Projeto Integrador 2021-22

Felipe Coutinho & Cauã Veiga
DFA - Faculdade de Ciências da Universidade do Porto

26 de junho de 2022

1 Introdução

Nesta unidade curricular, desenvolvemos um controlador para equilíbrio de um protótipo de *drone* com 1 grau de liberdade, através implementação de um algoritmo PID num microcontrolador *ESP32-PICO-KIT*. Buscamos estabilizar o eixo longitudinal de uma haste com motores ligados a hélices nas extremidades e um eixo de rotação no seu centro geométrico, como mostrado na figura 1.

Para isso, foram utilizados motores sem escovas, retirados de um *drone* inativo que nos foi cedido para efeito deste trabalho. Estes motores foram por sua vez controlados pela ESP32, através de unidades ESC (*Electronic Speed Control*), também retiradas do *drone*. Ainda, utilizou-se um sensor *MPU-6050*, que integra um acelerómetro de 3 eixos e um giroscópio.

Finalmente, de modo a obter informação quantitativa sobre a estabilização, se aproveitou das funcionalidades *Wi-Fi* da placa para, a partir de um servidor hospedado nela, acessar aos dados em tempo real por computadores conectados à mesma rede.

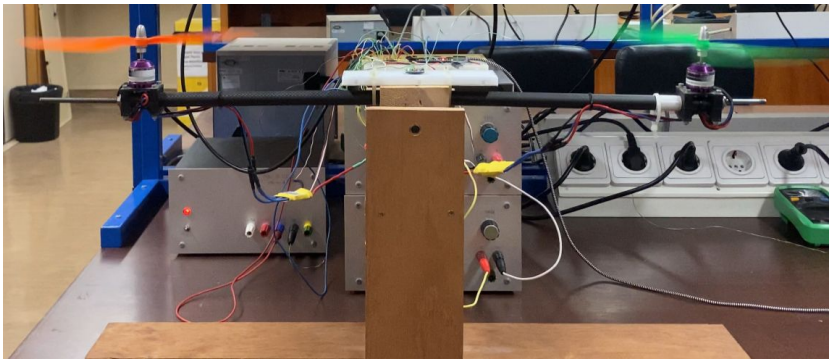


Figura 1: Montagem protótipo drone com 1 grau de liberdade.

2 Seleção das componentes

Como os motores e as unidades ESC foram reaproveitadas, numa primeira etapa foi necessário identificar as características destas componentes.

Os motores estão dimensionados para serem alimentados com 2-4 baterias S LiPo. Para evitar o uso das baterias, decidimos substituí-las por fontes. Para tal, utilizamos uma tabela para encontrar o intervalo de tensão correspondente, pelo que fixamos a alimentação em cerca de $8\,V$. Além disso, tendo em conta que não necessitaríamos de atingir o torque máximo do motor (correspondendo a um empuxo de cerca de $1\,Kg$), optamos por fontes capazes de ceder até $1.5\,A$ (num regime de operação com empuxo máximo em torno dos $200\,g$). Outra motivação para esta escolha é reduzir riscos associados as hélices, que não são protegidas.

Numa primeira fase, utilizamos uma fonte comum para os dois motores. Contudo, apesar de satisfazer a tensão necessária, acabávamos por puxar demasiada corrente pelo que a fonte acusava um curto-circuito. Assim, decidimos por incorporar 2 fontes, uma para cada motor.

Estudamos o funcionamento das unidades ESC, que permitem o controlo dos motores sem escovas. Cada uma possui ao todo 5 conexões, sendo duas destinadas à alimentação dos motores e as demais ao sinal de controlo e alimentação da própria ESC. O controlo é feito através de um sinal PWM (*Pulse Width Modulation*), com frequência de $50\,Hz$ e tensão pico-a-pico de $5\,V$ - permitindo o controlo do motor sem escovas como se de um servomotor se tratasse.

Figura 2: Esquema das conexões da ESC. Os fios grossos, exteriores, destinam-se a alimentação do motor. Os fios vermelho e preto mais finos alimentam a ESC e estabelecem uma referência para o sinal de controlo V_{in}, que é fornecido pelo fio branco.

Escolhemos o microcontrolador *ESP32-PICO-KIT* por três razões principais. Na unidade curriular Eletrónica Digital e Microprocessadores (EDM) , que frequentamos em simultâneo, trabalhamos com este modelo, pelo que já tínhamos certa familiaridade. Além disso, o modelo permite programação na linguagem *MicroPython*, extremamente similar ao *Python* e como tal, mais confortável para nós. Por último, o modelo inclui funcionalidades *Wi-Fi*, que intencionamos explorar. Em contrapartida, esta placa trabalha em lógica de $3.3\,V$, o que implica em adicionar um circuito de conversão de nível lógico para poder efetuar o controlo a partir do sinal PWM.

Enfim, escolhemos o sensor MPU-6050 por ser de baixo custo e com ampla

documentação sobre seu uso disponível. Como este contém um acelerómetro e um giroscópio, permite ainda combinar medições obtidas por cada um num valor final para o ângulo mais estável e mais exato. Além disso, sua leitura é feita através do protocolo I2C, que também foi estudado em EDM.

3 Desenvolvimento inicial

Numa primeira etapa do trabalho, ainda não estava disponível a estrutura de madeira para montar os motores. Sendo assim, optou-se por desenvolver os módulos para leitura do sensor e controle do motor em separado, com a perspectiva de integrá-los quando a estrutura mecânica estivesse realizada.

3.1 Sensor MPU-6050

Como referido anteriormente, o protocolo utilizado pelo MPU-6050 é o I2C. Tratando-se de um protocolo síncrono, funciona a partir de um sinal de relógio (SCL) e um sinal de transmissão de dados (SDA). A linguagem MicroPython contém as funções básicas para estabelecer este tipo de comunicação, sendo o sinal do relógio e a emissão dos *bytes* feita automaticamente. Sendo assim, recorrendo à *datasheet* do sensor, basta ler os endereços corretos para aceder aos dados desejados.

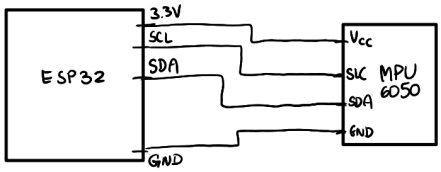


Figura 3: Esquema das conexões do sensor MPU-6050.

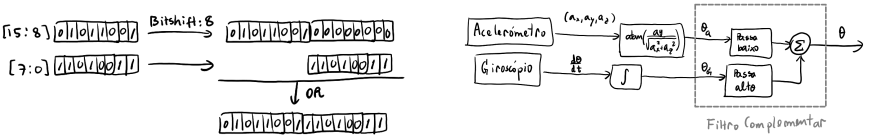
É importante notar que cada medição é representada 16 bits, em complemento para 2, mas estão armazenadas em dois registros de 8 bits cada. Sendo assim, é necessário agregar a informação dos dois registros num único valor para obter o resultado da medição. Realizar estas operações de modo eficiente mostrou-se importante para que a latência vinda da execução do código não prejudique a taxa de amostragem; o procedimento está exemplificado no fluxograma 4.

Além disso, a informação na *datasheet* permitiu a conversão de cada leitura para as unidades corretas, além da seleção da gama de medição mais adequada ($\pm 250\,\mathrm{deg}/s$ para a leitura do giroscópio e $\pm 2g$ para leitura do acelerómetro), maximizando a resolução do instrumento. Enfim, adicionou-se uma rotina de calibração dos sensores, para evitar erros sistemáticos de desvio.

O ângulo θ que buscamos medir é obtido das leituras do acelerómetro através da expressão $\theta = \arctan \frac{a_y}{\sqrt{a_x^2 + a_z^2}}$; contudo, existirão outras acelerações sobre o

sensor, de frequência elevada, que não a gravítica (constante). Por outro lado, como o giroscópio lê o valor de $\frac{\mathrm{d}\theta}{\mathrm{d}t}$, pode-se obter θ integrando sua saída. Evidentemente, o resultado apresentará um *drift*, devido a desvios que o giroscópio possa ter (mesmo após a calibração).

Para combinar os dois resultados, foi utilizado um filtro complementar, esquematizado em 4. O princípio de funcionamento consiste em extrair da leitura do acelerómetro apenas a informação relativa a componente DC (gravítica), aplicando um filtro passa-baixo. Por outro lado, aplicando um passa-alto na leitura do giroscópio, pretende-se remover os transientes longos associados a integração do desvio.



(a) Leitura dos registros

(b) Processamento dos dados

Figura 4: Esquema ilustrando rotina para aceder aos registros do MPU-6050 e combinação das leituras do acelerómetro e giroscópio para obter o valor do ângulo através de um filtro complementar

Referências: [1, 2, 3, 4, 5, 6]

3.2 Motor sem escovas

Os motores sem escovas têm um princípio de funcionamento relativamente complexo, sendo operados por um sinal AC trifásico criado pelas unidades ESC a partir de um sinal PWM.

Sendo assim, o esquema de controlo é efetivamente análogo ao de um servomotor, bastando fornecer à ESC um sinal PWM com frequência de $50\,Hz$, amplitude de $5\,V$ e *duty-cycle* entre $[5,\,10]\,\%$. Assim, a potência dos motores é controlada diretamente fazendo variar o *duty-cycle* do sinal PWM.

As unidades ESC utilizadas contém ainda um BEC (*Battery Eliminator Circuit*), pelo que o sistema necessitaria de apenas uma bateria para alimentar tanto a ESC ($5\,V$) e o motor ($8-12\,V$). Contudo, como a estrutura tem apenas 1 grau de liberdade, optamos por alimentar o motor com uma fonte externa, evitando recorrer ao uso de baterias.

Considerando que a placa ESP32 funciona em lógica de $3.3\,V$, para alimentar a ESC e gerar o sinal PWM adequado, é necessário ainda subir a sua tensão para $5\,V$.

Numa primeira montagem , isto foi feito recorrendo a dois amplificadores operacionais TL081 em configuração não-inversora (um para cada sinal PWM). Escolhendo para as resistências R_1, R_2 valores satisfazendo $R_2 \approx 0.51 R_1$, obtém-se o ganho necessário. Esta montagem, numa etapa mais avançada, foi substituída.

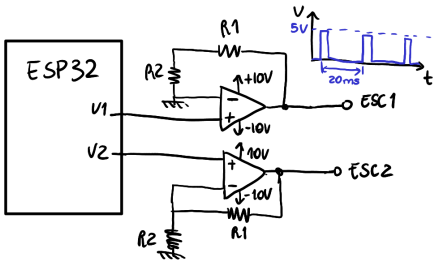


Figura 5: Circuito conversor de nível lógico $3.3\,V \rightarrow 5\,V$ através de amplificadores não-inversores

Após o controle dos motores ser validado, identificou-se o sentido do empuxo gerado por eles. Assim, uma vez adquirida a estrutura mecânica, seria possível posicioná-los de modo a obter empuxos em sentidos contrários. Este ponto é crucial tendo em vista que os motores utilizados permitem rotação em apenas um sentido.

3.3 Calibração das ESCs

Como mencionado nas secções 2 e 3.2 as ESCs funcionam a partir de um sinal PWM com frequência de 50Hz e duty-cycle entre [5, 10] %, entretanto para isso é necessária uma rotina de calibração antes de que as ESCs possam ser usadas.

Após definir a frequência do sinal PWM a $50\,Hz$ e garantir que o sinal de entrada as ESCs terão tensão de $5\,V$ o processo de calibração resume-se em definir um duty-cycle mínimo e em seguida um duty-cycle máximo. Para tal, como já mencionado, define-se o mínimo a 5% e o máximo a 10%, representando 51 e 102 respectivamente, como inteiros em 10 bits, estabelecendo uma gama de operação linear entre a potência mínima e máxima do motor.

Tendo em vista a gama de operação determinada realizou-se uma normalização nos valores de duty-cycle, apresentada na eq.(1), de modo fixar a entrada entre $[0, 1]$ no programa principal, facilitando por exemplo a integração com PID:

$$Duty = \left(\frac{X+1}{20}\right) \times 1023 \tag{1}$$

... onde *Duty* é o duty-cycle que será gerado o novo sinal PWM e X é o input entre $[0, 1]$.

3.4 Comunicação Wi-Fi

Foi escolhido utilizar as funcionalidades Wi-Fi da ESP32 de modo a registrar dados do funcionamento do sistema para uma análise quantitativa de sua performance.

A abordagem escolhida foi através dos protocolos de comunição *TCP/IP* "Transmission Control Protocol/Internet Protocol", apresentado na figura 6,

onde a ESP32 funcionará como um servidor e um portátil pessoal (onde os dados são guardados) será um cliente.

O primeiro passo é estabelecer ligação a uma rede comum, tanto com a ESP32 tanto com o portátil. Para tal foi utilizada a rede móvel 4G do telemóvel.

Uma vez ligados a rede do telemóvel é possível prosseguir para a comunicação TCP em sí, onde será necessário dois scripts diferentes. No caso do servidor (ESP32) utilizando a biblioteca *socket* utilizamos o método *.bind()* (onde define-se o IP que será usado) e *.listen()*, para além de um ciclo while a aceitar conexões com através do método *.accept()*. Uma vez feito isso a ESP32 está agora configurada para agir como um servidor e estará sempre a escutar conexões de clientes conectados ao mesmo IP, assim que uma conexão é estabelecida é então lida o pedido feito pelo cliente e em seguida enviados os dados no formato json codificados a 8 bits pelo método *utf-8*, neste caso enviamos o anglo atual e o instante da leitura.

Já para o cliente é utilizado o método *.connect()* para se conectar ao respectivo IP, é então criado um ciclo para fazer pedidos ao servidor a uma frequência de 10Hz, após isso decodificamos os dados recebidos e são posteriormente guardados localmente no portátil.

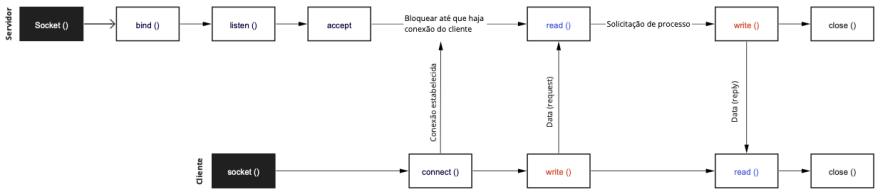


Figura 6: Fluxograma funcionamento protocolos TCP/IP

3.5 Controlador PID

Para realizar o controlo dos motores, utilizou-se um algoritmo PID.

O algoritmo baseia-se em calcular o ajuste necessário à potência dos motores, de modo a manter o valor de θ medido pelo sensor próximo a um valor fixado (designado ponto de operação) $\theta_{sp} = 0°$. A resposta u é então calculada a partir do erro $e = \theta - \theta_{sp}$ como:

$$u = K_p e + K_d \frac{de}{dt} + K_i \int e \, dt \equiv P + D + I$$

As constantes K_p, K_i e K_d são determinadas empiricamente por um processo de calibração.

O termo P é a contribuição mais fundamental para o controlador, que faz com que a resposta do sistema seja tão maior quanto o erro. Contudo, esta contribuição se utilizada sozinha levaria a transientes longos e o ponto de operação do sistema teria $u \neq 0$. A inclusão do termo D ajuda a diminuir a duração

do transiente, pois ajusta a resposta no tempo presente baseando-se numa estimativa do erro no instante seguinte. Por outro lado, o sistema ainda poderia estabilizar em torno de um ponto de equilíbrio diferente de θ_{sp}, pelo que o termo I garante que nesta situação, o erro seja eventualmente compensado uma vez que é acumulado ao longo do tempo no valor da resposta.

É importante notar que o algoritmo assim formulado retorna apenas 1 variável para o controlo. Porém, trabalha-se com 2 motores. Portanto, sua aplicação exige a seguinte abordagem: o sinal de controlo de cada motor, $u_{l,r}$, deverá ser atualizado incrementalmente e de maneira simétrica $u_l \rightarrow u_l + 0.5u$, $u_r \rightarrow u_r - 0.5u$

Além disso, certos cuidados foram tomados em relação a contribuição integral. Em particular, tentou-se evitar o fenómeno conhecido como *integral windup*, no qual o termo integral cresce demasiado e torna-se dominante, levando a longos transientes na resposta do sistema. Fez-se isso de dois modos:

1. Impor limites ao valor da componente I, i.e $|I| \leq I_{max}$
2. Adicionar o termo integral apenas se $|e| \leq e_{max}$

Isso é importante considerando que, em geral, procura-se observar a resposta do sistema a partir de um estado inicial com e elevado. Assim, caso o erro fosse integrado desde o instante inicial, ter-se-ia um acúmulo considerável do termo I de imediato, implicando nos longos transientes referidos acima. Por outro lado, como seu papel é fundamentalmente corrigir desvios em relação ao ponto de operação, sua ação pode ser restringida aos valores de e pequenos.

Outro ponto sobre a formulação do algoritmo é que esta permite facilmente escolher outro ponto de operação que não $e = 0$, permitindo o equilíbrio em torno de um ângulo arbitrário.

Referências: [10, 11]

4 Integração do sistema

Uma vez disponibilizada a estrutura mecânica, efetuou-se a montagem dos motores e iniciaram-se esforços para integrar os subsistemas acima descritos. A estrutura do algoritmo encontra-se esquematizada em 7

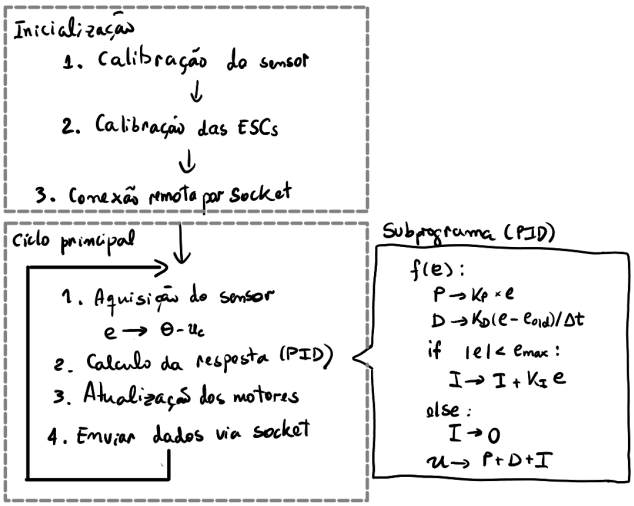


Figura 7: Fluxograma da rotina implementada para controle do *drone*

4.1 Danificação da ESP32

O maior problema enfrentado foi a danificação da placa devido a passagem de uma corrente demasiado elevada nesta. Isto aconteceu devido a dois problemas na montagem e dimensionamento do circuito. Em primeiro lugar, a referência da placa foi conectada à referência da fonte de alimentação dos motores. Além disso, utilizou-se a mesma fonte para alimentar os dois motores, bem como os amplificadores operacionais 5. Tal como referido antes, cada motor deveria consumir entre $1 - 1.5\,A$ no regime de operação fixado, enquanto que a fonte utilizada fornecia no máximo $1.5\,A$. Portanto, ao conectar os dois motores ao mesmo tempo, demasiada corrente foi puxada - facto que se comprovou pelo *display* da fonte, que acusou um curto-circuito. Como agravante, o circuito de alimentação do motor estava ligado ao circuito da placa através do comum, o que permitiu corrente excessiva passar por esta.

Após adquirir uma nova placa, fez-se também três mudanças fundamentais ao circuito. Em primeiro lugar, introduziu-se uma segunda fonte de alimentação, de modo a ter os motores alimentados de forma independente. Além disso, separou-se os comuns da placa e das fontes, removendo a conexão física entre os dois. Isso permitiu isolar a montagem em dois sub-circuitos: um com corrente na ordem das dezenas de mA, e outro com a corrente de alimentação.

Por fim, optou-se por uma simplificação na conversão de nível lógico. Os amplificadores foram substituídos por um circuito integrado dedicado a conversão bidirecional entre sinais digitais em $3.3\,V$ e $5\,V$, ver 8. Assim, a montagem tornou-se mais económica em termos de espaço utilizado e por dispensar a alimentação externa dos amplificadores. Por outro lado, foi necessário introduzir uma fonte para estabelecer a referência dos $5\,V$. A maior vantagem da utilização do integrado é o facto deste permitir converter em simultâneo até 4 sinais. Neste

projeto, são necessários apenas dois canais (um para o PWM de cada motor), mas em perspectiva da adaptação do trabalho para um drone de 4 motores, esta solução mostrou-se a mais flexível.

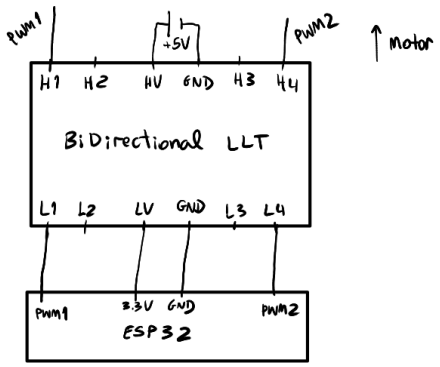


Figura 8: Esquema das conexões com o conversor de nível lógico bi-direcional (LLT - *Logic Level Translator*)

4.2 Incompatibilidade de *firmware*

Outro desafio enfrentado na fase de integração dos subsistemas esteve associado a incompatibilidades no *firmware* utilizado para a ESP32. Nomeadamente, a versão utilizada na primeira etapa era tal que corrigia um *bug* que impedia a geração de sinais de PWM com baixas frequências. Evidentemente, como é necessário um sinal de $50\,Hz$ para o controlo dos motores, esta situação era inadmissível.

Contudo, com a danificação da placa e aquisição de uma nova unidade, o *firmware* nesta instalado representava uma versão mais recente. Por se tratarem de versões *nightly* (de teste e mais instáveis), calhou do código para leitura do sensor deixar de funcionar. Como a versão anterior já não estava disponível, por erro em termos de gestão do projeto, foi preciso dedicar uma quantidade de tempo considerável para conciliar esta questão.

A solução encontrada foi utilizar uma versão mais antiga do *firmware*, que não contemplava nenhum dos *bugs* encontrados e que mostrou-se compatível com o restante do código escrito. Neste caso, o maior erro foi não ter tratado o *firmware* como parte integral do código escrito, ocasionando os problemas em controlo de versões acima descrito.

4.3 Dificuldades gerais para integração

Para além das dificuldades mencionadas, o processo de integrar os subsistemas em si revelou mais dificuldades do que o antecipado. Em particular, coordenar a inicialização dos diferentes módulos (comunicação Wi-Fi, calibração da ESP e calibração do sensor) foi um processo exigente. A solução encontrada foi

adaptar as rotinas de calibração de modo a excusar sua execução em todas as inicializações, além de tratar como opcional as funcionalidades Wi-Fi.

Estas questões tornaram evidentes as fragilidades desta filosofia *bottom-up* na organização do projeto. Claro que esta abordagem foi escolhida pelo facto de numa fase inicial não se dispor da estrutura mecânica para integração concreta dos subsistemas. Contudo, as dificuldades enfrentadas na segunda etapa deixaram claro que mais tempo deveria ter sido alocado ao estudo das interações entre os vários módulos desenvolvidos.

5 Resultados e conclusões

O circuito completo implementado está esquematizado abaixo.

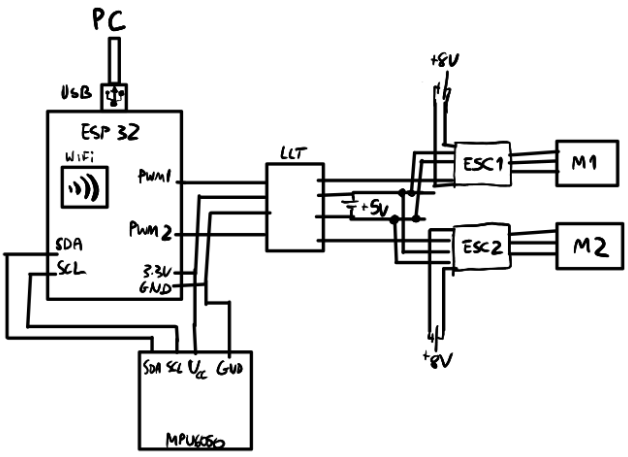


Figura 9: Circuito completo. M1 e M2 representam os dois motores utilizados. O chip que permite as funcionalidades Wi-Fi está contido na própria placa ESP32, que é alimentada via USB por um computador (PC). O conversor de nível lógico (LLT) é utilizado na comunicação com as ESCs.

O projeto foi bem sucedido e conseguiu-se fazer o equilíbrio automático da haste em torno de um ângulo arbitrário. Porém, apesar de ter sido desenvolvido um sistema para tal, não retirou-se dados quantitativos. Isto seria útil para avaliar o desempenho do algoritmo em termos de tempo de estabilização e *overshoot*.

Um ponto importante é o facto de existir muito atrito no eixo da haste, pelo que é preciso que os motores estejam acima de um certo nível de potência para que a haste se mova. Isto tem implicações para o tempo de estabilização, pois naturalmente exige que a resposta do motor cresça demasiado para vencer o atrito estático, mas volte a descer uma vez que o atrito dinâmico é menor. Este problema com o atrito é agravado pelo facto dos motores não estarem devidamente alinhados com a vertical. Assim, existe também um empuxo lateral

que contribui com o atrito sentido no eixo. Esta situação limita o *fine tuning* do algoritmo PID e deveria ser corrigida limando o furo no qual se insere o eixo, de modo a reduzir o atrito.

Nota: Repositório e código fonte

Todo o código desenvolvido, bem como o trabalho contínuo efetuado ao longo do semestre, está documentado neste repositório.

Referências

[1] Shane Colton, "The Balance Filter - A simple solution for integrating accelerometer and gyroscope measurements for a balancing platform", June 25 2007

[2] "MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2"

[3] "MPU-6000 and MPU-6050 Product Specification Revision 3.4"

[4] Autocalibração para MPU-6050

[5] adamjezek98, MPU6050-ESP8266-MicroPython - Simple library for MPU6050 on ESP8266 with micropython

[6] Mark Pedley, "Tilt sensing using a three-axis accelerometer", Mar. 2013

[7] Páginas da Atividade Docente e de Investigação - Hélio Sousa Mendonça - EDM 2021/22

[8] Electronoobs. (2017, 27 de maio). PID brushless motor control tutorial. Youtube.

[9] How To Mechatronics. (2019, 2 de fevereiro). How Brushless Motor and ESC Work and How To Control them using Arduino. Youtube.

[10] Karl J. Åström, T. H. (1995). PID Controllers (2 Sub). International Society for Measurement.

[11] Karl J. Åström, B. W. (1997). Computer-Controlled Systems - Theory and Design (3rd ed.). Prentice Hall.

[12] Exemplo comunicação TCP/IP em Python