Relatório do Trabalho de Programação em Tempo Real Para Sistemas Embarcados

UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Julho - 2022

Ana Clara M. S. Batista
Salvador, BA
clara.malheiro@ufba.br

André Paiva C. Rodrigues
Salvador, BA
andre.paiva@ufba.br

Pedro A. D. N. Correia
Salvador, BA
pedro.correia@ufba.br

Resumo

Este é o relatório do projeto da disciplina ENGD33 conduzida pelo Professor Jés de Jesus Fiais Cerqueira no primeiro semestre de 2022 com o tópico Solução de Tempo Real para um Problema de Robótica. Nessa subdivisão do projeto, confiada aos alunos descritos acima, foi desenvolvido uma interface via Display com o micro controlador STM32F411CEU6 da placa de desenvolvimento *Blackpill* e com um módulo display com controlador de display ILI9341 e controlador de touchscreen XPT2046. Foram utilizadas diversas funções e aspectos do FreeRTOS aprendidos em sala.

Palavras-Chave: RTOS, Display, ILI9341, SPI, STM32

Introdução

Nesse trabalho procuramos aplicar os conhecimentos da matéria Programação em Tempo Real Para Sistemas Embarcados aprendidos ao longo do semestre em um projeto de robótica. O projeto, que tem como nome Solução de Tempo Real para um Problema de Robótica e faz parte do programa de mestrado do aluno Wellington, tem como objetivo implementar funcionalidades necessárias para o acionamento de uma base robótica móvel e foi divida em sete partes listadas a seguir.

- Controlador de tração e velocidade de um robô omnidirecional de três rodas
- Controlador da posição de um robô omnidirecional de três rodas
- Medidor de atitude e direção
- Datalogger
- Comunicação com Sigle Board Computer (SBC) via ethernet
- Comunicação via UART para monitoramento remoto
- Interface via display

O grupo citado neste relatório foi designado para a última tarefa, a exibição das informações através de interface gráfica projetada em um display de resolução 320x240. Foi utilizado o sistema operacional FreeRTOS para a construção deste sistema de tempo real. A placa de desenvolvimento utilizada para essa tarefa foi a *Blackpill* baseada em microcontrolador STM32F411CEU6, juntamente com gravador/debugger STLINK V2 e o display TFT com controlador ILI9341. O display também possui controlador de touchscreen XPT2046, mas o mesmo apresentou problemas, e para interação do usuário utilizamos três botões físicos. Utilizamos uma biblioteca gráfica que será detalhada a seguir e fizemos algumas modificações e novas funções para se adequar às nossas necessidades, processo este que também será detalhado ao longo do relatório.

Todos os arquivos do trabalho podem ser encontrados através deste repositório no GitHub e no arquivo enviado através do e-mail para o Professor.

1 Descrição do problema

O problema em questão para este trabalho é a coleta e exibição de informações de sensores de um sistema robótico. Dentre todas as informações que trafegam por entre as partes do sistema, o display TFT 320x240 é utilizado para exibir as seguintes telas:

- Valores de velocidade linear e posição do robô em forma numérica;
- Gráficos das velocidades angulares de cada motor;
- Gráficos da intensidade de corrente capturada nos três motores ao longo do tempo.

Sobre o intervalo de tempo de captura, as três informações de posicionamento (X, Y, Z) são coletadas a cada 100ms, já as três informações de velocidade angular são coletadas a cada 10ms e, por fim, as três informações de intensidade de corrente nos motores são coletadas a cada 1ms.

Para a alternância de exibição da tela, duas soluções foram propostas, sendo a primeira solução a mudança de tela por interação do usuário com o touchscreen do display e a segunda solução uma tarefa periódica para alternar automaticamente entre as três telas.

2 Planejamento inicial da solução

Pelo planejamento inicial definido pela equipe, utilizaríamos quatro tipos de tasks:

- Tasks geradoras de dados, com três instâncias, uma para cada tipo de variável;
- Tasks leitoras dos dados gerados, com três instâncias, uma para cada tipo de variável;
- Task para gerenciar a exibição de informações no display;
- Task para receber feedback do touchscreen, ativada mediante interrupção.

3 Testes iniciais com exibição e touch do display

Inicialmente, fizemos testes com a exibição e o touchscreen do display, ainda sem utilização do RTOS. Configuramos a frequência de clock do microcontrolador para 100 MHz. Para o controlador de touchscreen, foi verificado em gráficos de operação no Datasheet do controlador XPT2046 [1] a utilização da frequência de comunicação de 2 MHz. Como taxa de transferência de dados da comunicação SPI da exibição do display foi definida para 50 Mbit/s, utilizamos um segundo canal de SPI com taxa de transmissão de dados por volta de 1,5 Mbit/s.

Para exibição do display, utilizamos a biblioteca STM32_HAL_ILI9341, encontrada no Github e construída pelo usuário "eziya" [2], e para o touchscreen utilizamos alguns arquivos da biblioteca STM32-ILI9341, encontrada no Github construída pelo usuário "martnak" [3].

Conseguimos testar as funcionalidades de exibição na tela: as funções de geração de gráficos funcionaram bem. No entanto, a funcionalidade de leitura do touchscreen, apesar de estar retornando corretamente a requisição de interrupção, não retornou dados corretos em relação à posição (X, Y) da tela em que foi detectado o toque. Ao pesquisar na internet, descobrimos um documento da Texas Instruments [4] falando sobre técnicas de redução de ruído no sinal analógico de detecção de posição de um touchscreen resistivo em um controlador correlato ao XPT2046. Segundo o documento, é necessária a instalação de capacitores de desacoplamento nas vias de sinal analógico, e esses capacitores devem ser dimensionados de modo a não comprometer muito o delay de leitura de posição. Na placa do display utilizada para este projeto não foram identificados capacitores de desacoplamento nas entradas analógicas do XPT2046, e não foram identificadas trilhas para que pudesse ser feita a soldagem dos mesmos em placa.

No documento da Texas Instruments são apresentadas algumas alternativas de tratamento de erros de leitura de coordenadas por software, as quais tentamos implementar, porém não retornaram resultados satisfatórios.

Devido ao problema encontrado com a utilização de touchscreen, decidimos não utilizar o recurso. No entanto, ao invés de produzir uma tarefa para alternância periódica das telas, preferimos manter a interação do usuário com o sistema, inserindo no projeto três botões físicos. Cada um dos botões ativa uma interrupção, e nesta interrupção é feito o tratamento necessário para a troca de tela.

4 Mudanças no planejamento da solução

Com as mudanças devido à não utilização do recurso de touchscreen, passamos a ter os seguintes tipos de tasks:

- Tasks geradoras de dados, com três instâncias, uma para cada tipo de variável;
- Tasks leitoras dos dados gerados, com três instâncias, uma para cada tipo de variável;
- Task para gerenciar a exibição de informações no display;
- Tasks para receber feedback dos três botões físicos, ativadas mediante interrupção.

5 Arquitetura do Projeto e Recursos utilizados

Neste tópico entraremos em detalhes sobre todos os recursos usados e traremos uma visão geral da implementação do projeto em FreeRTOS, abordando tanto as tasks geradas como também os recursos de gerenciamento de interrupções e gerenciamento de transmissão e acesso de informações. Para o desenvolvimento da arquitetura e utilização de recursos, consultamos o livro do FreeRTOS [5].

5.1 Visão geral do funcionamento

Para a descrição do funcionamento geral do sistema implementado, partiremos das tasks de geração de dados. Estas tasks simulam o comportamento de leitura de informações. Criamos três tasks diferentes, uma para cada tipo de variável (corrente, velocidade angular e posição), com os respectivos deadlines simulando o intervalo de tempo de captura (1ms para corrente, 10ms para velocidade angular e 100ms para posição). A cada geração de dados, tais tarefas geram um `dataset` (tipo abstrato de dados criado pela equipe, e que contém três valores no formato `float` e um timestamp em `TickType_t`). Estes `datasets` são inseridos em suas respectivas queues.

Tais queues são lidas por tasks leitoras de dados. As tasks leem estas queues até que fiquem vazias. Para os casos específicos das tasks leitoras das queues de corrente e velocidades angulares, os `datasets` lidos das queues são inseridos em `buffers circulares` declarados em escopo global, outro tipo abstrato de dado definido pela equipe que contém um array de 40 `datasets`, um `uint16_t` de índice de início do buffer e um `uint16_t` de índice de fim do buffer. Os buffers globais tem seu acesso regulado por mutexes, e posteriormente serão lidos para plotagem de gráficos. Para o caso específico da task de leitura de velocidades angulares, são feitos os cálculos das variáveis de velocidade linear V_x, V_y e ω do último dado lido da queue, e então é criado um novo `dataset` com os valores calculados, o qual é armazenado em uma variável `dataset` global de velocidades lineares, cujo acesso é regulado por mutex. Para o caso da task de leitura de posição, a queue é lida até que fique vazia e o último `dataset` lido é gravado em uma variável `dataset` global de posição, cujo acesso também é regulado por mutex. Os `datasets` globais com acesso regulado por mutex são utilizados para a tela de exibição de valores numéricos.

Para selecionar qual tela será exibida, o usuário pode pressionar um dos três botões físicos. Cada botão ativa uma task de interrupção diferente. Existem duas variáveis `uint16_t` globais que armazenam valores inteiros que representam a tela exibida anteriormente e a tela atual. Uma task engatilhada pela interrupção ativada pelo botão altera o valor da variável que armazena a tela atual, e esta variável será acessada pela task de gerenciamento do display.

A task de gerenciamento do display, toda vez que é chamada, faz a comparação entre as variáveis de tela anterior e tela atual, para verificar se houve alguma mudança. Se o usuário tiver acionado uma mudança de tela, as variáveis serão diferentes, e então, é plotada a base da tela (informações como título da tela, plot de informações que ficam fixas na tela, como títulos de variáveis, legendas, eixos de gráfico, etc.). Logo depois, são plotados os dados na tela, o que é feito por meio de funções auxiliares.

No caso da tela 1, a função auxiliar acessa a variável `dataset` global via mutex, faz uma cópia para uma variável `dataset` local e libera o mutex. Os valores, então, são lidos da cópia local do `dataset` e exibidos na tela em suas respectivas posições com sua devida formatação.

No caso das telas 2 e 3, as respectivas funções auxiliares acessam a variável `buffer circular` global via mutex e fazem uma cópia local do `buffer circular`, liberando o mutex depois da cópia. Os datasets então são lidos e é feita a adequação dos valores para a escala de pixels, e então é feito o plot do gráfico.

5.2 Biblioteca gráfica e modificações

Para a exibição de dados na tela, foram usadas funções auxiliares resultantes da composição e adaptação de bibliotecas gráficas encontradas no Github. Utilizamos como base os arquivos da biblioteca encontrada no Github do usuário “eziya” [2], de modo que deletamos as funções não utilizadas e mantivemos apenas as funções utilizadas. No entanto, a biblioteca não dispunha de uma função importante: a função de desenho de linhas na diagonal, a qual é importante para o plot dos gráficos. As funções originais de desenho de linha da bibioteca do usuário “eziya” apenas desenhavam linhas na horizontal e na vertical. Fizemos a adaptação de uma função de desenho de linhas na diagonal encontrada em outra biblioteca no Github do usuário “ardnew” [6].

5.3 Códigos desenvolvidos

Criamos três arquivos adicionais além dos arquivos gerados automaticamente pela IDE, que são os arquivos `lookuptable.h`, `usertypedefs.h` e `usercode.h`, sendo o primeiro arquivo gerado automaticamente por um script em Python desenvolvido pela equipe.

5.3.1 Geração de números aleatórios no arquivo `lookuptable.h`

Desenvolvemos um script em Python que gera automaticamente os vetores de dados utilizados pelas tasks geradoras já num formato utilizável por códigos em C. Tais vetores são automaticamente gravados no arquivo `lookuptable.h`. O script é o arquivo `rng.py` encontrado na pasta `DatagenPy` no repositório do GitHub do projeto).

5.3.2 O arquivo `usertypedefs.h`

No arquivo `usertypedefs.h` são definidos os tipos abstratos de dados `dataset` e `circle_buffer`, que são utilizados para transmissão e manipulação de dados por entre as tasks.

5.3.3 O arquivo `usercode.h`

O arquivo `usercode.h` é o arquivo principal do projeto. Nele constam todas as tasks e funções auxiliares, bem como a função `inicializar()`, que faz a inicialização prévia do display, e `userRTOS()`, que faz a criação de tasks e inicialização do escalonador.

As funções auxiliares são, em sua maioria, para plotagem de informações na tela, salvo as funções de leitura e escrita de dados em `buffer circular` e a função de gerenciamento de callbacks das interrupções para execução das respectivas tasks do FreeRTOS.

As funções de inserção de dados em buffer circular sempre inserem dados no final do buffer. A função de leitura de dados de um buffer circular abstrai o deslocamento de índices no vetor, o que facilita na construção da lógica de acesso aos dados.

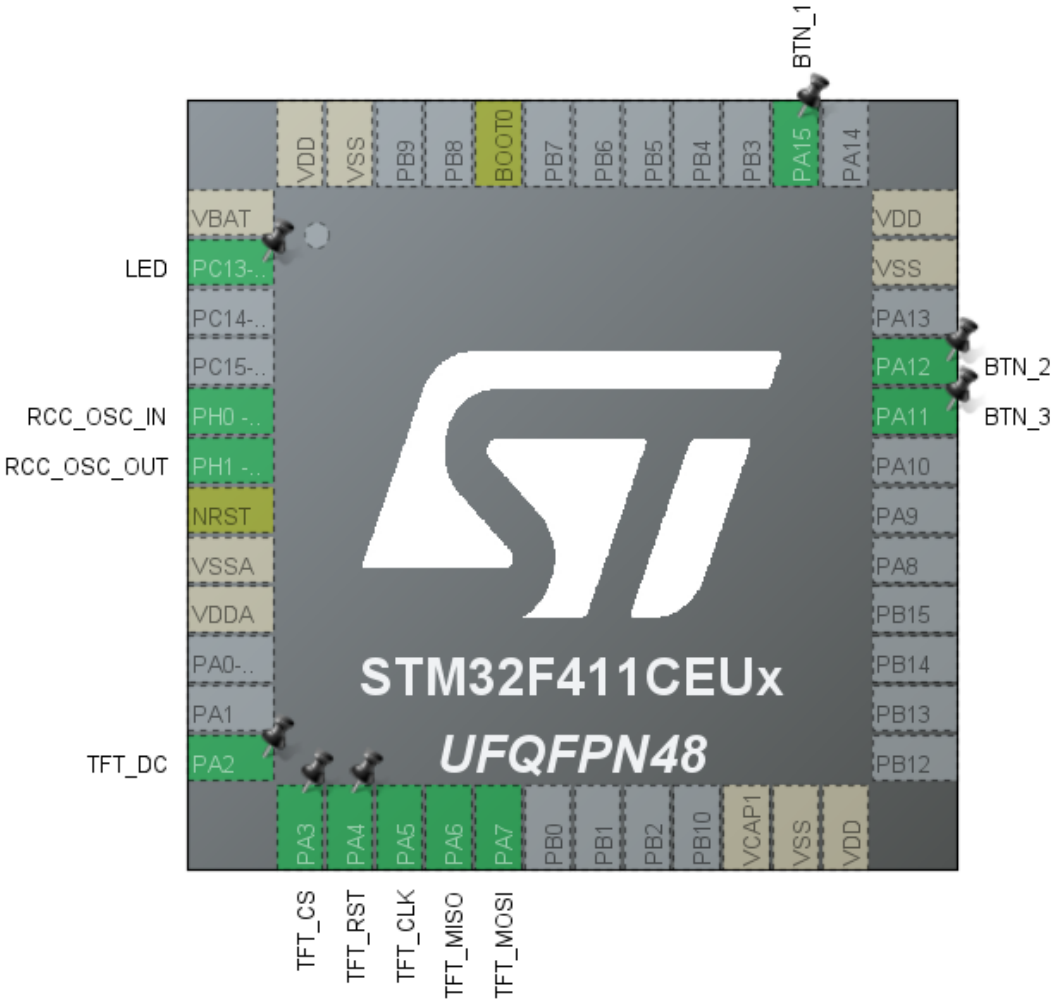

Figura 1 – Pin planning do projeto.

6 Montagem, testes e debug

O pin planning do projeto é visível na Figura 1. Utilizamos os pinos PA5, PA6 e PA7 para comunicação SPI com o display, bem como os pinos PA2, PA3 e PA4 para controle do mesmo. Os botões foram associados às entradas PA11, PA12 e PA15, os quais estão associados com a linha de interrupção EXTI Line [15:10]. Na Figura 2 é possível ver como ficou a montagem.

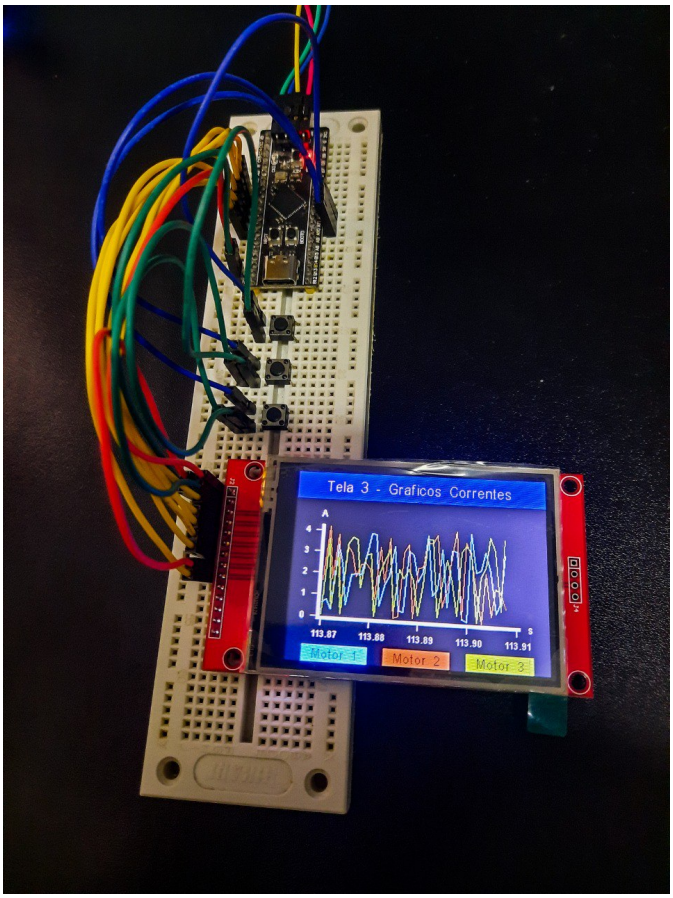

Figura 2 – Circuito montado em protoboard.

Durante a implementação das funções nos deparamos com algumas situações inesperadas que nos levaram à utilizar recursos de depuração. Em algumas vezes, os problemas eram sanados apenas com revisão de código, e em outras, foi necessário o uso do cenário de debug da IDE.

Em algumas situações a aplicação travou e não tornou a executar novamente até ser dado o reset. Em uma das ocasiões a causa foi *stack overflow*, e o tamanho da *stack* de algumas tarefas foi reajustado, sanando o problema. Em outra ocasião, o problema se deu por má configuração do mutex: algumas rotinas estavam se apropriando do mutex e não estavam liberando o mesmo ao final das operações na seção crítica. Tais problemas relacionados a mutexes foram sanados adicionando a instrução `xSemaphoreGive()` nos trechos de código em que era necessário.

Também houveram erros relacionados à transmissão e exibição de dados no display. Alguns dados estavam sendo exibidos como "0.00", mesmo com valores aleatórios presentes na *look up table*. Tais erros foram sanados aumentando o tamanho do memory heap, e provavelmente esse aumento se fez necessário em razão do tamanho considerável que estabelecemos para as queues de transmissão de dados entre as tasks geradoras e leitoras.

Durante a construção da função de plotagem de gráfico, encontramos condições bem específicas de operação nas quais eram plotados retângulos da mesma cor das linhas do gráfico, sobrepondo informações na tela. Analisando o código da função `ILI9341_DrawLine`, a qual foi criada com base em adaptações de uma função encontrada em biblioteca no Github [6], encontramos um erro na função que fazia com que tais retângulos fossem gerados. Corrigimos o problema na adaptação da função e o problema foi sanado.

Durante a execução da plotagem dos gráficos, notamos que alguns pontos bem específicos extrapolaram a área do gráfico que foi delimitada e chegavam até o topo da tela. Utilizando dos recursos de debug e breakpoints nas tasks de geração de dados, descobrimos que havia um erro na configuração do range de índices da *look up table* a serem acessados. Fizemos a correção do range de índices e o problema foi sanado.

Encontramos um problema que quase passou despercebido: na função de leitura de dados de um buffer circular, também houve erro na configuração de range de índices acessíveis, o qual também foi sanado.

Nos deparamos com o travamento da aplicação durante a plotagem de gráficos: a aplicação não mais respondia aos comandos de interrupção engatilhados pelos botões. Descobrimos que a aplicação não estava conseguindo plotar as informações do gráfico em sua completude em tempo hábil antes do deadline da task gerenciadora de display. A solução para este problema foi aumentar a periodicidade da task de atualização da tela.

Após todas as correções de bugs, fizemos testes de uso e não encontramos mais problemas no que diz respeito ao escalonamento do sistema e à integridade dos dados. Na Figura 3 podem ser visualizadas as telas exibidas pela aplicação após todas as correções de bug acima citadas.

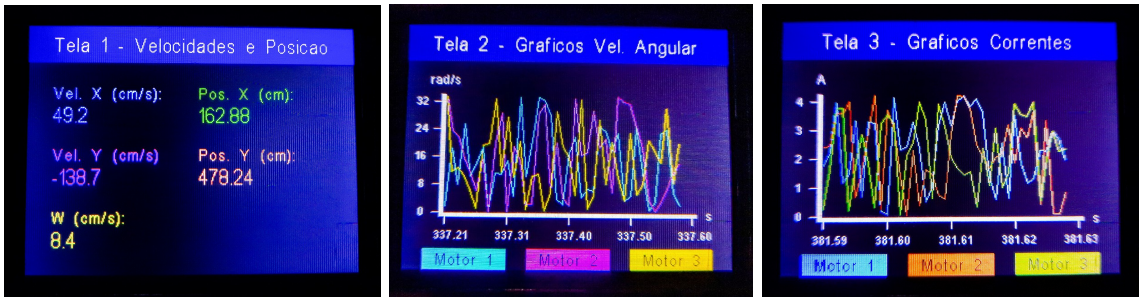

Figura 3 – Telas exibidas pelo display.

7 Resultados e conclusão

Os resultados de implementação mostram que conseguimos desenvolver uma aplicação de tempo real escalonável e funcional utilizando o port de FreeRTOS para o microcontrolador utilizado. Por meio do trabalho foi possível exercitar na prática os conceitos de sistemas de tempo real estudados nesta disciplina. Nos deparamos com alguns desafios de implementação e limitações de recursos, já que se trata de um sistema embarcado baseado em microcontrolador, os quais conseguimos contornar e trazer resultados satisfatórios.

Referências

[1] XPT2046 Datasheet, Shenzhen XPTEK, 2007. [Online]. Disponível em: http://grobotronics.com/images/datasheets/xpt2046-datasheet.pdf. [Acesso em: 27/06/2022].

[2] Biblioteca STM32_HAL_ILI9341, "eziya", 2019. [Online]. Disponível em: https://github.com/eziya/STM32_HAL_ILI9341/tree/master/Src. [Acesso em: 27/06/2022].

[3] Biblioteca STM32-ILI9341, "martnak", 2018. [Online]. Disponível em: https://github.com/martnak/STM32-ILI9341. [Acesso em: 27/06/2022].

[4] Reducing Analog Input Noise in Touch Screen Systems, Texas Instruments, 2007. [Online]. Disponível em: https://www.ti.com/lit/an/sbaa155a/sbaa155a.pdf. [Acesso em: 27/06/2022].

[5] Mastering the FreeRTOS™ Real Time Kernel, Richard Barry, 2016. [Online]. Disponível em: https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. [Acesso em: 27/06/2022].

[6] Biblioteca ILI9341-STM32-HAL, "ardnew", 2021. [Online]. Disponível em: https://github.com/ardnew/ILI9341-STM32-HAL. [Acesso em: 27/06/2022].