UNIVERSIDADE DE SÃO PAULO
Escola Politécnica

PCS 3732 – Laboratório de Processadores

Uma Biblioteca HAL em C++ e sua aplicação em um robô seguidor de linha

Prof. Bruno Basseto

Gabriel Cosme Barbosa **N° USP: 11844051**
Pedro Henrique Martins de Santi **N° USP: 11803483**

SÃO PAULO, 15/08/2023

1. INTRODUÇÃO

A utilização de sistemas embarcados sem um sistema operacional implica em uma grande barreira de entrada para desenvolvedores, visto que cada plataforma possui suas próprias características e configurações necessárias para o bom funcionamento do programa. A fim de solucionar este problema, são criadas camadas de abstração de hardware (HAL) que implementam uma interface para se utilizar funcionalidades do microcontrolador de forma mais fácil e intuitiva, ou seja, ao invés de ser necessário consultar um datasheet para se obter os endereços dos registradores utilizados para configurar e ler um GPIO, a HAL cria uma classe referente a essa funcionalidade, bastando apenas utilizar os métodos dessa classe.

Além disso, escolhemos a linguagem C++ para a implementação da biblioteca, que é pouco usada no contexto de embarcados, e desenvolvemos um algoritmo para um robô seguidor de linha, de forma que também foram criadas classes para utilizar sensores e atuadores, chamadas de proxys.

2. DESENVOLVIMENTO

A biblioteca foi desenvolvida para o microcontrolador STM32F411CE, que possui um processador ARM Cortex-M4 de 32 bits, 512Kb de memória flash, 128Kb de memória RAM e um clock máximo de 100MHz. A implementação foi dividida em duas camadas, sendo a camada de baixo nível responsável por definir as classes que acessam os registradores do microcontrolador, e a camada de alto nível responsável por implementar classes que utilizam os métodos da camada de baixo nível para realizar tarefas mais complexas.

2.1 HAL

As classes implementadas para a biblioteca HAL são acompanhadas por structs de configuração da classe, que são passadas para o construtor e definem tudo o que precisa ser configurado para o bom funcionamento da aplicação.

Além disso, utilizamos a biblioteca libopencm3 para facilitar a interação com o hardware, visto que essa biblioteca possui drivers para diversos dispositivos que utilizam o ARM Cortex-M, inclusive os da família STM32F4, que foi utilizado em nosso projeto, reduzindo a necessidade de consultar o datasheet para se obter o endereço dos registradores, por exemplo.

2.1.1 GPIO

A primeira classe implementada foi o GPIO, que também já havia sido utilizado ao longo do curso. Esta classe representa um pino do microcontrolador, e sua struct de configuração tem os seguintes campos:

- port: A porta a qual o GPIO pertence;
- pin: O pino do GPIO;
- mode: O modo de utilização do GPIO (input, output, analog);
- pull_resistor: A configuração do resistor ligado ao GPIO (nennhum, pullup, pulldown);
- rcc_clock: O bit que ativa o clock nesse periférico.

Também existem configurações extras, exclusivas para pinos de output:

- otype: O tipo de driver no pino de output (push-pull, open drain);
- speed: A velocidade máxima de chaveamento do sinal de saída;
- alt_func_num: configuração específica para se utilizar um pino no modo alternativo, representa o número da função desejada como consta no datasheet.

Além disso, a classe possui métodos para realizar a leitura digital do pino e escrever ou alternar o seu sinal de saída.

2.1.2 Timer

A classe de timer é baseada na pré-configuração dos system ticks, que são interrupções que ocorrem em intervalos definidos, a cada 1 milissegundo no nosso caso, de forma que através da contagem desses ticks, é possível medir a passagem

de tempo em um intervalo, ou parar a execução do programa durante um período. Essa classe não tem uma struct de configuração, visto que tanto a configuração dos systick, quando a do clock, é feita na classe Clock, que tem os seguintes valores a serem configurados:

- clock_scale: O endereço de uma struct que contém todas as configurações necessárias para possibilitar um determinado valor de clock, como as frequências de diversos periféricos, prescalers e etc;
- reload: Quantos clock devem ocorrer para a contagem de um system tick, para um período de 1ms, deve se utilizar a frequência do clock dividida por 1000;
- clock_source: A fonte do clock a ser utilizada, no nosso caso, as opções são o clock original ou o mesmo dividido por 8.

2.1.3 PWM

A classe de PWM também é baseada na utilização de um timer, mas de forma a variar o sinal de um pino de saída de forma digital, conforme um duty cycle escolhido. Para isso, sua struct de configuração possui os seguintes campos:

- gpio: Struct de configuração do GPIO utilizado pela PWM, deve ser configurado como saída e com a alternate function relativa ao timer escolhido;
- timer: Timer que será utilizado para gerar o sinal de PWM;
- od_id: Canal do timer que será utilizado;
- rcc_clock: O bit que ativa o clock nesse periférico;
- period: A quantidade de ticks até o PWM ser resetado, também define a resolução do sinal;
- clock_div: Define se o clock utilizado será o original ou o mesmo divido por 2 ou 4;
- prescaler: Quantidade de clocks que devem ocorrer para a contagem de 1 tick;
- oc_mode: Define de que forma o sinal se comporta com a presença de um valor de entrada para comparação, no modo PWM, o sinal está em um

estado quando a contagem de ticks é menor que o sinal escolhido, e em outro estado quando a contagem passa esse valor.

Além disso, foram definidas algumas configurações fixas para a utilização da funcionalidade de PWM, como ativar o modo de output-compare, que é necessário para a alterar o duty cycle da onda, ativar o fast mode, que diminui o tempo necessário para variar a saída, e definir a polaridade do canal de saída como ativo em alto.

Dessa forma, para variar o sinal de uma instância da classe de PWM, basta chamar o método set_compare, passando um valor entre 0 e o período definido, que o duty cycle da onda resultante será alterado conforme especificado.

2.1.4 ADC

O ADC é uma classe que implementa funcionalidades do conversor analógico-digital do microcontrolador, ela é implementada como um template, recebendo o número de canais como parâmetro, visto que o array que armazena as medições de cada canal deve ter seu tamanho definido em tempo de compilação. Sua struct de configuração tem os seguintes campos:

- gpio: Struct de configuração do GPIO utilizado pelo ADC, deve ser definida como entrada e configurada no modo analógico;
- adc_number: Número do ADC utilizado pela classe para realizar as conversões;
- mode: Define se os diferentes ADCs vão ser utilizados de forma independente, ou ativados simultaneamente com o dual mode;
- rcc_clock: O bit que ativa o clock nesse periférico;
- rcc_reset: O bit que reseta este periférico específico;
- prescaler: Fator pelo o qual a frequência do clock será dividida antes de entrar no ADC;
- resolution: Quantidade de bits do resultado da conversão, também afeta o tempo necessário para completar a operação;
- channels: Array com os canais que serão utilizados pelo ADC, na ordem em que as conversões devem ocorrer;

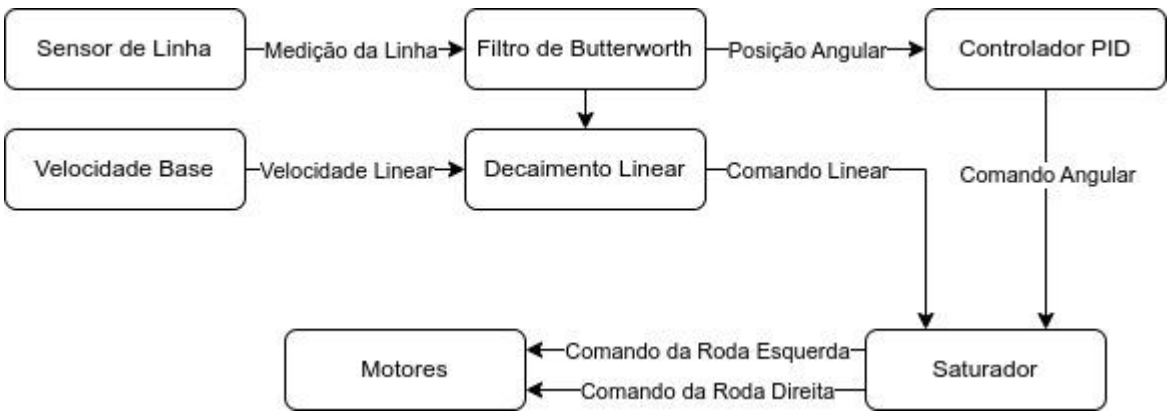
- sample_time: Tempo de amostragem para realizar a medição do valor analógico.

Além dessas, foram definidas algumas configurações fixas dentro do construtor da classe, como definir o alinhamento dos bits do resultado da conversão como right aligned, ativar o modo scan, que garante que todos os canais terão suas medições realizadas, e ativar a configuração para ligar a flag de fim da conversão após cada canal individual ter terminado, ao invés de só ligar ao fim de todos canais.

Também foram implementados métodos para utilizar as funcionalidades da classe de ADC: O update_reading, que atualiza o valor medido em todos os canais; E o get_reading, que retorna o valor medido em um canal específico passado como parâmetro.

2.2 Alto Nível

No alto nível, foi desenvolvido um robô seguidor de linha utilizando a biblioteca HAL. Para isso, também foram implementadas classes que abstraem os sensores e atuadores utilizados no projeto, chamadas de proxy. Além disso, também foram aplicados alguns algoritmos necessários para o bom funcionamento do projeto. Abaixo, está representado um diagrama que demonstra o fluxo de informação, desde a sua aquisição pelos sensores até seu envio para os atuadores.



2.2.1 Botão

A classe do botão utiliza o HAL do GPIO para ter acesso a um pino de entrada, além de lidar com outras funcionalidades específicas desta classe, como negar o debounce e diferenciar os tipos de aperto pelo tempo que o botão ficou pressionado, através da utilização da classe de Timer. E isso sem travar a execução do programa, visto que o timer continua sua contagem através de interrupções do system tick.

2.2.2 Motor

A classe Motor utiliza o HAL da PWM para realizar o acionamento dos drivers do robô, no nosso caso, são necessárias duas PWMs para cada motor, uma responsável para movimentar a roda em um sentido e outra para girar no sentido contrário. é uma classe que implementa funcionalidades de um motor. A classe possui um método que recebe um comando de velocidade de -100 a 100 e mapeia para a escala da PWM, também considerando uma deadzone visando linearizar a curva de velocidade do motor.

2.2.3 Locomoção

A locomoção do robô é feita através de dois motores, de forma que cabe a essa classe receber a velocidade linear e angular desejada e converter esse comando para cada motor, saturando no máximo definido pela classe Motor, mas mantendo a relação entre as velocidades no processo.

Além disso, a classe possui um método estático para realizar o decaimento da velocidade linear, de forma que o robô se movimente mais devagar quando o erro angular for alto, geralmente em curvas.

2.2.4 Sensores de Linha

Esta classe é responsável por lidar com o array de sensores de linha presentes na frente do robô, ela utiliza a classe ADC do HAL, e portanto, também deve ser declarada como um template, recebendo o número de sensores como parâmetro. Além do método principal get_position, que retorna a posição angular do robô em relação à linha, a classe também implementa métodos de calibração, utilizando as medições dos sensores para atualizar as definições de preto e branco da instância da classe.

2.2.5 Filtro de Butterworth

O filtro de Butterworth é um algoritmo utilizado para reduzir ruídos e melhorar a qualidade do sinal, no nosso caso ele é utilizado nas medições do sensor de linha e pode ser configurado variando a sampling frequency, deixando seu efeito no sinal mais intenso ou mais leve.

2.2.6 Controlador P.I.D

Esta classe implementa um controlador PID completamente configurável e com mecanismos de anti-windup para evitar o acúmulo da componente integrativa em momentos indesejados. Todas as constantes do controlador podem ser alteradas no construtor, ou após sua instanciação. A classe Timer também é utilizada, visto que as componentes derivativa e integrativa utilizam o tempo decorrido da última amostragem no cálculo de seus valores.

Na nossa aplicação, esta classe foi utilizada para fornecer um comando de resposta ao erro angular, visando sempre minimizar o ângulo do robô em relação a linha.

3. CONCLUSÃO

Em síntese, a criação e aplicação da biblioteca de Hardware Abstraction Layer (HAL) para o microcontrolador STM32F411CE trouxe soluções eficazes para a complexidade do desenvolvimento de sistemas embarcados. Ao fornecer uma

camada de abstração para as funcionalidades do hardware, a biblioteca simplificou a criação de programas e viabilizou a implementação de projetos mais avançados, como o robô seguidor de linha. A escolha da linguagem C++ e a integração da biblioteca libopencm3 fortaleceram essa abordagem, evidenciando a importância da abstração e modularização para o sucesso no desenvolvimento de sistemas embarcados.