CountCloud – Solução na nuvem para contagem de peças produzidas

Júlio Azevedo da Costa

Área temática: Indústria.

RESUMO

A região do Seridó Potiguar é amplamente reconhecida em nível nacional por sua forte tradição costureira, o que impulsionou o crescimento da indústria têxtil local. No entanto, os processos produtivos ainda fazem uso limitado de tecnologia, restringindo-se, em grande parte, às máquinas de costura.

O CountCloud tem como objetivo modernizar a logística da contagem de peças produzidas em um determinado contrato, por meio de um sistema baseado na placa BitDogLab. A solução conta com menus interativos que permitem adicionar, subtrair e resetar um contador visual implementado na matriz de LED, além de um contador na lógica de programação que, via conexão Wi-Fi, transmite os dados para um servidor no Thingspeak.

Como resultado, a solução foi totalmente implementada, garantindo todas as funcionalidades acima descritas e servindo de protótipo para um sistema mais robusto que pode ser construído no futuro.

Palavras-chave: Seridó, indústria têxtil, modernização, BitDogLab, logística.

INTRODUÇÃO

No contexto atual do Seridó Potiguar as fábricas têxteis são uma importante fonte de renda para muitas famílias. Sua produção funciona com base em contratos, uma empresa como a Guararapes busca a facção para terceirizar sua produção, envia um modelo dos padrões de qualidade que deseja que as peças sejam entregues e a quantidade de peças que serão produzidas. Muitas vezes na planta fabril da facção peças com padrões diferentes e de contratos diferentes estejam sendo produzidos em paralelo, o que cria um verdadeiro desafio logístico.

O grande problema é que esse desafio logístico é resolvido à mão, cada funcionário anota quantas peças foram produzidas por ele marcando um risco numa caderneta, e ao final da linha de produção um supervisor é responsável por contar a quantidade de peças feitas e separá-las pelos contratos.

Esse modelo é propenso a problemas como erros de cálculo, dificuldade de saber em qual estágio de progresso a produção de um contrato específico num determinado momento e cria atritos entre funcionários e empresas quando são questionados se a sua contagem está correta ou têm que fazer hora extra já que as peças para o contrato não foram produzidas em sua totalidade.

Com base nisso, o principal objetivo do CountCloud é criar uma solução simples de implementar, barata, criando uma abordagem de contagem precisa, ágil, visual e que disponibilize essa informação através da internet, sem que haja uma mudança muito grande na forma como já é feito o processo de contagem, buscando a maior simplicidade possível na implementação.

O projeto apresentado busca ser um protótipo de solução que pode ser desenvolvida para contribuir nesse contexto, proporcionando informações importantes que podem ajudar as empresas a ter um maior controle do processo produtivo, tendo como objetivo dar uma visão instantânea da quantidade, e em futuras versões da qualidade, do que está sendo produzido no chão de fábrica.

METODOLOGIA

O material utilizado foram a placa BitDogLab e seus periféricos: display OLED para exibição do menu do projeto, joystick para navegação no menu, botões A e B para selecionar opções ou voltar do menu 2 para o menu 1, matriz de LED para mostrar visualmente o estágio de contagem das peças e placa Wi-Fi para transmissão dos dados para a nuvem.

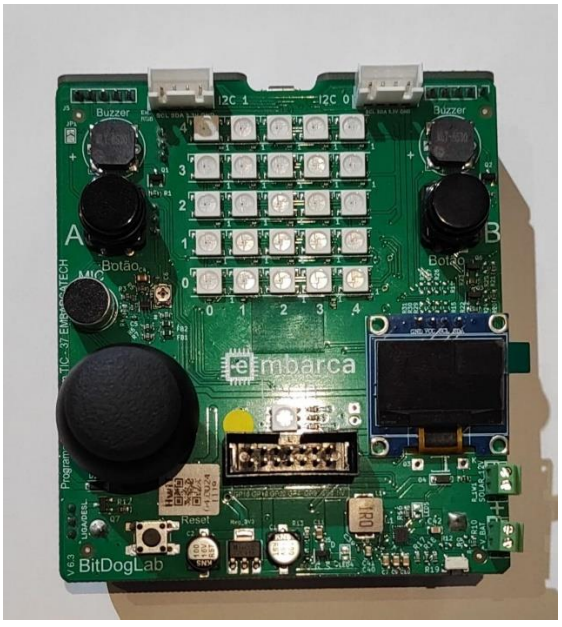

Fig. 1 – Placa de desenvolvimento BitDogLab e seus periféricos.

1. Display OLED

A parte do display OLED baseada no código Display OLED do Github da BitDogLab[1], a implementação do display utilizou imagens no formato bitmap, essa forma foi escolhida para facilitar e ao mesmo tempo deixar visualmente mais interessante, a comunicação entre o Raspberry Pi Pico W e o monitor OLED é feito através de comunicação I2C utilizando as bibliotecas hardware/i2c.h e o driver da placa inc/ssd1306.h, também disponibilizado pela BitDogLab. A conexão física entre os dispositivos é feita através dos pinos 14 e 15 da placa, respectivamente responsáveis pelo SDA e SCL.

Para diminuir a poluição visual dos códigos dos vários menus foi criado um arquivo separado do arquivo principal chamado menu.c e seu respectivo menu.h, esses arquivos foram deixados dentro de uma pasta chamada de func, onde outros arquivos separados com funções importantes para o código principal também serão colocados posteriormente.

Para o funcionamento mais rápido da impressão dos bitmaps na tela foi necessário realizar uma alteração na função ssd1306_draw_bitmap no driver ssd1306_i2c.c da tela oled, retirando a função ssd1306_send_data de dentro do for e colocando-a para fora, como mostrado a seguir:

```
void ssd1306_draw_bitmap(ssd1306_t *ssd, const uint8_t *bitmap) {
    for (int i = 0; i < ssd->bufsize - 1; i++) {
        ssd->ram_buffer[i + 1] = bitmap[i];
    }
    ssd1306_send_data(ssd);
}
```

Fig. 2 – Alteração realizada na função ssd1306_draw_bitmap do driver da tela OLED.

As imagens foram feitas a partir de um vídeo voltado para Arduino do canal do Youtube upir[2] que usa a mesma tela SSD1306, utilizando o site Photopea foram feitas no total 9 imagens, sendo três para o primeiro menu e seis para o segundo, as imagens utilizadas no menu estão disponibilizadas em anexo e podem ser vistas abaixo:

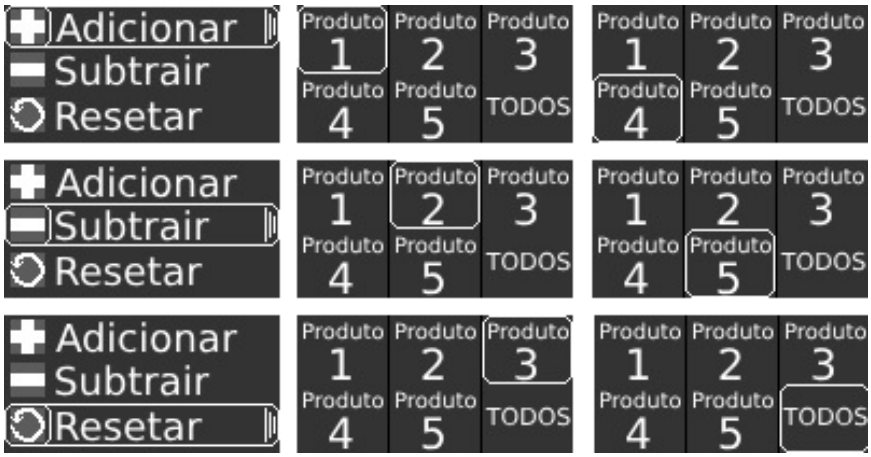

Fig. 3 – Imagens que serviram como base para criação dos códigos bitmap.

A conversão das imagens do formato jpg para o código em bitmap foi feita com base em vídeo do canal do Youtube Prof Jivago[3], esses códigos foram salvos em funções dentro do arquivo menu.c sendo uma função para cada imagem do menu.

De início o código apresentava erro e não exibia a imagem corretamente, então foi utilizado o código enviado na tarefa 6.2 para rodar com o código dos bitmaps que foram feitos, continuando o erro, que consistia nas primeiras imagens feitas terem tamanho 128x68 e não 128x64, após a correção o código funcionou normalmente. Para fazer o teste foi criado um loop simples que ia de uma imagem até a próxima até todas as imagens serem exibidas.

2. Joystick

O Joystick é responsável pela navegação nos menus, no primeiro menu a movimentação é unicamente no eixo vertical, já no segundo os dois eixos são utilizados para uma navegação lateral. Nesse caso, o código-base utilizado foi feito pelo próprio autor[4], durante a primeira semana com a placa foram realizados vários testes tentando aprender por conta própria como usar cada componente da placa individualmente, esse código foi reaproveitado tanto na tarefa 6.2 como nesse projeto.

Os dados são coletados usando a biblioteca hardware/adc, através de Conversores Analógicos Digitais(ADC) conectados fisicamente ao joystick através dos pinos 26 e 27 e aos ADCs 0 e 1 da placa, respectivamente responsáveis pelo eixo Y e pelo eixo X do joystick. A coleta de dados acontece sempre que os valores para um dos eixos ultrapassarem os limites menor que 50 ou maior que 4050.

O principal entrave dessa abordagem foi inserir a condição da mudança de valor no eixo X no while, como essa condição não foi colocada durante algum tempo a única forma de haver mudança na imagem exibida no menu 2 era quando o eixo Y era acionado, com o tempo a alteração foi realizada e o funcionamento foi normalizado.

Ao fim da execução da navegação é feito o uso da função pausa para parar o código durante 100ms, evitando excesso de checagens das variáveis adc_x e adc_y.

3. Botões

Foi utilizado como base para o funcionamento dos botões o código Button LED RGB do Github da Bitdoglab[5] o botão A é responsável por selecionar as opções dentro do menu, já o botão B por voltar do menu 2 para o menu 1.

Os botões são conectados fisicamente na placa através dos pinos 5 e 6, responsáveis respectivamente pelo botão A e pelo botão B. A coleta dos dados é feita através da verificação dos valores nos pinos utilizando funções do GPIO presentes na biblioteca pico/stdlib.

O teste realizado foi entrar e sair dos menus repetidas vezes, buscando estressar ao máximo o código até que o resultado fosse um funcionamento satisfatório.

A função pausa sempre é chamada ao pressionar um dos botões, pausando o programa durante 500ms para evitar problemas com debounce.

4. Matriz de LED

A matriz de LED é utilizada para dar uma demonstração visual do estado da contagem atual, para esse código foi utilizado como base tanto o código Neopixel Pio do Github da BitDogLab[6] como um também um código de criação própria que originalmente servia para jogar jogo da velha na Matriz de LED[7].

A placa e a matriz de LED são fisicamente conectadas através do pino 7, para o seu funcionamento são utilizadas as bibliotecas hardware/clocks, hardware/pio e o driver da matriz ws2812b.pio disponibilizado projeto do Github da BitDogLab, foi necessário fazer a compilação da biblioteca utilizando um site do Wowki[8] para obter o arquivo ws2812b.pio.h.

O trecho de código que faz os cálculos necessários, implementa a lógica dos contadores de produtos e faz o controle da Matriz de LED foi separado em um arquivo chamado corematrix.c e seu respectivo corematrix.h que podem ser encontrados dentro da pasta func.

Foram realizados muitos testes nessa lógica, usando printfs em conjunto com o monitor serial para verificar os valores presentes dentro do vetor de contadores e também se o código estava entrando nos ifs e outras condicionais da forma esperada.

Essa parte do código é a mais complexa pois realiza tanto a lógica da contagem dos produtos quanto a implementação da visualização na Matriz de LED, foram precisos muitos testes e validações para que o resultado final fosse satisfatório.

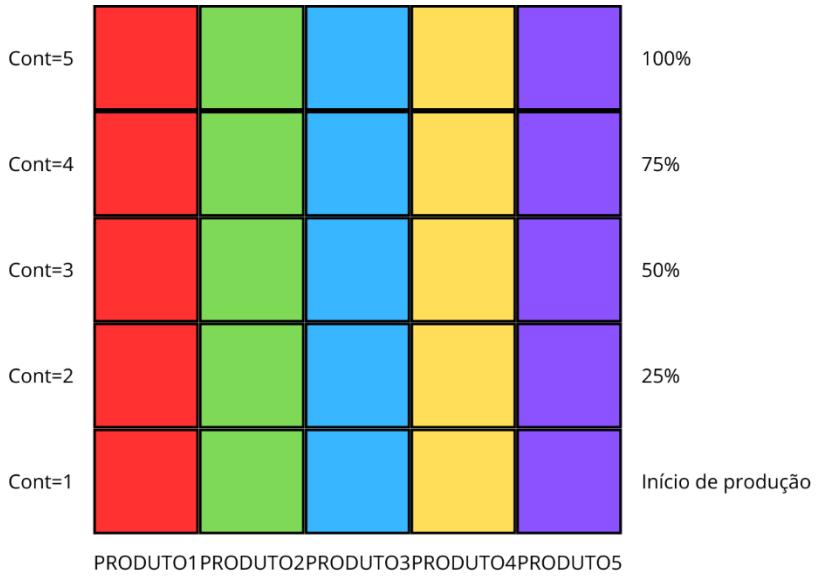

Fig. 4 – Lógica de funcionamento da matriz de LED.

A matriz de LED foi configurada de forma que cada produto será uma coluna com uma cor diferente indicando o estágio da sua contagem, para fins de prototipagem a placa está configurada para contar de um até cinco, mas a ideia final é que a contagem seja percentual, indicando estágios por porcentagem como exemplificados na direita da figura, ou por valores maiores do que cinco.

5. Placa de rede Wi-Fi

A Raspberry Pi Pico W possui uma placa de rede CYW43439, utilizada no projeto para realizar a comunicação entre a placa e o servidor na nuvem ThingSpeak, essa comunicação foi feita com base no código disponibilizado pelos tutores do Embarcatech[9].

Foram realizadas algumas modificações para adaptar à realidade do código, sendo criados os arquivos separados chamados wifi_embarcatech.c e seu respectivo wifi_embarcatech.h que podem ser encontrados dentro da pasta func.

Para integrar o código de conexão ao serviço do ThingSpeak com a main do projeto, foi criada uma função de Callback que usará temporizador para interromper o loop principal do programa, onde está o código principal, e chamar a função wifi_embarcatech. O funcionamento da função de Callback foi testada através de um printf ao final do loop principal do programa e outro printf dentro da função do temporizador que reiniciava a função de callback, indicando que o loop principal de fato havia sido quebrado, que o loop infinito e a função de Callback haviam sido iniciadas.

Com relação as alterações feitas na função wifi_embarcatech, foi preciso indicar no wifi_embarcatech.h o uso do vetor global cont, onde estão armazenados os valores do contador, chamando a função como extern no cabeçalho da função. Também foi necessário

retirar o código while já que não seria mais um firmware e sim apenas um dos programas do projeto. Por fim foi adaptada a string da requisição HTTP para que fossem enviados os valores de todas as posições do vetor cont e também de uma variável chamada cont_total, responsável por mostrar a soma de todos os produtos.

RESULTADOS ALCANÇADOS E DISCUSSÕES

Para um melhor entendimento do programa, as etapas sequenciais que serão explicadas nesse tópico foram separadas da seguinte forma: Inicialização, Menu, Navegação, CoreMatrix e Wifi. O fluxograma com o funcionamento total do programa está disponível em anexo e pode ser visualizado na imagem:

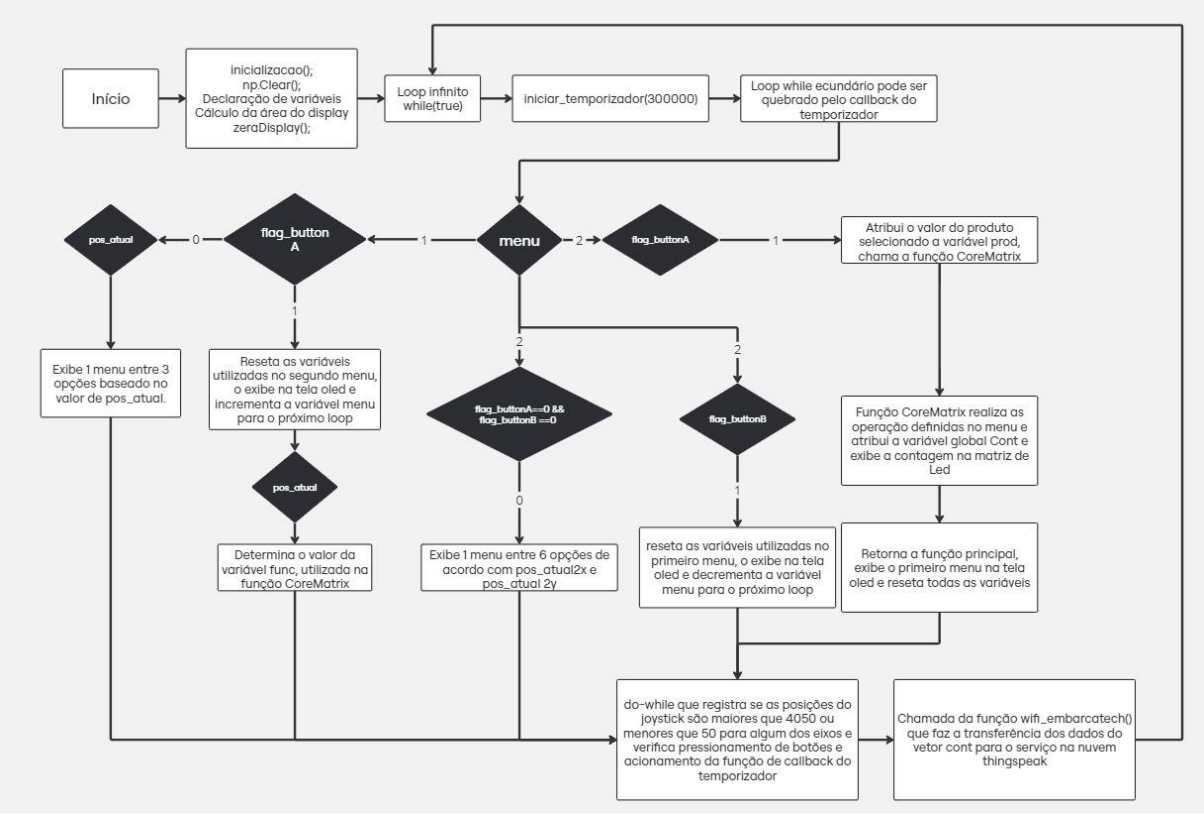

Fig. 5 – Fluxograma geral do programa.

O funcionamento esperado do programa é permitir que o usuário navegue entre dois menus, no menu 1, é possível adicionar ou subtrair uma unidade de um produto, além de realizar um reset total da contagem. No menu 2, o usuário pode escolher se a ação será aplicada a um produto específico ou a todos os produtos ou voltar para o menu 1.

Após a seleção, a quantidade de cada produto será exibida em uma matriz de LED, com cores distintas para facilitar a identificação. Além disso, a contagem é automaticamente

enviada via Wi-Fi para a plataforma ThingSpeak a cada cinco minutos, garantindo o monitoramento remoto dos dados em tempo real.

1. Inicialização

Nessa etapa do código foram chamadas as bibliotecas do Raspberry Pi Pico utilizadas no código principal do projeto, também foram chamadas as bibliotecas feitas especificamente para esse projeto que estão presentes na pasta func e são definidas constantes para os pinos SDA e SCL da tela OLED, dos pinos para o eixo X e eixo Y do Joystick, dos pinos dos botões A e B e também criada um vetor global chamado cont que possui cinco espaços e será o responsável por armazenar a quantidade de cada um dos cinco produtos.

```c
//Declaração de variáveis e de funções utilizando os temporizadores para chamar a função wi-fi de 5 em 5 minutos
//e para servir de espera para o debounce dos botões e dos joysticks
volatile bool tempo_acabou = false;
volatile uint64_t tempo_limite = 0;

int64_t callback_temporizador(alarm_id_t id, void *user_data) {
    tempo_acabou = true;
    return 0;  // Retorna 0 para não repetir o timer
}

void iniciar_temporizador(uint tempo_ms) {
    tempo_acabou = false;
    add_alarm_in_ms(tempo_ms, callback_temporizador, NULL, false);
}

//Função de substituição ao sleep_ms, utiliza a função time_us_64, feita dessa forma para que a marcação de tempo
//do programa seja feita utilizando temporizador
void pausa(uint tempo_ms) {
    uint64_t tempo_atual = time_us_64() / 1000;
    if (tempo_atual < tempo_limite) {
        while (time_us_64() / 1000 < tempo_limite) {
        }
    }
    tempo_limite = time_us_64() / 1000 + tempo_ms;
}
```

Fig. 6 – Criação de variáveis do tipo volátil, da função de Callback do temporizador do wi-fi e declaração da função pausa que substitui o uso do sleep_ms.

Foi criada uma variável volátil do tipo booleana para indicar ao programa quando o temporizador finalizou a contagem de tempo para chamar a função de envio de dados via Wi-Fi, além disso também foram criadas uma função de Callback e uma função para iniciar o temporizador e indicar quanto tempo até ativar a função de Callback, também foi criada a função pausa para substituir o uso da função sleep_ms, utilizando o temporizador da placa. Por fim foi feita uma função inicializacao() que realiza o setup de todos os componentes que são utilizados no main do projeto.

Na função main são chamadas as funções inicializacao() e npClear(), função do arquivo corematrix responsável por limpar a matriz de LED e impedir que o programa inicialize com qualquer LED aceso. É criada uma struct que vai conter as dimensões da tela de led e aonde ela começa e termina, além de calcular a área de renderização do display utilizando uma função do driver do OLED do arquivo ssd1306_i2c.c, localizado na pasta inc. É chamada a função zeraDisplay do arquivo menu e por fim, é iniciado o loop infinito, o temporizador é iniciado com um timer de 300000ms que equivale a cinco minutos e o while secundário é

iniciado, esse while só será quebrado quando os cinco minutos do temporizador passarem e fará com que a função do Wi-Fi que envia os dados ao ThingSpeak seja acionada.

2. Menu

Para a exibição dos menus o programa conta com uma série de condicionais que levam à exibição do bitmap desejado a depender das variáveis menu, pos_atual1, pos_atual2x e pos_atual2y.

A variável menu define qual dos dois menus o usuário está visualizando, enquanto a variável pos_atual1 serve para indicar qual das três imagens de bitmap será exibida quando o usuário estiver no menu 1, sendo análoga às variáveis pos_atual2x e pos_atual2y no menu 2, nesse caso serão duas já que no segundo menu é possível navegar pelos dois eixos.

```
if(menu==1){
    //Condicional da navegação no menu inicial
    if(flag_buttonA==0){
        if (pos_atual1 == 1) {
            zeraDisplay(frame_area);
            MenuA1(frame_area);
        }
        else if (pos_atual1 == 2) {···
        else if (pos_atual1 == 3) {···
    }

    //Condicional de pressionamento do botão A
    else if(flag_buttonA==1){
        flag_buttonA=0;
        pos_atual2x=1; pos_atual2y=1;
        menu++;
        zeraDisplay(frame_area);
        MenuB1(frame_area);
        if(pos_atual1==1){
            func=1;
        }
        else if(pos_atual1==2){···
        else if(pos_atual1==3){···
    }
}
```

Fig. 7 – Estrutura de condicionais que fazem a lógica do menu 1.

A primeira variável que é verificada será a menu, caso ela seja 1 o código acima será executado, as condicionais seguintes verificarão se o botão foi ou não pressionado na etapa de navegação, que será discutida posteriormente, caso não tenha sido apertado será exibido uma das três imagens para o menu 1 a depender da variável pos_atual1, como pode ser visto no if que foi deixado aberto.

Caso o botão seja pressionado, as variáveis referentes ao menu 2 são resetadas e a variável menu incrementada, a imagem do OLED será trocada para a do próximo menu e a

depender da pos_atual a variável func será alterada para um valor que é utilizado pela função corematrix para definir qual a opção selecionada entre três: adicionar, subtrair e resetar.

```
else if(menu==2){
    //Condicional de navegação no segundo menu
    if(flag_buttonA==0 && flag_buttonB==0){
        if(pos_atual2y==1 && pos_atual2x==1){
            zeraDisplay(frame_area);
            MenuB1(frame_area);
        }
        else if(pos_atual2y==1 && pos_atual2x==2){ ···
        else if(pos_atual2y==1 && pos_atual2x==3){ ···
        else if(pos_atual2y==2 && pos_atual2x==1){ ···
        else if(pos_atual2y==2 && pos_atual2x==2){ ···
        else if(pos_atual2y==2 && pos_atual2x==3){ ···
    }
    //Condicional de pressionamento do botão B no segundo m
    else if(flag_buttonA==1){
        if(pos_atual2y==1 && pos_atual2x==1){
            prod=0;
            CoreMatrix(func, prod, cont, flag_coreMatrix);
        }
        else if(pos_atual2y==1 && pos_atual2x==2){ ···
        else if(pos_atual2y==1 && pos_atual2x==3){ ···
        else if(pos_atual2y==2 && pos_atual2x==1){ ···
        else if(pos_atual2y==2 && pos_atual2x==2){ ···
        else if(pos_atual2y==2 && pos_atual2x==3){ ···
        MenuA1(frame_area);
        flag_coreMatrix=1;
        menu=1;
        pos_atual1 = 1;
        pos_atual2y=1; pos_atual2x=1;
        func=0, prod=0;
        flag_buttonA=0; flag_buttonB=0;
    }
//Condicional de pressionamento do botão B no segundo menu
    else if(flag_buttonB==1){ ···
}
```

Fig. 8 – Estrutura de condicionais que fazem a lógica do menu 2.

Quando a variável menu for igual a 2, primeiramente será verificado se os botões A e B foram pressionados, caso não tenham sido será exibido o menu correspondente às posições definidas pelas variáveis pos_atual2y e pos_atual2x.

Caso o botão A seja pressionado, a variável prod receberá um valor indicando qual produto foi selecionado para incrementar, subtrair ou resetar, as informações são enviadas para a função corematrix onde serão processadas e por fim, o menu 1 volta a ser exibido e todas as variáveis utilizadas até aqui são resetadas. Caso o botão B seja pressionado, a lógica do programa voltará do menu 2 para o menu 1.

Em cada uma dessas condicionais mostradas quando nenhum botão for pressionado tanto para o menu 1 como para o menu 2 haverá uma função de menu do arquivo menu.c localizada na pasta func que contém as funções do driver da tela OLED SSD1306_i2c, essas funções possuem as imagens em bitmap referentes a cada um dos nove menus do projeto e

também que são responsáveis por fazer a comunicação via I2C entre a placa e a tela OLED, imprimindo o conteúdo dos bitmaps na tela.

3. Navegação

```
do{
adc_select_input(1);
adc_x = adc_read();
adc_select_input(0);
adc_y = adc_read();
if(menu==1){
    if (adc_y < 50) {
        if (pos_atual1 != 3) {
            pos_atual1++;
        }
    }
    else if (adc_y > 4050) {···
    else if(gpio_get(ButtonA)==0){
        flag_buttonA=1;
    }
}
else if(menu==2){
    if(adc_x < 50){
        if(pos_atual2x != 1){
            pos_atual2x--;
        }
    }
    else if(adc_x > 4050){···
    else if (adc_y < 50) {
        if (pos_atual2y != 2) {
            pos_atual2y++;
        }
    }
    else if (adc_y > 4050) {···
    else if(gpio_get(ButtonA)==0){
        flag_buttonA=1;
        pausa(500);
    }
    else if(gpio_get(ButtonB)==0){
        flag_buttonB=1;
        pausa(500);
    }
}
pausa(100);
//condicionais para a saída do do-while de navegação
}while(adc_y>50 && adc_y<4050 && adc_x>50 &&
     adc_x<4050 && flag_buttonA!=1 && flag_buttonB!=1
      && !tempo_acabou);
```

Fig. 9 – Estrutura do do-while de navegação.

O do-while de navegação funciona coletando informações do Joystick nos eixos X e Y, através de leituras no ADC 1 e 0 da placa e depois verifica em qual dos dois menus o programa está, já que o primeiro menu se move apenas no eixo y enquanto o segundo se move nos dois eixos. A segunda verificação ocorre para analisar se houve uma mudança no valor para menor que 50 ou maior que 4050 tanto no eixo Y como no eixo X, também são feitas verificações sobre o pressionamento dos botões A ou B, caso qualquer uma dessas verificações aconteça ou o tempo de cinco minutos do temporizador acabe, o laço do-while de navegação será interrompido o while secundário iniciará um novo ciclo.

Caso o temporizador acione a função de Callback, o while secundário também é interrompido e a função do Wi-Fi será ativada, o que levará a um novo ciclo do while de looping infinito.

4. CoreMatrix

A função corematrix é responsável por pegar os dados enviados pela navegação dos menus através das variáveis func e prod, realizar a operação desejada e imprimir na matriz de LED o resultado final dessa contabilidade.

```
int CoreMatrix(int func, int prod, int* cont, int

    int leds_matrix[5][5] = {
        {4, 5, 14, 15, 24},  // LEDS Produto 1
        {3, 6, 13, 16, 23},  // LEDS Produto 2
        {2, 7, 12, 17, 22},  // LEDS Produto 3
        {1, 8, 11, 18, 21},  // LEDS Produto 4
        {0, 9, 10, 19, 20}   // LEDS Produto 5
    };
    npClear();
    // Inicializa os LEDs
    if(flag_coreMatrix == 0){
        npInit(LED_PIN);
        // setup_led();      // Configura o PWM
    }
```

Fig. 10 – Inicialização CoreMatrix.

Na inicialização da função, temos a criação da variável chamada leds_matrix, uma matriz 5x5 que detém todos os índices correspondente às posições dos leds na matriz que serão utilizados no momento de imprimir os valores na matriz de LED, além disso é dado npClear() mais uma vez para garantir que a matriz esteja vazia e por fim, caso seja a primeira vez que a função corematrix esteja sendo utilizada inicializará o pino de saída correspondente a matriz de LED.

```
if (func == 1) {
    if (prod != 5) {
        if(cont[prod]<5){
            cont[prod]++;
        }
    }
    else if (prod == 5) {
        for (int i = 0; i < 5; i++) {
            if(cont[i]<5){
                cont[i]++;
            }
        }
    }
}
else if (func == 2) {···
else if (func == 3) {···
```

Fig. 11 – Condicionais de função CoreMatrix.

Nas condicionais é definida qual função foi escolhida e qual lógica de contabilidade será implementada, caso o produto escolhido seja 5, que acontece quando o usuário seleciona Todos no segundo menu, a função escolhida no primeiro menu será aplicada a todos os produtos.

```
for(int i=0; i<5; i++){
    if(cont[i]>0){
        for(int j=0; j<cont[i]; j++){
            if(i==0){
                npSetLED(leds_matrix[i][j],30,0,0);
            }
            else if(i==1){···
            else if(i==2){···
            else if(i==3){···
            else if(i==4){···
        }
    }
}
npWrite(); // Imprime a atualização para os LEDs
```

Fig. 12 – Lógica de impressão dos LEDs na Matriz de LED.

Por fim, é realizada a verificação em cada uma das cinco colunas, caso o contador daquela coluna seja maior que zero, serão impressos os valores de índice de led_matrix de 0 até o valor presente no contador da coluna que está sendo analisada.

As condicionais dentro do for são usadas para criar diferenciação nas cores dos LEDs, indicando as colunas referentes a cada produto visualmente para o usuário.

5. Wifi Embarcatech

A função do arquivo wifi_embarcatech é criar a conexão entre a placa e o Wi-Fi, por enviar as informações presentes no vetor cont e também numa variável criada dentro dessa função chamada cont_total, que será utilizada para enviar a soma total de todos os valores do vetor cont.

Primeiramente, são definidas a estrutura para representação da conexão TCP, e uma variável chamada server_ip que armazenará o endereço IP do Thingspeak, por fim é chamado o vetor externo cont para que os valores possam ser enviados e manipulados no arquivo wifi_embarcatech.

```c
int wifi_embarcatech(uint flag_wifi_embarcatech) {
    if(flag_wifi_embarcatech==0){
        if (cyw43_arch_init()) {
            printf("Falha ao iniciar Wi-Fi\n");
            return 1;
        }
    }

    cyw43_arch_enable_sta_mode();
    printf("Conectando ao Wi-Fi...\n");

    if (cyw43_arch_wifi_connect_blocking(WIFI_SSID, WIFI_PASS, CYW43_AUTH_WPA2_MIXED_PSK)) {
        printf("Falha ao conectar ao Wi-Fi\n");
        return 1;
    }

    printf("Wi-Fi conectado!\n");

    dns_gethostbyname(THINGSPEAK_HOST, &server_ip, dns_callback, NULL);

    return 0;
}
```

Fig. 13– Função principal wifi_embarcatech.

Na função principal, a placa do Wi-Fi é iniciada e configurada para iniciar no Station Mode através da função cyw43_arch_enable_sta_mode(), permitindo a conexão a um roteador wi-fi. Então, as credenciais de conexão são passadas e o dispositivo tenta se conectar à rede indicada através do código cyw43_arch_wifi_connect_blocking(WIFI_SSID, WIFI_PASS, CYW43_AUTH_WPA2_MIXED_PSK), a execução do programa só continuará após a conexão ser estabelecida ou falhar.

Então é chamada a função dns_gethostbyname que passa o site api.thingspeak.com para ser transformado em um endereço ip pela função de Callback dns_callback, quando a resolução é concluída, a função de Callback consegue receber o endereço de IP, chama a função http_connected_callback que por sua vez é responsável por realizar a conexão TCP com o Thingspeak, por organizar a string da requisição que está sendo enviada, escrever essa api e por chamar a função de Callback que receberá a resposta se o pacote foi recebido ou não, caso a resposta seja nula a conexão TCP será fechada, caso haja resposta do Thingspeak o buffer será liberado.

6. Testes e validação

A forma como o teste final foi conduzido foi inserindo valores pré-determinados à variável global cont, isso foi feito para facilitar o preenchimento e também para que as funcionalidades de adição, subtração e reset fossem testadas de forma mais ágil, no código que será mostrado, a variável cont recebeu os valores de cinco unidades para o produto 1, quatro unidades para o produto 2, três unidades para o produto três, duas unidades para o produto 4 e uma unidade para o produto 5, resultando no seguinte vetor cont:

```c
int cont[5] = {5,4,3,2,1};
```

Fig. 14 – Configuração de testes da variável global cont.

Para que o padrão apareça na matriz de LED é necessário que alguma função e produto sejam selecionados pelo menos uma vez, para que o programa passe pela função CoreMatrix que é a responsável por imprimir os valores na matriz de LED, caso adicione-se um valor ao produto 1 não será alterada em nada a contagem já que o valor 5 é o limite do contador, dessa forma se chega resultado abaixo:

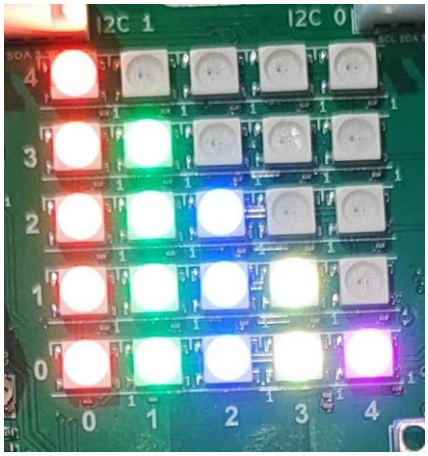

Fig. 15 – Teste da matriz de LED.

Da forma como foi pensada, a placa enviará os dados para a aplicação no Thingspeak a cada cinco minutos, porém para os testes esse tempo foi reduzido para um minuto visando estressar ao máximo o código, para isso foi alterada a linha de código que inicia o temporizador para 60000ms como mostrado abaixo:

```
while(true) {
    iniciar_temporizador(60000);
```

Fig. 16 – Alteração de testes para a inicialização do temporizador da função de Callback.

A placa ficou funcionando durante aproximadamente meia hora com o padrão de cont mostrado anteriormente para testar a sua conectividade, o teste foi realizado utilizando a rede da operadora TIM através de um hotspot de celular. Os resultados alcançados estão mostrados na figura a seguir:

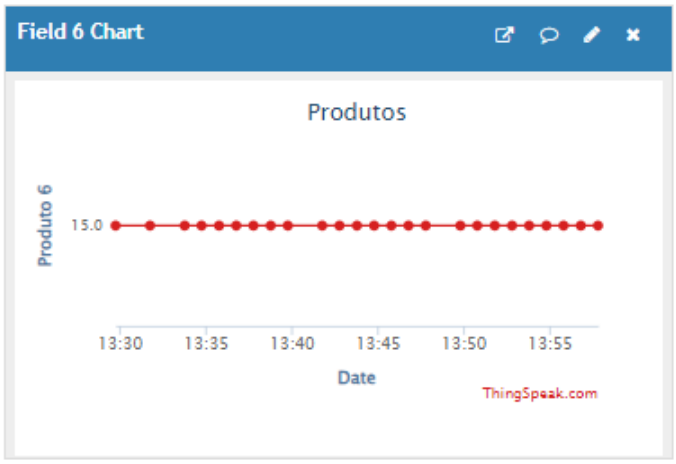

Fig. 17 – Teste de conectividade entre a placa e o Thingspeak.

A imagem mostra o valor enviado pela variável cont_total, que é a soma de todos os produtos produzidos até aquele momento na matriz, contando todos os LEDs que estão acesos, totalizando 15 produtos no valor mostrado no eixo Y do gráfico

O resultado acima foi considerado satisfatório já que em 29 tentativas de conexão, apenas 4 apresentaram falha, totalizando uma taxa de conectividade de aproximadamente 86%.

5 CONCLUSÃO

O projeto final apresentado está de acordo com o planejamento inicial da aplicação para este protótipo, todas os periféricos pensados foram de fato implementados e tiveram seu funcionamento testado por um período de tempo de pelo menos trinta minutos initerruptamente.

Os principais desafios foram a implementação da comunicação Wi-Fi com o serviço Thingspeak, a integração de todo o projeto, já que esse projeto foi uma mescla de vários códigos com os quais tive contato junto com códigos que produzi ao longo desse pouco mais de um mês com a placa, usando praticamente tudo que foi aprendido nesse período.

As principais limitações encontradas foram a falta de sensores que poderiam dar um corpo maior ao projeto, o próximo passo seria integrar a placa com sensores RFID ou então com códigos de barra que poderiam ser fixados no produto no início da linha de produção para serem lidos ao final, acabando com a necessidade de input manual do projeto e também faria com que implementá-lo se tornasse mais simples e eficiente. O Wi-Fi integrado na placa Raspberry Pi Pico não é robusto o suficiente para conexões com roteadores mais facilmente encontrados nas localidades. Resolver ambas as questões seria o próximo passo do projeto no futuro. Por fim, o código também pode ser melhorado para ficar mais limpo.

Criar uma solução útil, fácil de implementar e barata faria com que as fábricas da região se tornassem mais conscientes do próprio processo produtivo, diminuindo problemas logísticos e até mesmo na relação entre empregados e empresários.

Esse projeto busca ser o protótipo de uma solução real para um problema que ocorre hoje no chão das fábricas têxteis do Seridó Potiguar, tendo a capacidade de ser integrado a qualquer outro tipo de empreendimento que precise realizar a contagem precisa de diferentes produtos.

REFERÊNCIAS

[1]**BITDOGLAB**. *display_oled.c.* Github, 2025. Disponível em: https://github.com/BitDogLab/BitDogLab-C/blob/main/display_oled/display_oled.c. Acesso em: 13 fev. 2025.

[2]**upir**. *Arduino OLED Menu Tutorial (for begginers – Arduino UNO), 128x64px SSD1306 OLED screen, u8g.* Disponível em: https://youtu.be/HVHVkKt-ldc. Acesso em: 13 fev. 2025.

[3]**JIVAGO, Prof.** *Exibindo Imagens no Display OLED da BitDogLab.* Disponível em: https://www.youtube.com/watch?v=MLoararsJzA. Acesso em: 13 fev. 2025.

[4] **AZEVEDO DA COSTA, Júlio**. JoyMatrix. GitHub, 2024. Disponível em: https://github.com/Julioazvdo96/Embarcatech/tree/main/JoyLuz. Acesso em: 14 fev. 2025.

[5]**BITDOGLAB.** button_led_rgb.c. GitHub, 2025. Disponível em: https://github.com/BitDogLab/BitDogLab-C/blob/main/button_led_rgb/button_led_rgb.c. Acesso em: 13 fev. 2025.

[6]**BITDOGLAB**. *neopixel_pio.c*. GitHub, 2024. Disponível em: https://github.com/BitDogLab/BitDogLab-C/blob/main/neopixel_pio/neopixel_pio.c. Acesso em: 13 fev. 2025.

[7]**AZEVEDO DA COSTA, Júlio**. JoyMatrix. GitHub, 2024. Disponível em: https://github.com/Julioazvdo96/Embarcatech/tree/main/JoyMatrix. Acesso em: 14 fev. 2025.

[8]**WOKWI**. *Pioasm*. Disponível em: https://wokwi.com/tools/pioasm. Acesso em: 13 fev. 2025.

[9]**SOUZA, Iuri.** ThingSpeakTmp. GitHub, 2024. Disponível em: https://github.com/ProfIuriSouza/EmbarcaTech/tree/main/ThingSpeakTmp. Acesso em: 14 fev. 2025.