

Universidade Federal de Catalão
IMTec – Engenharia Mecatrônica e Matemática Industrial

Gabriel Percival Stoppa
João Pedro Amaro Bennett da Silva

Construção de um Cubo de Auto Balanceamento

Catalão-GO

2023

Universidade Federal de Catalão
IMTec – Engenharia Mecatrônica e Matemática Industrial

Gabriel Percival Stoppa
João Pedro Amaro Bennett da Silva

Construção de um Cubo de Auto Balanceamento

Trabalho final da disciplina de Princípios de
Controle e Servomecanismos.

Professor: José dos Reis

Catalão-GO

2023

1. Introdução

Um Cubo de Auto Balanceamento é um dispositivo mecânico que utiliza um sistema de controle para manter sua posição vertical, sem cair, mesmo quando empurrado ou perturbado. Esse tipo de dispositivo é um exemplo de um sistema de controle de feedback, que é capaz de detectar desvios em relação a um objetivo desejado e ajustar suas ações para corrigir esses desvios.

O controle do Cubo de Auto Balanceamento é alcançado por meio de um controlador PID, que é uma técnica amplamente utilizada em sistemas de controle de feedback. PID significa Proporcional, Integral e Derivativo, e essas são as três ações básicas do controlador. Cada uma dessas ações é responsável por corrigir um tipo diferente de erro no sistema.

A ação proporcional é a mais simples e é baseada no erro atual do sistema em relação ao objetivo desejado. Ele age diretamente no erro e é proporcional a ele, ou seja, quanto maior o erro, maior a correção aplicada pelo controlador. A ação integral é responsável por corrigir erros acumulados ao longo do tempo, enquanto a ação derivativa é responsável por prever como o erro mudará no futuro.

O controlador PID utiliza essas três ações juntas para calcular a correção necessária para manter o sistema em equilíbrio. Ele mede a posição atual do cubo, calcula o erro em relação ao objetivo desejado e aplica uma correção adequada usando as ações proporcional, integral e derivativa.

O controlador PID ajusta continuamente a correção aplicada ao sistema, até que ele atinja o equilíbrio perfeito. Isso é alcançado por meio de ciclos de feedback contínuos, em que o controlador mede constantemente a posição do cubo, calcula o erro e ajusta sua correção.

Em resumo, um Cubo de Auto Balanceamento é um exemplo de um sistema de controle de feedback que utiliza um controlador PID para manter sua posição vertical. O controlador PID usa três ações básicas (proporcional, integral e derivativa) para calcular a correção necessária e ajustar continuamente o sistema até que ele atinja o equilíbrio perfeito.

2. Materiais Utilizados

Nessa seção, serão descritos os materiais utilizados na montagem do projeto. A lista completa dos componentes se encontra no final do trabalho.

2.1. Motores

Neste trabalho, utilizamos 3 motores Nidec 24, como na Fig. 1. HO motor CC Nidec 24H é um motor elétrico de corrente contínua (CC) fabricado pela empresa japonesa Nidec. Ele é projetado para fornecer um alto torque e eficiência energética em uma ampla variedade de aplicações, incluindo robótica, automação industrial, equipamentos médicos e outras aplicações que exigem movimento preciso e controle de velocidade.

O motor Nidec 24H tem uma tensão nominal de 24 volts DC e é capaz de operar em uma faixa de tensão de 12-28 volts DC. Ele tem uma corrente nominal de 0,25 amperes e pode operar em uma faixa de corrente

de 0,05 a 0,35 amperes. O motor tem uma potência nominal de 6 watts e uma velocidade nominal de 6000 RPM.

O motor Nidec 24H é um motor de escova, o que significa que ele usa escovas de carvão para transferir energia elétrica para o rotor. Ele tem um diâmetro de 24 mm e um comprimento de 30,2 mm. O eixo do motor tem um diâmetro de 2 mm e é projetado para ser acoplado a uma engrenagem ou outro dispositivo mecânico.

O motor Nidec 24H tem oito pinos, porém, utilizamos apenas cinco pinos: 8,7,6,5 e 4. O pino 8 é o positivo, o 7 é o negativo do motor (VCC e GND), o pino 6 controla a parada do motor (Brake), o pino 5 controla a velocidade do motor e o pino 4 a direção na qual o motor vai girar.



Figura 1. **Motor Nidec 24H.**

2.2. Acelerômetro

O acelerômetro utilizado no projeto é o MPU6050, como na Fig. 2. O MPU6050 é um sensor de movimento de 6 eixos fabricado pela empresa japonesa InvenSense (agora parte da TDK). Ele combina um acelerômetro de 3 eixos e um giroscópio de 3 eixos em um único chip, permitindo que ele meça a aceleração e a velocidade angular em várias direções.

O acelerômetro MPU6050 funciona medindo a força da gravidade em três eixos: X, Y e Z. Ele usa microestruturas de silício (MEMS) para detectar as mudanças na força gravitacional, que são então convertidas em sinais elétricos para processamento pelo microcontrolador.

O giroscópio MPU6050 funciona medindo a rotação em torno dos três eixos X, Y e Z. Ele usa um sensor de movimento chamado MEMS para detectar as mudanças na velocidade angular, que são convertidas em sinais elétricos para processamento pelo microcontrolador.

A tensão de alimentação do MPU6050 é de 3,3 volts, embora ele possa tolerar uma faixa de tensão de 2,375 a 3,46 volts, no projeto ele está

liga à uma tensão de 5 volts. Ele tem um consumo de corrente muito baixo, tornando-o adequado para aplicações de baixo consumo de energia.

O MPU6050 tem um total de 8 pinos, que incluem pinos de alimentação e terra, pinos de comunicação I2C e pinos de saída do sensor. Os pinos SDA e SCL são usados para comunicação I2C, enquanto os pinos AD0, INT e AUX são usados para configuração e leitura dos dados de saída do sensor.

O MPU6050 é amplamente utilizado em aplicações de controle de movimento, como controle de drones, robôs, jogos de realidade virtual e controle de dispositivos portáteis. Ele é facilmente integrado a microcontroladores e plataformas de desenvolvimento, como o Arduino e o Raspberry Pi, tornando-o uma escolha popular para prototipagem rápida de projetos eletrônicos.

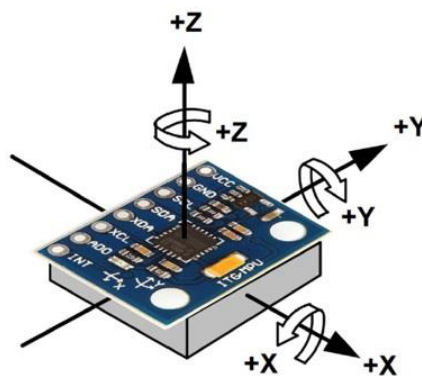


Figura 2. MPU6050.

Fonte: <https://blog.arduinoomega.com/5-passos-para-utilizar-o-sensor-de-inclinacao-giroscopio/>

2.3. Arduino

Para fazer o controle PID, controlar os motores e ler os dados do acelerômetro, utilizamos um Arduino Nano, como na Fig. 3. O Arduino Nano é uma placa microcontroladora baseada no chip ATmega328P fabricada pela empresa italiana Arduino. É uma das menores placas Arduino disponíveis e é frequentemente usada em projetos de eletrônica DIY (faça você mesmo) e prototipagem rápida.

O Arduino Nano possui entradas e saídas digitais e analógicas, além de interfaces de comunicação, como USB, UART e I2C. Ele é capaz de receber sinais de sensores, controlar atuadores e se comunicar com outros dispositivos eletrônicos.

A placa é programável usando a linguagem de programação baseada em C / C ++ e a IDE (ambiente de desenvolvimento integrado) do

Arduino, que é uma plataforma de desenvolvimento de software de código aberto e fácil de usar.

O Arduino Nano é amplamente utilizado em projetos de robótica, automação residencial, controle de motores, medição de sensores, controle de iluminação, entre outras aplicações. É uma ferramenta útil para prototipagem rápida de projetos eletrônicos e pode ser facilmente integrado a outros dispositivos e sistemas de controle.

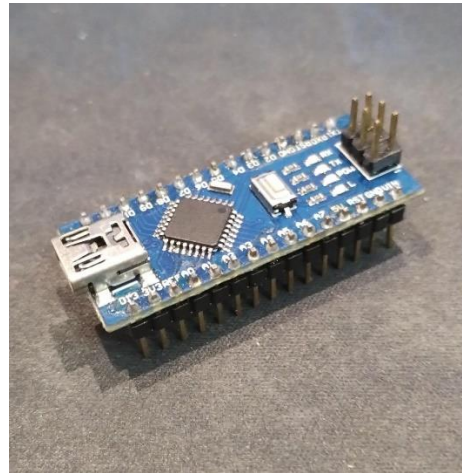


Figura 3. **Arduino Nano.**

2.4. Módulo Buzzer

Para indicar os sinais de aviso e calibração do cubo, utilizou-se um módulo buzzer para aviso sonoro. O módulo buzzer é um componente eletrônico utilizado para produzir sons em um circuito eletrônico, como na Fig. 4. Ele é composto por um pequeno alto-falante, que pode ser alimentado com uma tensão elétrica para gerar um som audível.

Existem diversos tipos de módulos buzzers disponíveis no mercado, mas em geral eles são alimentados por uma tensão de 5V ou 12V. A voltagem específica pode variar dependendo do modelo do módulo e da aplicação em que ele será utilizado.

O módulo buzzer possui três pinos de conexão: um pino positivo (+), um pino negativo (-) e um pino para o sinal. Para alimentar o módulo, é necessário conectar o pino positivo a uma fonte de alimentação elétrica e o pino negativo ao terra do circuito, e o sinal é passado pelo Arduino no terceiro pino por uma saída digital.



Figura 4. **Módulo Buzzer.**

Fonte: <https://tiggercomp.com.br/novaloja2/modulo-buzzer-passivo>

2.5. Peças Impressas em 3D

A parte estrutural e os discos de reação do cubo foram impressos em uma impressora 3D. A impressora utilizada foi a Creality Ender-3 e o filamento utilizado foi o PLA. Para fazer o fatiamento e gerar os arquivos de impressão, utilizamos o software Ultimaker Cura v.5.2.2. Todos os arquivos (.stl) para impressão se encontram no final do trabalho. Alguns dos parâmetros de impressão foram:

- Altura da camada: 0.3
- Quantidade de linhas da parede: 2 de 0.8mm
- Camadas Topo/Inferior: 4 de 1.2mm
- Ironing Habilitado
- Densidade de Preenchimento: 5%
- Temperatura da Extrusora: 200 °C
- Temperatura da Mesa Aquecida: 60 °C
- Velocidade de Impressão: 120 mm/s
- Velocidade da camada inicial: 25 mm/s
- Não há necessidade de gerar suportes
- Tipo de adesão à mesa: Brim
- Distância do Brim: 3mm
- Quantidade de linhas do Brim: 8

As peças utilizadas, junto com suas quantidades estão na seção de componentes no final do trabalho. Uma dica, na hora de retirar as peças da impressora, espere esfriar para retirar, pois retirando as peças muito quentes, elas podem empenar, afetando o desempenho do cubo na hora de se equilibrar.

3. Circuito

O circuito do cubo utiliza os seguintes materiais:

- 1 Arduino Nano
- 1 Protoboard
- 1 Giroscópio MPU6050
- 1 Módulo Buzzer
- 1 Jack para conexão de fonte
- Jumpers para conexão

A lista de peças detalhada está no final do trabalho. O circuito foi montado em uma protoboard, entretanto, para ter melhores resultados, recomenda-se que o circuito seja feito em uma placa de circuito impresso.

O circuito pode ser visto na Fig. 5, onde, na parte esquerda está ligado o jack da fonte no terminal de 2 portas, e os 3 motores nos terminais de 5 portas, onde: vermelho: Vcc, preto: GND, verde: break, laranja: PWM, amarelo: DIR, são os fios dos motores. Na parte da direita, no terminal de 7 pinos está ligado o giroscópio e o buzzer. As cores representam: vermelho: Vcc, preto: GND, branco: SCL (giroscópio), roxo: SDA (giroscópio), amarelo: sinal do buzzer. Na fase de testes, se não imprimir os valores do acelerômetro, ou imprimir os valores +1 ou -1, tente inverter as portas SCL e SDA.

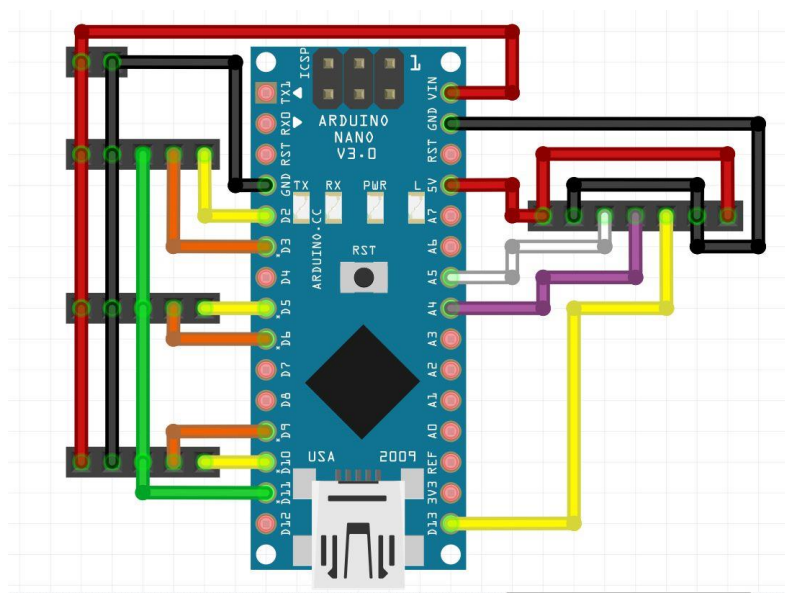


Figura 5. Circuito do Cubo.

4. Montagem

Comece a montagem retirando a proteção de metal que fica atrás dos motores, como na Fig. 6.



Figura 6. **Motores com e sem a proteção.**

Depois, com o disco de reação (peça 3) e os parafusos com porcas de 4mm, monte-os como na Fig.7 Utilize o mesmo padrão de posicionamento dos parafusos utilizado na figura. São utilizados 22 parafusos e porcas por disco.



Figura 7. **Disco de Reação.**

Para continuar com a montagem do disco, coloque 2 parafusos m3 e duas porcas no encaixe do eixo do motor no disco. Com 3 parafusos m3, junte o motor à peça 4, depois, coloque o disco de reação no eixo do motor e aperte os parafusos, como na Fig. 8. Certifique-se que o disco esteja bem alinhado no eixo, ou se os parafusos estão bem apertados, pois cada fator conta para dificultar o equilíbrio. Caso seja necessário, coloque um pedaço de fita ou borracha no eixo do motor para melhorar a aderência na peça.



Figura 8. **Disco de Reação ligado ao motor.**

Para finalizar a peça, use 4 parafusos e 8 porcas m3 para encaixar na peça 1. Coloque o parafuso na peça 4, depois rosqueie uma porca, depois, coloque as porcas na peça 1 e parafuse tudo, como na Fig. 9. Lembre-se de retirar a capa de plástico que fica atrás dos motores.

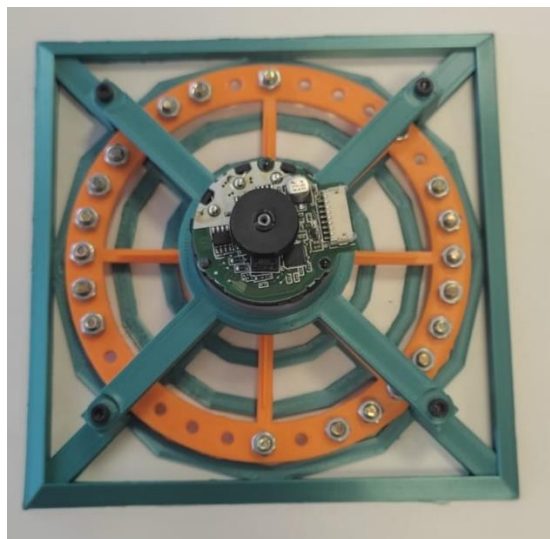


Figura 9. **Disco de Reação completo.**

Repita os passos até ter 3 faces montadas. Utilize 6 conectores para juntar as faces. Pronto, metade do cubo está montada. Para a segunda parte, parafuse o acelerômetro na peça 5, como na Fig.10.



Figura 10. Montagem do Acelerômetro.

Na mesma peça 5, do outro lado, parafuse 2 peças 7, 1 peça 8 e 3 peças 6. Depois, em cima das peças 7 e 8, parafuse a peça 9, como na Fig. 11 e na Fig. 12.



Figura 11. Montagem do suporte do circuito.



Figura 12. Montagem do suporte do circuito.

Com todo o suporte montado, encaixe a protoboard na peça de encaixe e parafuse-a no suporte. Com a protoboard, monte o circuito. Coloque o Arduino, conecte o sensor, conecte o buzzer, o jack da fonte e os fios dos motores, como na Fig. 13. O buzzer pode ser colocado na cavidade do suporte.

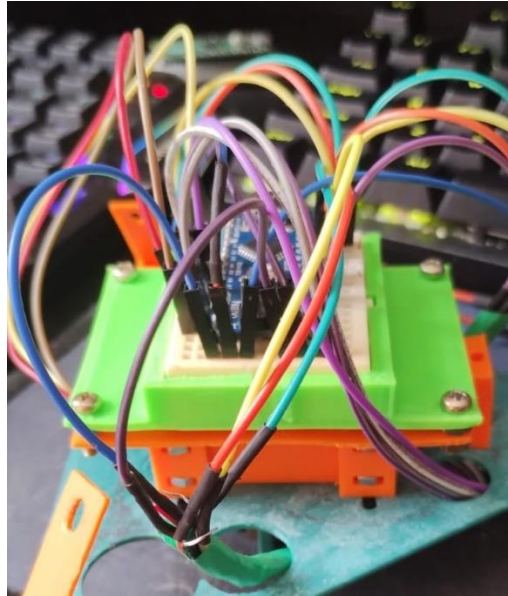


Figura 13. **Montagem do circuito.**

Utilizando 3 peças 2, junte-as com os conectores, como na Fig. 14. Utilize 6 conectores, 2 para cada aresta.

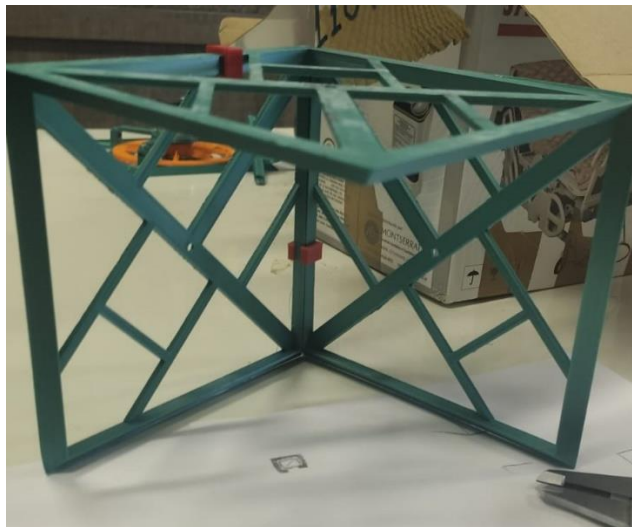


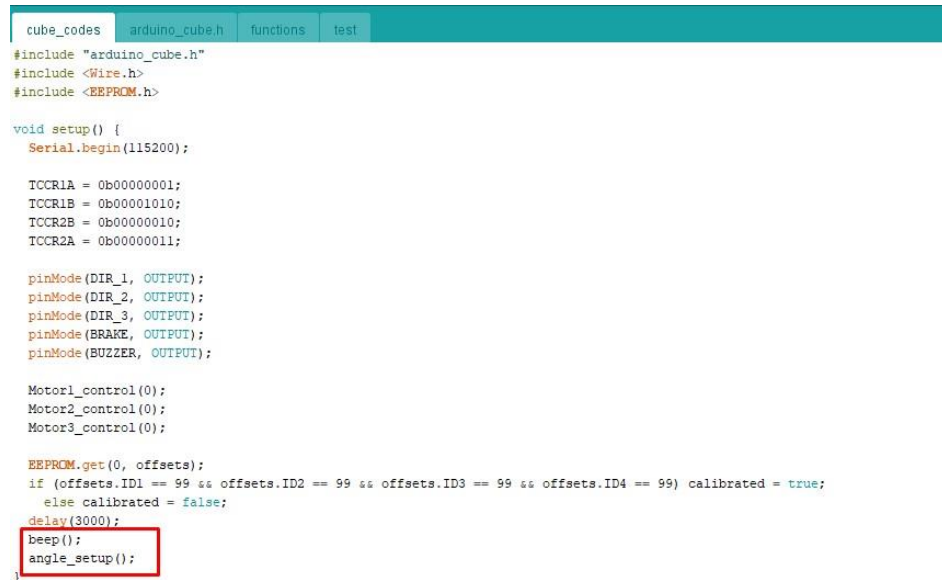
Figura 14. **Montagem da estrutura.**

Com a estrutura montada, junte-a com a peça do circuito utilizando 3 parafusos e porcas m3. Pronto, a segunda metade do cubo está pronta. Junte as duas metades utilizando os conectores e não se esqueça de ligar os fios dos motores.

5. Código

Com toda a parte estrutural e o circuito do cubo montados, agora é a hora de entender como o código funciona e fazer o upload para a placa do Arduino. Baixe os códigos no GitHub: <https://github.com/Gabrielstoppa18/UFCAT-Self->

[Balancing-Cube](#). O código possui 3 arquivos, `cube_codes.ino`, onde esse é o arquivo main, `arduino_cube.h`, onde está definido valores iniciais e as portas dos componentes, e `functions.ino`, onde as principais funções se encontram. No `cube_codes.ino`, na função `void setup`, parâmetros básicos estão definidos, depois é verificado se o cubo está calibrado, depois um sinal sonoro será emitido (bip) e a função `angle_setup` é chamada, como na Fig.15.



```

cube_codes  arduino_cube.h  functions  test

#include "arduino_cube.h"
#include <Wire.h>
#include <EEPROM.h>

void setup() {
  Serial.begin(115200);

  TCCR1A = 0b00000001;
  TCCR1B = 0b00001010;
  TCCR2B = 0b00000010;
  TCCR2A = 0b00000011;

  pinMode(DIR_1, OUTPUT);
  pinMode(DIR_2, OUTPUT);
  pinMode(DIR_3, OUTPUT);
  pinMode(BRAKE, OUTPUT);
  pinMode(BUZZER, OUTPUT);

  Motor1_control(0);
  Motor2_control(0);
  Motor3_control(0);

  EEPROM.get(0, offsets);
  if (offsets.ID1 == 99 && offsets.ID2 == 99 && offsets.ID3 == 99 && offsets.ID4 == 99) calibrated = true;
  else calibrated = false;
  delay(3000);
  beep();
  angle_setup();
}

```

Figura 15. Função `angle_setup`.

A função `angle_setup` acessa o acelerômetro e o calibra, mostrando no monitor serial os valores de GyZ e GyY, significando que os sensores estão calibrados. Após a calibração dos sensores, dois bips são emitidos e o código entra no loop principal. No loop principal, o código chama a função `Tuning`, que serve para calibrar os ângulos de equilíbrio do cubo e o PID, e a função `angle_calc`, que calcula os ângulos que o cubo se encontra, definindo em qual posição ele está, se está em uma das 3 arestas, se está no vértice ou se está na horizontal. Definindo em qual ponto cubo se encontra, verifica-se a diferença entre os ângulos e que o cubo está e os ângulos de equilíbrio, como na Fig. 16.


```

void loop() {

    currentT = 11;

    Tuning(); // derinimui
    angle_calc();

    if (balancing_point == 1) {
        angleX -= offsets.X1;
        angleY -= offsets.Y1;
        if (abs(angleX) > 8 || abs(angleY) > 8) vertical = false;
    } else if (balancing_point == 2) {
        angleX -= offsets.X2;
        angleY -= offsets.Y2;
        if (abs(angleY) > 6) vertical = false;
    } else if (balancing_point == 3) {
        angleX -= offsets.X3;
        angleY -= offsets.Y3;
        if (abs(angleY) > 6) vertical = false;
    } else if (balancing_point == 4) {
        angleX -= offsets.X4;
        angleY -= offsets.Y4;
        if (abs(angleX) > 6) vertical = false;
    }
}

```

Figura 16. Verificando os ângulos.

Com a diferença, erro do ângulo, os parâmetros de PID são acionados para determinar a velocidade em que os motores vão girar, como na Fig. 17.

```

,

if (vertical == calibrated == !calibrating) {
    digitalWrite(BRAKE, HIGH);
    gyroZ = GyZ / 131.0; // Convert to deg/s
    gyroY = GyY / 131.0; // Convert to deg/s

    gyroYfilt = alpha * gyroY + (1 - alpha) * gyroYfilt;
    gyroZfilt = alpha * gyroZ + (1 - alpha) * gyroZfilt;

    int pwm_X = constrain(pGain * angleX + iGain * gyroZfilt + sGain * motor_speed_pwmX, -255, 255); // LQR
    int pwm_Y = constrain(pGain * angleY + iGain * gyroYfilt + sGain * motor_speed_pwmY, -255, 255); // LQR
    motor_speed_pwmX += pwm_X;
    motor_speed_pwmY += pwm_Y;
}

```

Figura 17. Acionando o PID.

Após acionar o PID e girar os motores, o loop retorna para o início, verificando assim, se o erro diminuiu e o cubo está na posição desejada.

6. Ligando o cubo

Depois de entender o código, fazer o upload para a placa do Arduino, agora é hora de ligar a fonte e ver se tudo está funcionando bem (é bem provável que não). Ligue a fonte de 12V e 2A na tomada e conecte-a no jack do cubo, ligue o cabo do Arduino no computador e abra o monitor serial do Arduino e defina a velocidade do monitor serial para 115200. Aguarde o primeiro sinal sonoro, depois de alguns segundos o segundo sinal sonoro é ativado e o valor de GyZ aparece na tela, como na Fig 18.

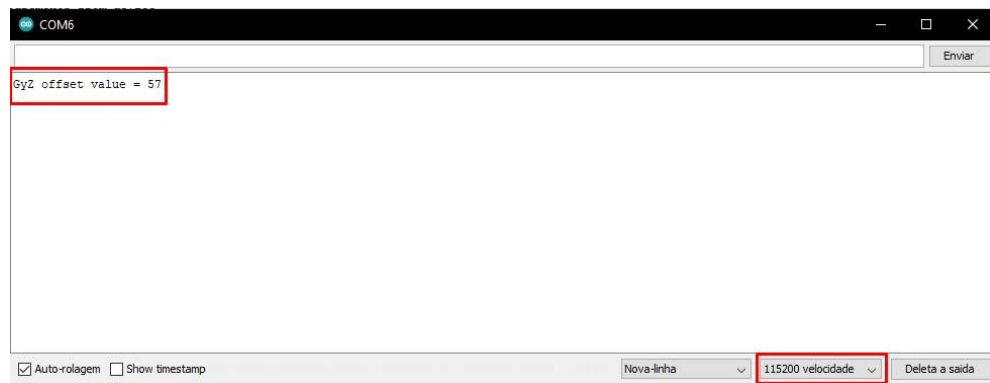


Figura 18. **Monitor serial.**

Depois de alguns segundos o valor de GyY aparece na tela e o cubo emite 2 bips. Com isso, os sensores estão calibrados, agora é hora de calibrar os ângulos de equilíbrio. No console do monitor serial digite c+ e a tecla enter para ligar o modo de calibração. Coloque o cubo em uma de suas arestas e equilibre-o na ponta do dedo, depois digite c- e a tecla enter para confirmar. Após a confirmação, deverá aparecer o número da aresta e se ela está calibrada, exemplo: “*First edge calibrated*”, dizendo que a primeira aresta foi calibrada, como na Fig. 19.

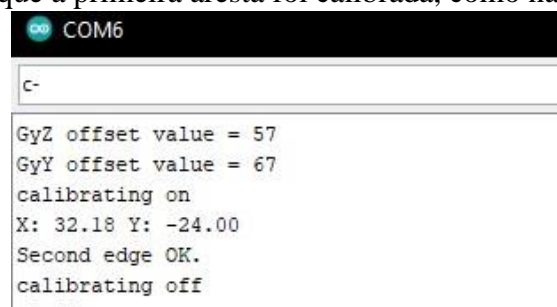


Figura 19. **Calibrando as arestas.**

Imediatamente o cubo faz um bip e o disco deve começar a girar. Repita o mesmo processo para todas as 3 arestas e depois para o vértice. Caso não gire ou gire o motor errado, verifique os fios dos motores, verifique se está conectado na tomada, verifique se está ligado corretamente no Arduino, estes são alguns erros que podem ocorrer.

Com as arestas calibradas, o cubo deve começar a se equilibrar, se não, agora chegou o momento de ajustar os parâmetros de PID. Ainda no monitor serial aperte p+, ou i+, ou s+ para calibrar cada um dos parâmetros, o valor deles será exibido na tela e você pode controlar utilizando + e -. Caso os valores não se estabilizem, vá no arquivo arduino_cube.h e procure os parâmetros de PID, como na Fig. 20.

```

cube_codes$ arduino_cube.h functions$ test

#define MPU6050 0x68 // Device address
#define ACCEL_CONFIG 0x1C // Accelerometer configuration address
#define GYRO_CONFIG 0x1B // Gyro configuration address
#define PWR_MGMT_1 0x6B
#define PWR_MGMT_2 0x6C

//Sensor output scaling
#define accSens 0 // 0 = 2g, 1 = 4g, 2 = 8g, 3 = 16g
#define gyroSens 1 // 0 = 250rad/s, 1 = 500rad/s, 2 = 1000rad/s, 3 = 2000rad/s

float Gyro_amount = 0.996;

bool vertical = false;
bool calibrating = false;
bool calibrated = false;

//auto balancing point = 0;

float pGain = 297;//250;//150;
float iGain = 45;//40.00;//15.00;
float sGain = 0.025;//0.025;//0.040;
int loop_time = 10;

```

Figura 20. Ajustando o PID dentro do código.

Coloque os valores de I e S em 0 e ajuste o valor de P até que o cubo tenha força suficiente nos motores e comece a equilibrar. O gráfico do ângulo por tempo do cubo com apenas o controlador P, pode ser visto na Fig. 21, onde o cubo apenas ficou equilibrado por pouco tempo e caiu, entretanto, o discos tinham força para equilibrá-lo.

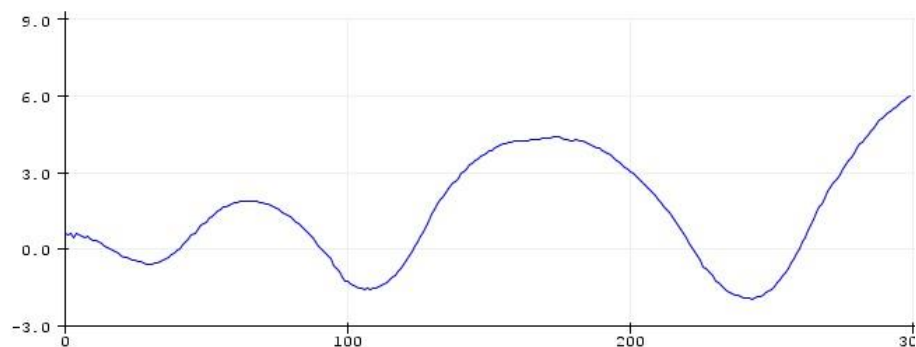


Figura 21. Gráfico com controle P.

Com o valor de P definido, aumente o valor de I até que se equilibre. Com o controle PI, o cubo permanece em equilíbrio por mais tempo, como na Fig.22.

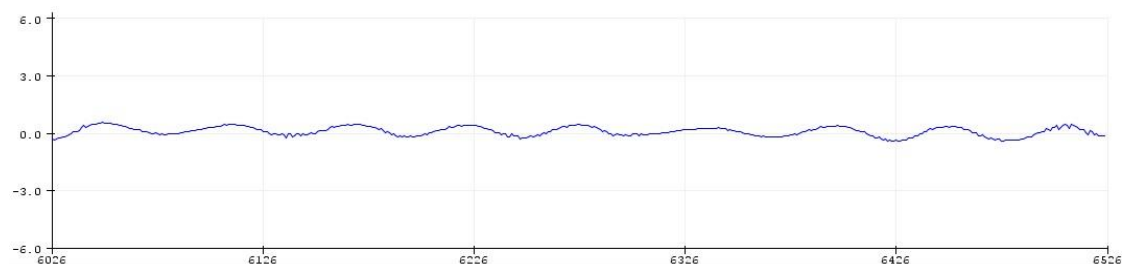


Figura 22. Gráfico com controle PI.

Por último, aumente o valor de S, que é o termo derivativo até que o cubo esteja perfeitamente estável. O gráfico do controle PID pode ser visto na Fig. 23.

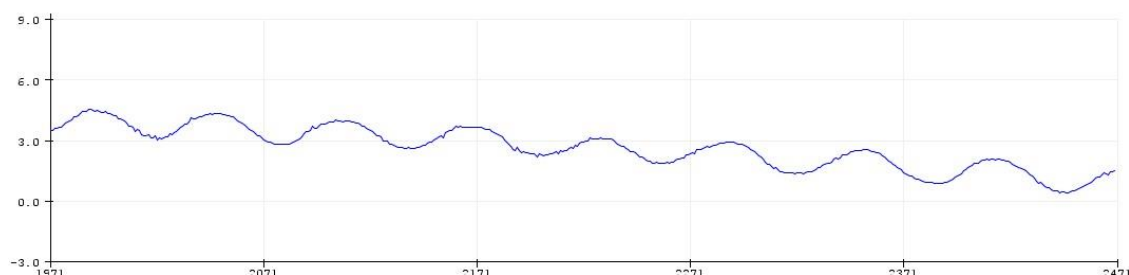


Figura 23. Gráfico com controle PID.

7. Conclusão

O cubo conseguiu se equilibrar com êxito em todos os testes, entretanto, alguns pontos devem ser ressaltados. A estrutura é frágil e algumas peças podem quebrar ou empenar, tornando muito difícil o equilíbrio pleno do cubo. Para trabalhos futuros, recomendamos que os discos de reação sejam cortados em MDF ou acrílico, o resultado pode ser melhor. Também, é recomendado que o circuito seja feito em uma PCB, a fim de eliminar a quantidade de fios. No geral a construção do projeto é simples, e ajustar os parâmetros do código leva um certo tempo, mas é um trabalho gratificante e que promove um aprendizado imenso em diversos fatores.

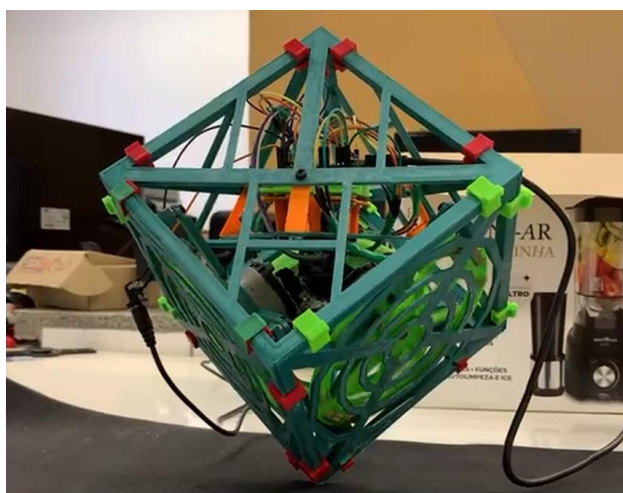


Figura 24. Cubo de Auto Balanceamento.

8. Tabelas de materiais

Componentes	Quantidade
Arduino nano	1
Fonte 12v 2A	1
Jack fêmea p/ fonte	1

Jumper macho modificado	20
Jumper macho/fêmea	10
Motor Nidec 24h	3
MPU6050	1
Protoboard 170 pontos	1

Tabela 1. Componentes eletrônicos.

Componentes	Quantidade
Parafuso m4	66
Porcas m4	66
Parafuso m3	50
Porca m3	50

Tabela 2. Componentes Mecânicos.

Componentes	Quantidade
Face externa para fixar motores (Peça 1)	3
Face externa (Peça 2)	3
Disco de Reação (Peça 3)	3
Suporte do motor (Peça 4)	3
Placa de suporte do circuito (Peça 9)	1
Suporte para protoboard	1
Suporte geral do circuito (Peça 5)	1
Presilha para fixar as faces	24
Suporte para prender circuito nas faces (Peça 6)	3
Suporte para placa do circuito 1 (Peça 7)	2
Suporte para placa do circuito 2 (Peça 8)	1

Tabela 3. Peças Impressas em 3D.

Link para o GitHub com todos os arquivos:

<https://github.com/Gabrielstoppa18/UFCAT-Self-Balancing-Cube>