Instituto Federal do Espírito Santo
Campus Serra
Disciplina de Sistemas Microcontrolados

Relatório de Projeto: Robô Equilibrista

Aluno: Matheus Corteletti Delfino
Professor: Leonardo Azevedo Scardua

Julho
2024

Conteúdo

1 Introdução

Este relatório descreve o desenvolvimento e a implementação de um robô equilibrista. O projeto envolve tanto a construção do hardware quanto a programação necessária para que o robô se equilibre de forma autônoma utilizando um controlador PID e um sensor MPU6050.

2 Hardware

2.1 Componentes Utilizados

- Placa Arduino Uno: A placa Arduino Uno foi escolhida para este projeto devido ao seu clock de 16 MHz, que oferece um bom equilíbrio entre desempenho e simplicidade de uso. Além disso, a placa fornece quatro portas PWM, essenciais para o controle preciso dos motores do robô equilibrista.

- Sensor MPU6050 (acelerômetro e giroscópio): O sensor MPU6050 foi uma peça fundamental no projeto, oferecendo seis graus de liberdade (3 eixos de aceleração e 3 eixos de rotação). Ele foi posicionado no centro de gravidade do chassi, utilizando como base as proporções de outros projetos semelhantes. As leituras específicas utilizadas foram:

```
uint8_t i2c_data[14];
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
```

  As medições incluem os três eixos de aceleração (accX, accY, accZ) e os três eixos de rotação (gyroX, gyroY, gyroZ), conhecidos como pitch, roll e yaw.

- Motores DC com driver L298N: Foram utilizados motores DC comuns de até 6V, que, embora simples, são adequados para as necessidades do projeto. O driver L298N facilitou o controle desses motores, permitindo o uso de saídas analógicas e PWM para uma gestão eficiente e precisa da velocidade e direção dos motores.

- Estrutura do robô (chassi, rodas, suportes): A estrutura do robô foi composta por um chassi leve, rodas e suportes. O material do chassi foi escolhido pela sua leveza e resistência, permitindo um fácil manuseio

e montagem. As rodas e suportes foram selecionados para proporcionar estabilidade e mobilidade ao robô.

- Bateria e componentes de alimentação: O robô é alimentado por uma bateria de 1300 mAh e 11V, que fornece energia suficiente para todos os componentes. A alimentação foi distribuída de forma a garantir independência entre a placa Arduino e os motores. Um switch foi incluído para facilitar o controle de energia, e a bateria pode ser desacoplada para carregamento.

2.2 Montagem do Hardware

A montagem do robô equilibrista envolve várias etapas importantes para garantir a estabilidade e o funcionamento correto do sistema.

O chassi do robô é composto por placas de acrílico, reaproveitadas de um projeto descrito em Eletrogate. O corte das placas não ficou preciso, o que dificultou o setup das variáveis de controle, como será comentado posteriormente. Para futuras implementações, recomenda-se um cuidado especial com o corte das placas; se possível, utilizar um chassi cortado a laser para maior precisão e facilidade na montagem.

Os motores DC foram fixados ao chassi utilizando solda nos terminais e conexões, garantindo um contato elétrico seguro e durável. Além disso, cola quente foi utilizada para fixar os dispositivos no chassi, proporcionando uma montagem rápida e eficiente.

Como comentado, o sensor MPU6050 foi instalado na posição correta, próximo ao centro de gravidade do robô, para medir com precisão a inclinação e os movimentos do robô. Ele foi posicionado de forma que o eixo X ficasse perpendicular a face do robô, portanto foi montade para baixo.

3 Software

3.1 Configuração Inicial

O código do projeto foi adaptado de um código fonte existente, com ajustes para o novo hardware e configurações específicas do robô equilibrista. Abaixo está a configuração inicial dos pinos e a inclusão das bibliotecas necessárias.

```
#include <Wire.h>
#include <Kalman.h>
```

```
    uint8_t i2c_data[14];
    double accX, accY, accZ;
    double gyroX, gyroY, gyroZ;

    uint32_t timer;

    Kalman KalmanX;
    Kalman KalmanY;
    Kalman KalmanZ;

    double KalAngleX;
    double KalAngleY;
    double KalAngleZ;

    double gyroXangle;
    double gyroYangle;

    void setup() {

      /* Serial para exibir mensagens de Debug */
      Serial.begin(115200);

      /* Barramento i2c para comunicação com a MPU6050 */
      Wire.begin();

    #if ARDUINO >= 157
      Wire.setClock(400000UL); // Freq = 400kHz.
    #else
      TWBR = ((F_CPU/400000UL) - 16) / 2; // Freq = 400kHz
    #endif

/*Taxa de amostragem  8kHz/(7 + 1) = 1000Hz */
      i2c_data[0] = 7;

/*Desabilitar FSYNC, Configurar o Filtro de ACC 260Hz,
Configurar Filtro de Gyro 256Hz, Amostragem de 8Khz */
      i2c_data[1] = 0x00;

/*Configurar o fundo de escala do Gyro
±250deg/s - Faixa */
```

```
      i2c_data[2] = 0x00;

/*Configurar o fundo de escala do Acelerômetro para
±2g - Faixa */
      i2c_data[3] = 0x00;

      /* Configirações do i2c*/
      while(i2cWrite(0x19, i2c_data, 4, false));

      /* PLL tenha como referência o gyro de eixo X,
      Desabilitando Sleep Mode */
      while(i2cWrite(0x6B, 0x01, true));

      /* */
      while(i2cRead(0x75, i2c_data, 1));

      if(i2c_data[0] != 0x68){
        Serial.print("Erro. Placa desconhecida\n");
        while(1){
          Serial.print("Conectar MPU no barramento i2c\n");
        }
      }

      /* Tempo de estabilização do Sensor MPU6050 */
      delay(100);

      /* 1 - Leitura dos dados de Acc XYZ */
      while(i2cRead(0x3B, i2c_data, 14));

      /* 2 - Organizar os dados de Acc XYZ */
      accX = (int16_t)((i2c_data[0] << 8) | i2c_data[1]);
      accZ = (int16_t)((i2c_data[4] << 8) | i2c_data[5]);

      /* 3 - Calculo de Pitch e Roll */
      double pitch = atan(accX/sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
      double roll = atan(accY/sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;

      /* 4 - Inicialização do Filtro de Kalman XY */
      KalmanX.setAngle(roll);
      KalmanY.setAngle(pitch);
```

```
    gyroXangle = roll;
    gyroYangle = pitch;

    timer = micros();

  //  Serial.print("Fim Setup\n");

    init_motores();
  }
```

O projeto é composto por três módulos principais que são utilizados no loop principal: Motores.ino, PID.ino e i2cMPU6050.ino.

3.2 Motores.ino

Este módulo contém toda a lógica de controle dos motores. Ele utiliza as portas PWM do Arduino para controlar a velocidade e direção dos motores DC, garantindo a estabilidade e mobilidade do robô. A configuração do driver L298N é realizada neste módulo, permitindo o controle preciso dos motores.

3.3 PID.ino

Este módulo implementa o controlador PID que ajusta a velocidade dos motores com base nos dados de inclinação fornecidos pelo MPU6050. O controlador PID utiliza os parâmetros Kp, Ki e Kd para calcular a saída necessária para manter o robô equilibrado. A configuração inicial dos parâmetros e o cálculo da saída do PID são realizados neste módulo.

3.4 i2cMPU6050.ino

Este módulo lida com a comunicação I2C entre o Arduino e o sensor MPU6050. Ele realiza a leitura dos dados brutos do acelerômetro e giroscópio, e aplica o filtro de Kalman para estimar os ângulos de inclinação (pitch e roll) com precisão. As funções de inicialização e leitura dos dados do sensor, bem como a aplicação do filtro de Kalman, estão todas contidas neste módulo.

3.5 Integrando o uso da MPU6050 com I2C

A integração do sensor MPU6050 via I2C permite a leitura dos dados do acelerômetro e giroscópio. A comunicação I2C é configurada e inicializada,

garantindo a coleta correta dos dados do sensor.

O protocolo I2C (Inter-Integrated Circuit) é um padrão de comunicação serial que permite a conexão de múltiplos dispositivos em um barramento de dois fios, geralmente chamados de SDA (Serial Data Line) e SCL (Serial Clock Line). É amplamente utilizado devido à sua simplicidade e eficiência em interconectar microcontroladores e periféricos, como sensores e atuadores.

minted fontspec Consolas

```
/* Tempo de estabilização do Sensor MPU6050 */
delay(100);

/* 1 - Leitura dos dados de Acc XYZ */
while(i2cRead(0x3B, i2c_data, 14));

/* 2 - Organizar os dados de Acc XYZ */
accX = (int16_t)((i2c_data[0] << 8) | i2c_data[1]); // ([ MSB ] [ LSB ])
accY = (int16_t)((i2c_data[2] << 8) | i2c_data[3]); // ([ MSB ] [ LSB ])
accZ = (int16_t)((i2c_data[4] << 8) | i2c_data[5]); // ([ MSB ] [ LSB ])
```

No código acima, ocorrem as seguintes etapas:

1. **Tempo de estabilização do Sensor MPU6050**: Após a inicialização, é necessário um tempo para que o sensor se estabilize e forneça leituras precisas. O delay de 100 milissegundos garante essa estabilização.

2. **Leitura dos dados de Acc XYZ**: A função `i2cRead` é utilizada para ler 14 bytes de dados do registro do sensor MPU6050. Esses dados incluem as leituras dos acelerômetros e giroscópios nos três eixos (X, Y, Z).

3. **Organização dos dados de Acc XYZ**: Os dados brutos lidos do sensor são organizados e convertidos em valores significativos de aceleração. Os dados são combinados a partir dos bytes mais significativos (MSB) e menos significativos (LSB) para formar os valores de aceleração nos eixos X, Y e Z.

3.6 i2cMPU6050.ino

O módulo ‘i2cMPU6050.ino‘ é fundamental para a comunicação entre o Arduino e o sensor MPU6050. Ele abstrai a complexidade do protocolo I2C, fornecendo funções que facilitam a leitura e escrita de dados no sensor. Abaixo, destacamos as principais funções e sua importância:

minted fontspec Consolas

```
// AD0 is logic low on the PCB
const uint8_t IMUAddress = 0x68;
// Used to check for errors in I2C communication
const uint16_t I2C_TIMEOUT = 1000;

// Returns 0 on success
uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop) {
  return i2cWrite(registerAddress, &data, 1, sendStop);
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop) {
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  Wire.write(data, length);
  uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on success
  if (rcode) {
    Serial.print(F("i2cWrite failed: "));
    Serial.println(rcode);
  }
  return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
}

uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {
  uint32_t timeOutTimer;
  Wire.beginTransmission(IMUAddress);
  Wire.write(registerAddress);
  uint8_t rcode = Wire.endTransmission(false); // Don't release the bus
  if (rcode) {
    Serial.print(F("i2cRead failed: "));
    Serial.println(rcode);
    return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
  }
  // Send a repeated start and then release the bus after reading
  Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true);

  for (uint8_t i = 0; i < nbytes; i++) {
    if (Wire.available())
      data[i] = Wire.read();
    else {
```

```
        timeOutTimer = micros();
        while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.available());
        if (Wire.available())
          data[i] = Wire.read();
        else {
          Serial.println(F("i2cRead timeout"));
          return 5; // This error value is not already taken by endTransmission
        }
      }
    }
    return 0; // Success
}
```

3.7 Estimando Ângulos com Filtro de Kalman

O filtro de Kalman é utilizado para estimar com precisão os ângulos de inclinação do robô, combinando as leituras do acelerômetro e do giroscópio para fornecer uma medida suave e confiável. A imagem abaixo mostra um gráfico que ilustra como o filtro de Kalman (linha vermelha) reduz o ruído das medições (linha azul).

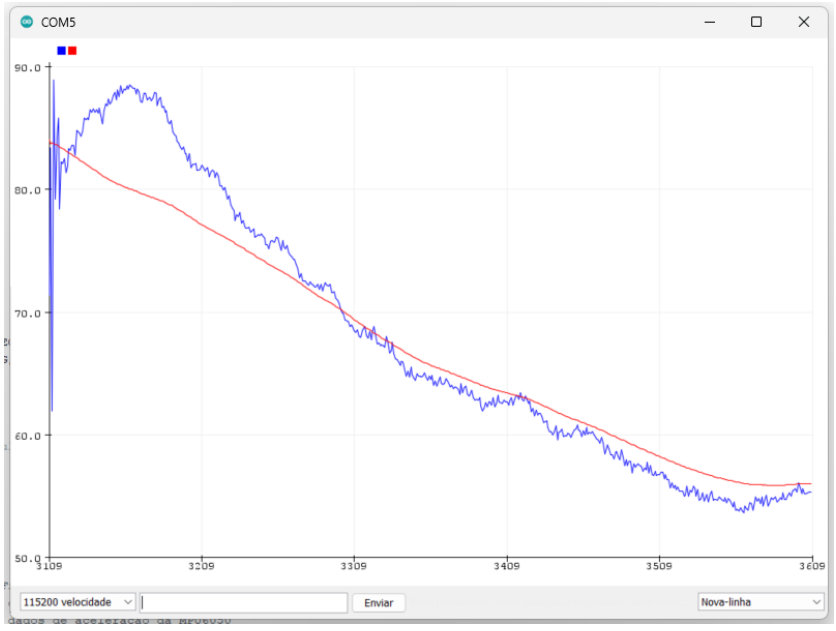

Figura 1: Efeito do Filtro de Kalman na Redução de Ruído

O filtro de Kalman é um algoritmo matemático que utiliza uma série de medições observadas ao longo do tempo, contendo ruído (ou outras incertezas), e produz estimativas de parâmetros desconhecidos que tendem a ser mais precisas do que aquelas baseadas em uma única medição. Ele é fundamental para viabilizar o controle do robô, pois reduz significativamente o ruído das leituras dos sensores.

A biblioteca 'Kalman Filter Library v1.0.2', implementada por 'Kristian Lauszus' fornece a implementação do filtro de Kalman para facilitar a integração no projeto. O filtro utiliza os dados de aceleração e rotação para calcular uma estimativa mais precisa da inclinação do robô.

Aqui está o código detalhado que utiliza o filtro de Kalman:

```
/****************** Filtro de Kalman ************************/

/* Calculo do Delta Time */
double dt = (double)(micros() - timer)/1000000;
timer = micros();

double pitch = atan(accX/sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
double roll = atan(accY/sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;

/* Convertendo de Rad/Segundo para Graus/Segundo Calculo da Taxa angular baseado no Giro
gyroXangle = gyroX / 131.0; //deg/s
gyroYangle = gyroY / 131.0; //deg/s

/* Estimativa de Ângulo nos Eixos X e Y usando Filtro de Kalman */
KalAngleX = KalmanX.getAngle(roll, gyroXangle, dt);
KalAngleY = KalmanY.getAngle(pitch, gyroYangle, dt);
```

Neste código, ocorre o seguinte:

1. **Calculo do Delta Time**: - O delta time (`dt`) é calculado para determinar o intervalo de tempo entre as leituras, o que é crucial para o cálculo do filtro de Kalman.

2. **Cálculo de Pitch e Roll**: - Os ângulos de pitch e roll são calculados a partir das leituras do acelerômetro utilizando a função arctangente.

3. **Conversão da Taxa Angular do Giroscópio**: - As leituras do giroscópio são convertidas de radianos por segundo para graus por segundo, o que facilita a interpretação dos dados.

4. **Estimativa de Ângulo com Filtro de Kalman**: - O filtro de Kalman é aplicado para obter estimativas mais precisas dos ângulos de roll

e pitch, combinando os dados do acelerômetro e giroscópio.

3.8 Controle da Posição com PID e PWM

O controlador PID ajusta a velocidade dos motores com base nos ângulos estimados pelo filtro de Kalman, permitindo que o robô mantenha seu equilíbrio.

Foi utilizado um PID simples para controlar a posição do robô. O código abaixo mostra a implementação do controlador PID e o ajuste da velocidade dos motores usando PWM:

```
double kp = 18;
double kd = 2.5;
double ki = 0.1;

int OUTMAX = 200;
int OUTMIN = -200;

double SetPoint = 10;

float lastInput = 0.0;

double ITerm = 0.0;

double Compute(double input){

  /* Calculo do Erro */
  double erro = SetPoint - input;

  ITerm += (ki * erro);

  if(ITerm > OUTMAX){
    ITerm = OUTMAX;
  } else if(ITerm <= OUTMIN){
    ITerm = OUTMIN;
  }

  /* Diferença da entrada atual - anterior */
  double dInput = input - lastInput;

  /* Calculo do PID */
  double output = kp * erro + ITerm + kd * dInput;
```

```
    if(output > OUTMAX){
      output = OUTMAX;
    } else if(output <= OUTMIN){
      output = OUTMIN;
    }

    lastInput = input;

    return output;
  }
```

Explicação do Código:

1. **Parâmetros do PID**: - **kp**: Ganho proporcional, ajusta a força da resposta proporcional ao erro. - **kd**: Ganho derivativo, ajusta a força da resposta proporcional à taxa de mudança do erro. - **ki**: Ganho integrativo, ajusta a força da resposta proporcional à integral do erro ao longo do tempo.

2. **Limites de Saída**: - **OUTMAX** e **OUTMIN** definem os limites máximos e mínimos da saída do controlador para evitar valores extremos.

3. **SetPoint**: - **SetPoint** é o valor desejado para a posição do robô.

4. **Variáveis de Estado**: - **lastInput**: Armazena a última leitura do sensor para calcular a diferença de entrada. - **ITerm**: Armazena o termo integrativo do PID.

5. **Função Compute**: - **Cálculo do Erro**: Calcula a diferença entre o **SetPoint** e a entrada atual (**input**). - **Termo Integrativo**: Atualiza o termo integrativo e aplica limites. - **Diferença de Entrada**: Calcula a diferença entre a entrada atual e a última entrada. - **Cálculo do PID**: Combina os termos proporcional, integrativo e derivativo para calcular a saída. - **Limites de Saída**: Aplica limites à saída do PID. - **Atualização do Último Input**: Atualiza **lastInput** com a entrada atual. - **Retorno da Saída**: Retorna o valor calculado pelo PID para ajustar a velocidade dos motores.

4 Conclusão

Este projeto demonstrou a implementação de um robô equilibrista utilizando um controlador PID para manter o equilíbrio. A combinação de hardware bem configurado e software ajustado permite que o robô responda rapidamente às mudanças de inclinação e se mantenha estável.

A implementação do projeto foi significativamente simplificada graças ao uso do protocolo I2C e às interfaces proporcionadas pelo Arduino. O I2C facilitou a comunicação entre o Arduino e o sensor MPU6050, permitindo a integração eficiente dos dados do acelerômetro e giroscópio. As interfaces do Arduino proporcionaram uma plataforma robusta e fácil de usar para o desenvolvimento e a depuração do projeto.

Além disso, a possibilidade de reaproveitar bibliotecas já desenvolvidas difundiu o acesso a algoritmos relativamente complexos. Bibliotecas como 'Wire.h' para comunicação I2C e 'Kalman.h' para filtragem de dados permitiram que algoritmos avançados fossem implementados de maneira simples e eficaz, reduzindo a complexidade do desenvolvimento e acelerando o tempo de implementação.

O controlador PID, apesar de ser um algoritmo simples, serviu brilhantemente ao propósito de controle do robô equilibrista. Com apenas quatro parâmetros (kp, ki, kd, SetPoint), o PID conseguiu ajustar dinamicamente a velocidade dos motores, mantendo o robô equilibrado de maneira eficaz. Este projeto demonstra como algoritmos clássicos de controle, quando bem aplicados, podem fornecer soluções robustas e eficientes para problemas de engenharia.

5 Referências

Referências

Playlist do Projeto Robô Equilibrista. Disponível em: ⟨https://www.youtube.com/playlist?list=PLOta4d-65kVcj91XHAb_tBKsP6DpfpdRs⟩. Acesso em: 29 jun. 2024.