteza da medição.

setQangle

O parâmetro ”*setQangle*” define a variância do ruído de processo associado à estimativa do ângulo, ou seja, regula quanto o filtro confia na modelação dinâmica do ângulo versus as medições; valores mais elevados tornam o filtro mais reativo a alterações súbitas, mas também mais sensível a ruídos.

setQbias

O parâmetro ”*setQbias*” controla o “ritmo” a que o filtro de Kalman ajusta o desvio sistemático do giroscópio ao longo do tempo. Ao atribuir um valor baixo a este parâmetro, estamos a dizer ao filtro que o erro do sensor é relativamente constante, pelo que apenas fará pequenas correções graduais. Por outro lado, um valor alto indica que o erro pode variar com mais rapidez, e o filtro irá então adaptar-se de forma mais agressiva a possíveis desvios sistemáticos do giroscópio.

setRmeasure

O valor de ”*SetRmeasure*” corresponde à variância do ruído de medida proveniente do acelerómetro, calibrando a confiança atribuída às leituras externas; ao aumentar o *Rmeasure*, o filtro dá mais peso ao modelo interno em detrimento das medições, reduzindo a influência de leituras potencialmente ruidosas.

8.2 Implementação de ganhos

Após a estimativa dos ângulos de orientação através do filtro de Kalman, é necessário aplicar uma estratégia de controlo que permita corrigir os desvios entre a posição atual do drone e a posição desejada. Para este efeito, foi implementado um controlador PID (Proporcional–Integral–Derivative), uma abordagem clássica em sistemas de controlo de tempo real, devido à sua simplicidade e eficácia. Este controlador calcula a correção a aplicar aos motores com base no erro atual, no histórico de erros e na tendência de variação, ajustando continuamente a potência fornecida pelos ESCs. A correta afinação dos ganhos K_p , K_i e K_d é essencial para adaptar a resposta do sistema, garantindo estabilidade, rapidez de reação e minimização de oscilações.

K_p

O ganho proporcional (K_p) define a força da resposta imediata ao erro entre o ângulo desejado e o ângulo estimado em qualquer dado momento. Em outras palavras, é a componente que reage diretamente à diferença instantânea entre o objetivo e a realidade. Valores mais elevados de K_p aumentam a agressividade da correção, permitindo que o sistema reaja mais rapidamente a desvios, no entanto, um K_p excessivo pode resultar em oscilações, instabilidade ou até vibrações perceptíveis no drone, especialmente em sistemas com atrasos ou ruído.

K_i

O ganho integral (K_i) corrige erros persistentes ao longo do tempo, somando o erro acumulado em períodos sucessivos. Esta componente é crucial para compensar desequilíbrios estruturais, desvios constantes ou erros sistemáticos que o ganho proporcional sozinho não resolve. Um K_i corretamente ajustado elimina offsets estáticos, garantindo que o drone se mantenha nivelado mesmo com pequenas assimetrias. No entanto, valores excessivos de K_i podem provocar *overshoot* (ultrapassagem do valor alvo) e instabilidades acumulativas, especialmente se houver ruído nas medições ou variações rápidas nos comandos.

K_d

O ganho derivativo (K_d) atua sobre a taxa de variação do erro, funcionando como um amortecedor que antecipa mudanças rápidas. Serve para suavizar a resposta do sistema, reduzindo oscilações e amortecendo movimentos bruscos, como entradas de comando repentinas ou correções causadas por turbulência. Um valor adequado de K_d melhora a estabilidade geral, conferindo ao drone um comportamento mais previsível e controlado. Por outro lado, valores demasiado altos podem introduzir atraso na resposta e reduzir a resposta do drone a perturbações, além de amplificar o efeito de ruído de alta frequência nos sensores.

9 Implementação em NodeRed

De modo a implementar os valores do acelerómetro no código de controlo dos motores, há a necessidade de ter a combinação adequada de valores para os filtros de Kalman, assim como para os ganhos do algoritmo de calibração em si, que permitem ao drone estabilizar em resposta a perturbações que possam ocorrer.

Para obter estes valores planeámos observar o comportamento do drone em funcionamento à medida que os mesmos eram alterados, e para isto, a nossa *interface* Node-Red foi essencial. Através de um broker MQTT, é possível fazer comunicação em tempo real entre o Node-Red e o nosso Esp, permitindo não só a mudança de variáveis em tempo real, mas também a visualização das mesmas no nosso computador para efeitos de diagnóstico.

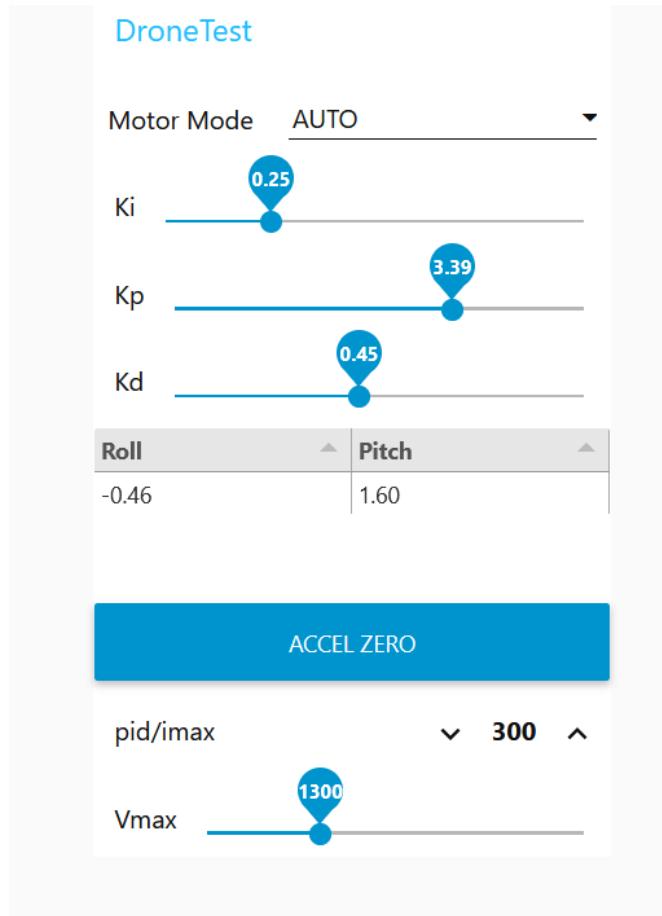


Figura 14: Interface Node-Red

A figura fig.14 representa a nossa interface que foi utilizada para todas as nossas interações com o drone, estas incluíram:

- Escolha de modo de voo: seleção entre 3 modos de funcionamento, o modo off, onde o drone se mantém não operacional; o modo básico, onde todos os motores rodam à velocidade máxima escolhida; e o modo automático, onde os motores vão realmente ser influenciados pelos valores do acelerómetro de modo a tentar equilibrar-se.
- Ganhos do sistema PID: atualização em tempo real dos valores de Kp, Ki, e Kd que o algoritmo de calibração está a utilizar.
- Accel Zero: Botão desenhado para a definição do "zero" do acelerómetro, de modo a definir corretamente a posição horizontal a que o drone deve tentar regressar em resposta a qualquer perturbação.
- PID/imax: Definição de um valor máximo para o quanto o algoritmo de correção pode alterar a velocidade de cada motor, foi implementado numa fase onde teorizámos que isto poderia ser um problema a prevenir, mas acabou por ser desnecessário, sendo que as correções excessivas eram corrigíveis totalmente através da atualização dos ganhos.
- Vmax: Escolha de um valor para a velocidade dos motores, no modo básico todos os motores rodam a esta velocidade, enquanto que no modo automático este valor serve como a base à qual é depois aplicada a correção do algoritmo de calibração.

Na figura fig.15 é possível verificar a organização dos blocos por detrás desta interface.

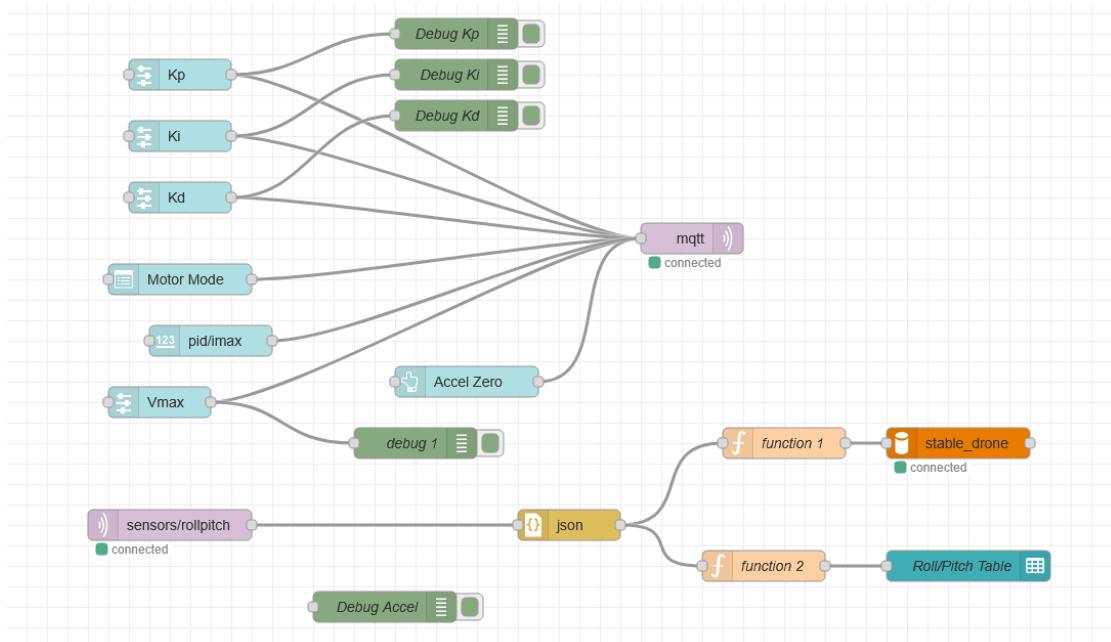


Figura 15: Blocos Node-Red

10 Montagem Final

Para a realização dos testes físicos com o drone, foi necessária a montagem completa do sistema. Este processo envolveu tanto a vertente eletrónica como o projeto mecânico.

No que diz respeito à parte eletrónica, conforme ilustrado na Figura 16, destaca-se a inversão dos polos do ESC em dois dos motores, com o objetivo de garantir que duas das hélices do drone rodassem no sentido anti-horário. Adicionalmente, é importante referir que o circuito foi projetado assumindo que o ESP32 está alimentado através da sua entrada *micro-USB*, não sendo funcional caso esta condição não seja atendida.

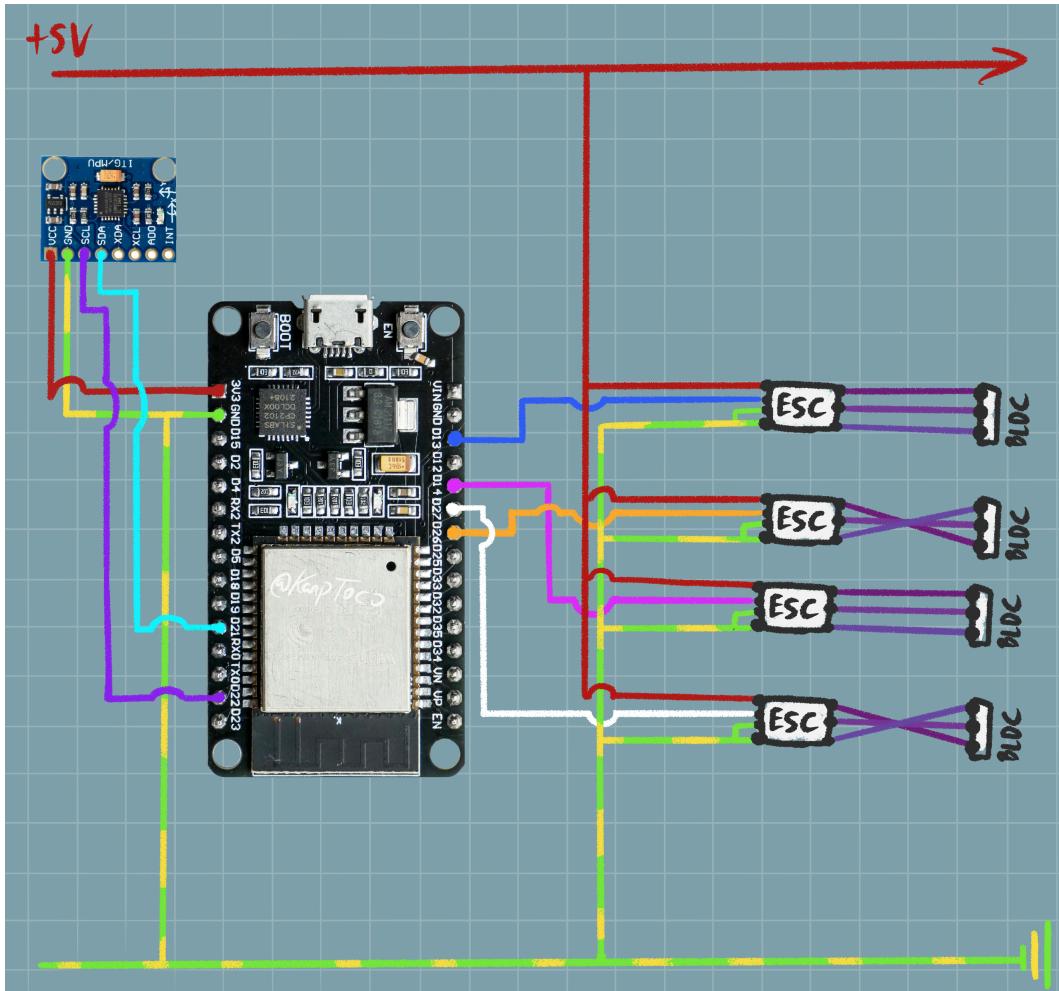


Figura 16: Esquema Elétrico Final

Na vertente mecânica da montagem, a principal dificuldade consistiu em encontrar uma solução para estabilizar as células de lítio que iriam fornecer a alimentação para os motores. Estas células, sendo relativamente grandes para o tamanho do drone, não possuíam uma solução pré-definida de fixação. Assim, optou-se pelo desenvolvimento de uma peça em CAD, cuja função seria servir de base para os componentes, conforme ilustrado na Figura 17. A peça foi projetada de forma a suportar até 4 células, uma vez que não se sabia ao certo se duas seriam suficientes para permitir o levantamento do

drone.

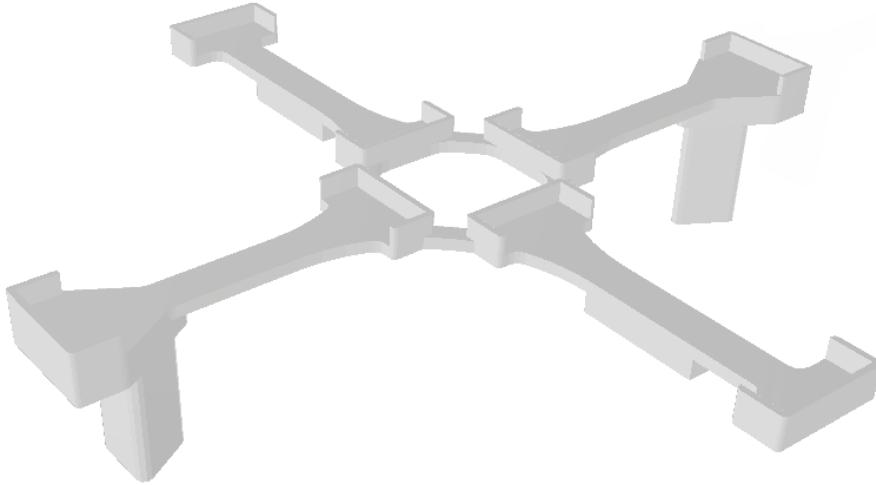


Figura 17: CAD do suporte das células

11 Dificuldades e soluções

Durante os ensaios iniciais, uma configuração com duas células em série (2S, 7,4 V) revelou uma rotação insuficiente para gerar sustentação, embora não se tenham detetado falhas de comunicação nem disparo das proteções dos ESCs, uma vez que a corrente solicitada se manteve baixa. A transição para quatro células em série (4S, 14,8 V) permitiria atingir rotações superiores, em teoria. No entanto, acima de cerca de 70 % de aceleração, os motores deixaram de aumentar a velocidade, registando-se quebras na ligação Wi-Fi.

A análise da telemetria indicou quedas abruptas de tensão, resultado da limitação de corrente das células INR21700-50ME, cuja taxa de descarga se revelou insuficiente para sustentar as novas exigências de corrente. O afundamento de tensão ativou as proteções internas dos ESCs, interrompendo a alimentação.

A experimentação confirmou o que já havia sido antecipado nas fases anteriores do projeto, nomeadamente que estas células não são adequadas para este tipo de aplicação, não apenas devido à sua taxa de descarga relativamente baixa, mas também pelo peso relativamente elevado. Para este tipo de drone, seria aconselhável a utilização de baterias LiPo, uma vez que apresentam altas taxas de descarga e maiores densidades de potência (mais potência por unidade de peso) [Robotics, 2023]. .

Para os objetivos deste projeto, o drone não necessitou de capacidade para se sustentar de forma a permitir a realização dos testes ao sistema de calibração. As variáveis de Kalman e os ganhos do PID irão variar em função de um drone em voo livre, com uma distribuição de peso diferente devido ao uso de uma alimentação distinta. No entanto, para testar a capacidade do nosso sistema em estabilizar a montagem do drone, foi utilizado um sistema de pêndulo, no qual o drone foi suspenso por um fio e posteriormente ligado em modo automático. Com esta montagem experimental, o objetivo foi induzir oscilações no drone e analisar como o algoritmo respondia, ajustando os ganhos até que o drone se estabilizasse corretamente.

12 Discussão de Resultados

Os testes realizados nesta montagem experimental permitiram atingir a estabilidade para uma determinada velocidade dos motores. No entanto, os comportamentos do drone sob essas condições revelaram a necessidade de reavaliar o método a ser considerado num cenário no qual este projeto fosse aplicado a voos reais, utilizando a alimentação adequada.

Verifica-se também a possibilidade do ESP32 não ser capaz de suportar este projeto numa escala real. A estabilização foi alcançada para uma velocidade relativamente baixa, e testes a velocidades superiores, regra geral, demonstraram dificuldades em acompanhar o ritmo das oscilações, independentemente dos ganhos aplicados, resultando muitas vezes num reset do Esp em si.

13 Conclusão

Em suma, o processo descrito neste projeto permitiu realizar testes ao algoritmo de calibração para um drone, recorrendo a um sistema de pêndulo com oscilações causadas por forças externas. Não foi possível montar o drone de forma a que este se sustentasse, devido às limitações das fontes de alimentação disponíveis.

Com base nas observações feitas, surgiu a dúvida sobre a capacidade do ESP32 em realizar a calibração do drone numa situação de voo livre. Apesar das dificuldades encontradas em termos de *hardware*, verificou-se com sucesso a viabilidade de utilizar o algoritmo para equilibrar o drone com base nas oscilações lidas pelo acelerômetro.

Bibliografia

[Herrero, 2017] Herrero, R. A. (2017). Drone design. Project Report, Dumlupınar University, Industrial Engineering Department. Cover page retrieved from project documentation.

[InvenSense, 2014] InvenSense (2014). Mpu-9250 product specification revision 1.1. Accessed: 2025-01-20.

[Robotics, 2018] Robotics, B. (2018). Blhelis manual silabs rev16.x. Accessed: 2025-01-20.

[Robotics, 2023] Robotics, T. (2023). A guide to lithium polymer batteries for drones. Acedido em 11 de junho de 2025.

[Systems, 2023] Systems, E. (2023). Esp32-wroom-32 datasheet. Accessed: 2025-01-20.