UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Escola de Engenharia
Departamento de Engenharia Eletrônica
Disciplina: Informática Industrial (ELT008) - Turma TECAD
Prof Luis T. S. Mendes

Trabalho Final - Etapa 2

Alunos: Guilherme de Souza Campos - 2020021263
Victória Tejada Siqueira Moreira - 2022421056

Belo Horizonte, 22 de junho de 2025

Introdução

Este trabalho apresenta o projeto e a implementação de um sistema de controle completo para um elevador, desenvolvido como avaliação final na disciplina de Informática Industrial (ELT008). O objetivo principal foi criar uma solução de automação segura e eficiente, utilizando uma arquitetura de software modular para simular o funcionamento de um elevador de quatro andares. O sistema foi dividido em três tarefas principais, projetadas para operar de forma contínua e isolada, permitindo sua execução simultânea sem interferências, garantindo respostas rápidas a comandos e eventos.

A arquitetura do sistema é composta pelas tarefas Main, Atende_Chamadas e Posicao_Elevador. A tarefa Main foi implementada por meio de um gráfico de funções sequenciais (SFC), pois a lógica de operação de um elevador segue uma sequência bem definida de etapas. Essa tarefa atua como o controlador principal, gerenciando os estados do elevador, como "Fechando Porta", "Movimentando Elevador", "Sobrecarga" e "Emergência". A tarefa Atende_Chamadas é responsável por toda a lógica de gerenciamento das chamadas, funcionando como o "cérebro" do sistema ao determinar a ordem de atendimentos, de forma análoga a um algoritmo patenteado por empresas do setor. Por fim, a tarefa Posicao_Elevador simula com precisão a física do elevador, controlando sua posição, velocidade e acionamento dos sensores de fim de curso.

O projeto também se destaca pelo uso de Functions Blocks (FB), que encapsulam funcionalidades específicas, como os mecanismos de abertura e fechamento de porta, para legibilidade e a manutenção do código. Para complementar o controle, foi desenvolvida uma interface gráfica que, embora não seja uma IHM convencional, serve como uma representação abstrata e funcional do elevador. Essa interface permite a visualização clara do comportamento do sistema e foi uma ferramenta essencial para a realização de testes e validação da lógica implementada.

Programa do CLP

Tarefas

O projeto está dividido em três tarefas principais: Main, Atende_Chamadas e Posicao_elevador, sendo cada uma responsável por uma função específica e executada de forma independente.

Cada tarefa é projetada para rodar continuamente de forma isolada, permitindo que o CODESYS execute as funções simultaneamente, sem que uma tarefa interfira na execução da outra. Dessa maneira, o sistema é capaz de especializar cada tarefa em uma função específica e rodar continuamente sem perda de desempenho, garantindo a resposta imediata a qualquer comando ou evento, além de facilitar a manutenção e o diagnóstico, já que problemas em uma tarefa não afetam diretamente as outras.

De maneira resumida, a tarefa Main é um Sequential function chart (SFC) que orquestra o funcionamento do elevador através de diferentes estados, sendo esses estados, por exemplo, o de fechar ou abrir a porta, esperar chamadas ou movimentar o elevador, por exemplo. Já a tarefa Atende_Chamada contém toda a lógica de atendimento do elevador. Fazendo uma analogia com o que foi dito no Enunciado do Trabalho, há empresas que patenteiam seus algoritmos de atendimento ou, até mesmo, os mantém como segredo industrial. De maneira análoga, esse é o nosso algoritmo de atendimento. Por fim, temos a terceira e última tarefa, a de Posicao_Elevador. Como o nome sugere, ela se assegura de simular a posição do elevador, bem como sua velocidade, fins de curso acionados, etc. Nos aprofundaremos em cada tarefa a seguir:

Main

A Main do projeto funciona como o mais alto nível do Projeto Elevador. Ela é responsável por percorrer todos os possíveis estados que o elevador se encontra, guiando o estado atual para o próximo conforme disparos de transições.

Os estados que a compõem são:

- Init;
- Fechando_Porta;
- Mov_Elevador;
- Abrindo_Porta;
- Espera_Chamadas;
- Sobrecarga;
- EMG;
- EMG_Desce;
- EMG_Terreo.

Tivemos o cuidado de dar nomes significados, que descrevem intuitivamente o que fazem cada estado. Todos esses estados executam funções quando estão ativas, por exemplo, o estado Sobrecarga_active executa o seguinte código em ST:

Figura: Action Sobrecarga_active

```
1  // Elevador não se movimenta
2  Posicao_Elevador.Movimenta := FALSE;
3
4  // Acende LED de sobrecarga
5  LD_PESO := TRUE;
```

Os únicos steps que executam uma *action association* são o Fechando_Porta e o Abrindo_Porta, os quais executam os algoritmos Fecha_Porta e Abre_Porta, respectivamente.

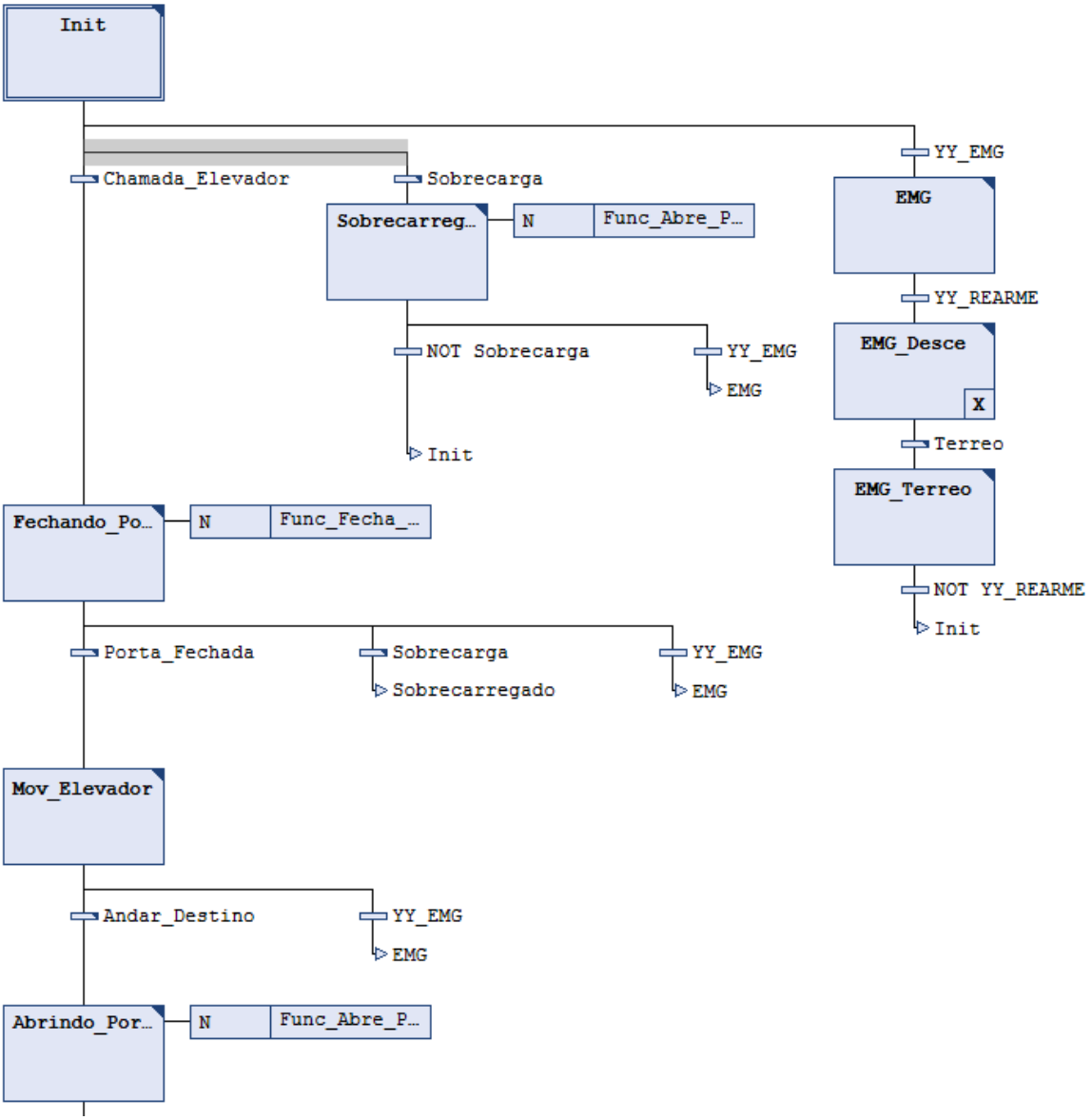
As transições entre estados são ou objetos transition ou únicos sensores autuados. Tivemos esse cuidado para garantir uma lógica SFC mais compacta e limpa aos olhos. Isso quer dizer que, ou as transições são ativadas por expressões booleanas relativamente complexas, como a Andar_Destino, ou sinais simples, como o YY_EMG para o estado de

emergência. Além disso, tivemos o cuidado de manter todas as transições mutuamente exclusivas, sempre adicionando condições como, por exemplo `NOT YY_EMG AND NOT Sobrecarga`.

Figura: Arquivo de transição Andar_Destino

```
// Elevador chegou no andar de destino
(
(Atende_Chamadas.AndarDestino = 0 AND YY_T) AND (NOT YY_EMG) OR  // Chegou no Térreo
(Atende_Chamadas.AndarDestino = 1 AND YY_1) AND (NOT YY_EMG) OR  // Chegou no 1° Andar
(Atende_Chamadas.AndarDestino = 2 AND YY_2) AND (NOT YY_EMG) OR  // Chegou no 2° Andar
(Atende_Chamadas.AndarDestino = 3 AND YY_3) AND (NOT YY_EMG)     // Chegou no 3° Andar
) AND NOT YY_EMG
```

Figura: Program Main



É digno de nota destacar a escolha do SFC para compor a Tarefa Main do nosso projeto. O escolhemos pois a lógica de funcionamento de um elevador segue naturalmente uma sequência de etapas bem definidas, como abrir portas, aguardar passageiros, fechar portas, movimentar, parar no andar e reiniciar o ciclo. O SFC permite representar essas etapas de

forma clara, ordenada e visual, facilitando tanto o desenvolvimento quanto o entendimento do comportamento do sistema. Essa abordagem sequencial também evita conflitos de estados e torna a lógica mais segura e controlada.

Posição_Elevador

A tarefa Posicao_Elevador gerencia o movimento do elevador entre os andares. Ela controla a velocidade de deslocamento e o sentido do movimento (subindo ou descendo), com base nas chamadas de passageiros e na última posição do elevador. Além disso, ela cuida dos fins de curso para garantir que o elevador mostre estar parado corretamente em cada andar, e também atualiza os mostradores LCD com a posição atual do elevador. Essas duas últimas ações descritas são feitas através dos Functions Blocks Aciona_Fins_de_Curso e Atualiza_Mostradores_LCD, as quais apresentam uma pequena documentação nos arquivos do Codesys. Convidamos que acesse os arquivos para entender exatamente sua lógica e ver a documentação do código lá feita.

Figura: Final de Posicao_Elevador, onde as funções de fim de curso e LCDs são chamadas

```
50  // Aciona ou desativa os fins de curso de cada andar
51  Fins_de_Curso(Posicao := ZIC_1);
52
53  // Atualiza os Mostradores LDC
54  LCDs();
```

Além de atualizar os Fins de Curso e os displays, a tarefa Posicao_Elevador, obviamente, se encarrega de ditar a posição do elevador. Para isso, as seguintes açõe são feitas:

- **Controle de Direção:** Determina se o elevador deve subir ou descer, com base na direção da última chamada recebida. A variável direção é: `Atende_Chamadas.Direcao`, do tipo into, cuja representação é 1 para subir e -1 para descer.

- **Controle de Velocidade**: Controla a velocidade do elevador com base na distância do andar de partida e chegada. Existem duas velocidades: Velocidade_Alta (30 m/min) e Velocidade_Baixa (10 m/min). O elevador acelera entre os andares (alta velocidade) e desacelera ao se aproximar do andar de partida ou destino (baixa velocidade).

- **Movimento**: Uma vez obtida a direção, e a velocidade, essa parte do código usa a VAR_INPUT `Movimenta : BOOL` para ditar a posição do elevador. Se `Movimenta =` TRUE, o elevador se desloca da seguinte forma:
  Subindo: ZIC_1 aumenta;
  Descendo: ZIC_1 diminui.
  Movimenta é diretamente setado como TRUE ou FALSE pelos estados da Tarefa Main. Se Movimenta = FALSE, os motores são desligados.

- **Atualização dos Sensores:** A função Fins_de_Curso() ativa sensores conforme a posição e a função LCDs() atualiza os mostradores de posição.

Essa Tarefa é fundamental para o movimento preciso e controlado do elevador, além de garantir que o sistema de display esteja sempre atualizado com a posição real do elevador

Papeis de desempenho

Componente	**Papel**
Movimenta	Controla se o elevador deve se mover ou não
UltimoAndar	Serve de referência para ajustar a velocidade na partida
ZIC_1	Representa a posição contínua atual do elevador em metros
VEL_MOTOR	Define a saída de velocidade para os motores
Velocidade	Armazena o valor real da velocidade aplicada
M_SOBE, M_DESCE	Indicadores de direção do movimento para motores e lógicas externas
Fins_de_Curso	Aciona sensores virtuais conforme a posição
LCDs	Atualiza a interface com a posição do elevador

Atende_Chamadas

A Tarefa Atende_Chamadas gerencia as chamadas feitas pelo usuário para os diferentes andares. Ela lida com as solicitações de passageiros tanto no interno quanto no externo do elevador, priorizando as chamadas de acordo com a direção do movimento do elevador. Quando o elevador chega ao destino de uma chamada, ele a registra como atendida e atualiza a direção do movimento. Veja a seguir uma descrição detalhada do funcionamento do Programa, com base numa sequência de passos

1. **Verificação de Emergência:**
   O programa começa verificando se o elevador não está em emergência (F_EMG) e se a função Zera_Chamadas não foi acionada. Caso contrário, o sistema pode proceder com as chamadas.

2. **Verificação de Chamadas de Andar:**
   A lógica verifica se há chamadas nos andares internos e externos:
   YY_IBT e YY_EBT referem-se aos botões internos e externos do elevador para o térreo. YY_IB1 a YY_IB3 e YY_EB1 a YY_EB3 são os botões de chamada para os andares 1, 2 e 3. Se o botão de chamada correspondente for pressionado e o elevador *não estiver já no andar chamado*, o programa marca o andar como chamado e define a variável NovaChamada como TRUE.

3. **Adicionando Chamadas:**
   Quando uma nova chamada é detectada (NovaChamada = TRUE), o sistema verifica se já há uma chamada registrada para o andar. Se já houver, não faz nada. Caso contrário, ele registra a chamada no vetor Chamadas. Cada posição do vetor

corresponde a um andar específico (0 para térreo, 1 para 1º andar, etc.).

4. **Contagem de Chamadas:**
   O sistema então conta quantas chamadas existem, somando 1 para cada posição do vetor Chamadas que está marcada como 1 (indicando que há uma chamada para aquele andar).

5. **Atendimento das Chamadas:**
   Se houver alguma chamada para ser atendida (NumChamadas > 0), o programa verifica a direção do elevador (se está subindo ou descendo) e determina qual será o próximo andar a ser atendido.

   Caso de Subida: O elevador atende as chamadas dos andares em ordem crescente:
   - Se o elevador está no térreo e há chamada para o 1º andar, o elevador vai para o 1º andar, e assim por diante.
   - Caso o elevador esteja em um andar mais baixo, ele atende as chamadas dos andares acima, se houver.
   - Se não houver mais chamadas para andares superiores, o elevador inverte a direção para descer.

   Caso de Descida: O elevador atende as chamadas dos andares em ordem decrescente:
   - Se o elevador está no 3º andar e há chamada para o 2º andar, ele vai para o 2º andar, e assim por diante.
   - Caso o elevador esteja em um andar mais alto, ele atende as chamadas dos andares abaixo, se houver.
   - Se não houver mais chamadas para andares inferiores, o elevador inverte a direção para subir.

   Queremos destacar que essa parte em específico do atendimento de chamadas foi realizada “na força bruta”. Para evitar o uso de laços `FOR` no Atende_Chamadas, consideramos cada possível possível do elevador, cada chamada possível e as duas direções possível. Isso foi viável pois o elevador atendia somente 4 andares. Um número maior de andares colocaria essa abordagem em cheque.

   Apesar de não ser a solução mais elegante, ela mostrou atender todos os testes que fizemos.

Figura: Algoritmo de atendimento feito considerando cada caso individualmente

```
59          // Caso elevador SUBINDO
60          IF Direcao = 1 THEN
61              IF PosicaoAtual = 0 OR UltimoAndar = 0 THEN  // Elevador no TERREO
62                  IF Chamadas[1] = 1 THEN // Chamada no 1° Andar
63                      AndarDestino := 1;
64                  ELSIF Chamadas[2] = 1 THEN  // Chamada no 2° Andar
65                      AndarDestino := 2;
66                  ELSIF Chamadas[3] = 1 THEN  // Chamada no 3° Andar
67                      AndarDestino := 3;
68                  END_IF
69              ELSIF PosicaoAtual = 1 OR UltimoAndar = 1 THEN  // Elevador no 1° ANDAR
70                  IF Chamadas[2] = 1 THEN  // Chamada no 2° Andar
71                      AndarDestino := 2;
72                  ELSIF Chamadas[3] = 1 THEN  // Chamada no 3° Andar
73                      AndarDestino := 3;
74                  ELSE
75                      Direcao := -1;  // Muda direcao do elevador
76                  END_IF
77              ELSIF PosicaoAtual = 2 OR UltimoAndar = 2 THEN  // Elevador no 2° ANDAR
78                  IF Chamadas[3] = 1 THEN  // Chamada no 3° Andar
79                      AndarDestino := 3;
80                  ELSE
81                      Direcao := -1;  // Muda direcao do elevador
82                  END_IF
83              ELSIF PosicaoAtual = 3 OR UltimoAndar = 3 THEN  // Elevador no 3° ANDAR
84                  Direcao := -1;      // Muda direcao do elevador
85              END_IF
86          END_IF
87
88          // Caso elevador DESCENDO
89          IF Direcao = -1 THEN
90              IF PosicaoAtual = 3 OR UltimoAndar = 3 THEN  // Elevador no 3° Andar
91                  IF Chamadas[2] = 1 THEN // Chamada no 2° Andar
92                      AndarDestino := 2;
```

6. **Atualização da Posição:**
   A posição atual do elevador é atualizada conforme os botões de chamada são pressionados:
   - YY_T para o térreo (Andar 0)
   - YY_1 para o 1º andar
   - YY_2 para o 2º andar
   - YY_3 para o 3º andar

   Se nenhum botão for pressionado, a posição do elevador é definida como -1, indicando que o elevador está em trânsito.

7. **Chegada ao Destino:**
   Quando o elevador chega ao andar destino, a chamada para aquele andar é desfeita (o valor no vetor Chamadas é definido como 0).

8. **Atualização dos LEDs (Indicadores de Chamada):**
   Com base nas chamadas registradas, os LEDs correspondentes a cada andar são acesos ou apagados:
   - LD_IT, LD_I1, LD_I2, LD_I3: LEDs internos para indicar chamadas no térreo

e andares 1, 2 e 3.
- LD_ET, LD_E1, LD_E2, LD_E3: LEDs externos para indicar chamadas nos mesmos andares.

9. **Modo de Emergência (F_EMG):**
   - Se o modo de emergência for ativado (F_EMG), o elevador será forçado a ir para o térreo (andar 0), e as chamadas serão zeradas, exceto a do térreo.

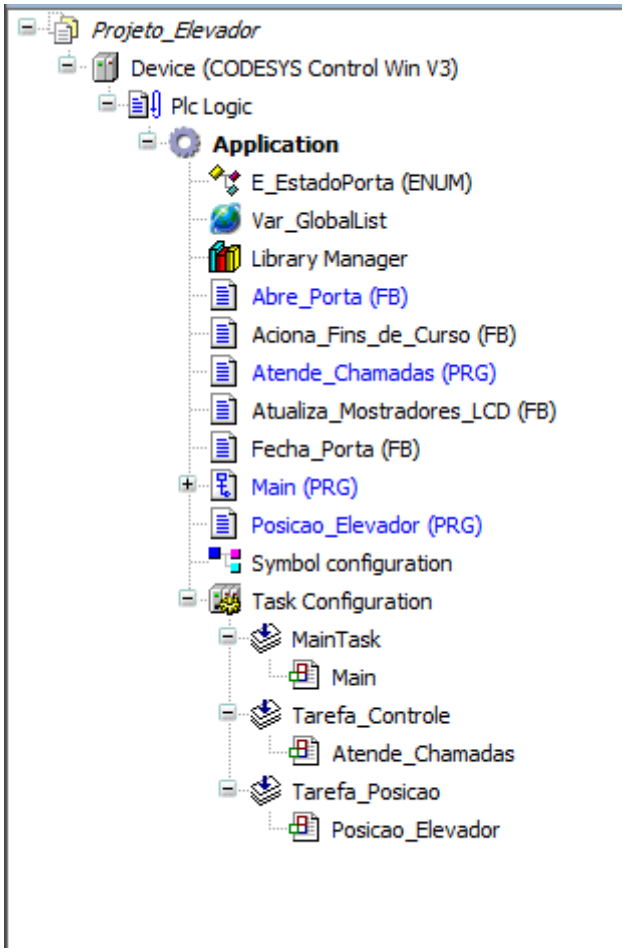
10. **Zerar Chamadas (Zera_Chamadas):**
    - Se a função Zera_Chamadas for acionada, o sistema limpa todas as chamadas registradas e reinicia a posição do elevador para o térreo, reiniciando a direção para subir. Zera_Chamadas é chamada como uma *exit action para o estado EMG_Desce*

Sem sombra de dúvidas, o Programa Atende_Chamadas foi o mais difícil e trabalhoso de ser desenvolvido. Foi necessário tratar um algoritmo naturalmente complexo, o de ordenamento de escolha de chamadas de elevador, e durante o seu desenvolvimento surgiram muitos bugs, desde erros de sintaxe os quais 'não pareciam estar errados', até mesmo erros lógicos decorrentes de um desenvolvimento incorreto do algoritmo. Discorreremos mais adiante no trabalho sobre esses e outros bugs. Por fim, apesar de ter sido nessa tarefa onde gastamos a maior quantidade de horas de engenharia do nosso trabalho, acreditamos que não foi possível cobrir todos os casos possíveis de testes

Escolhas de Projeto

No desenvolvimento do sistema de controle do elevador, as escolhas de projeto seguiram uma abordagem modular e orientada a funções específicas, com a utilização extensiva de Function Blocks (FB). Cada FBencapsula uma parte bem definida da lógica de controle, o que melhora a organização, a legibilidade e a capacidade de manutenção do código. A separação clara entre as diferentes funcionalidades do sistema possibilita testes mais eficazes e uma depuração facilitada em caso de falhas.

Figura: Árvore de projeto contendo as diversas FBs



Como exemplo representativo, pode-se destacar o FB Fecha_Porta, que implementa uma máquina de estados responsável por controlar o processo completo de fechamento da porta do elevador. Esse bloco inicia no estado PORTA_ABERTA_ESPERANDO, onde aguarda a ausência de passageiros por um intervalo de 5 segundos antes de iniciar o fechamento. Caso o fechamento seja iniciado e o sensor de esmagamento (YY_ESMAG) seja ativado, o estado é imediatamente alterado para ESMAGAMENTO_DETECTADO, forçando a reabertura da porta para garantir a segurança. Após a abertura, o bloco retorna ao estado inicial. Se não houver obstruções, o fechamento continua até que o tempo definido para o motor expire, momento em que o sistema assume que a porta está completamente fechada, atualizando os sinais correspondentes.

Esse comportamento estruturado em estados permite um controle claro, previsível e seguro da porta, essencial em aplicações reais. Além disso, vale ressaltar que todos os FB utilizados no trabalho foram devidamente documentados, contendo descrições das entradas, saídas e da lógica implementada.

Figura: Declaração da FB Fecha_Porta e sua documentação

```
(*
    Function Block: Fecha_Porta

    Descrição:

    Este Function Block implementa uma máquina de estados  para gerenciar o fechamento
    da porta do elevador. Os estados são:
    1. Espera de passageiros;
    2. Fechamento da porta;
    3. Detecção de  esmagamento com reabertura automática;
    4. Porta fechada.

    O estado da porta (aberta, fechada ou em movimento) é alterado em cada estado
*)

FUNCTION_BLOCK Fecha_Porta
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    // Possiveis estados da porta do elevador
    estadoPorta : E_EstadoPorta := E_EstadoPorta.PORTA_ABERTA_ESPERANDO;

    START : BOOL := TRUE;  // Variavel de controle usada para começar sempre do Estado PORTA_ABERTA_ESPERANDO

    TMR_Espera_Passageiro : TON; // Temporizador de 5s para aguardar passageiro
    TMR_Acao_Motor        : TON; // Temporizador de 3s para manter motores da porta acionados
END_VAR
```

Para além da divisão em FBs, tivemos outras escolhas de projeto que julgamos dignas de nota. Como dito anteriormente, tivemos o cuidado de manter todas as transições mutuamente exclusivas. Essa boa prática garante que as transições não terão comportamentos inesperados, ou mesmo ao acaso.

Além disso, garantimos que cada step do nosso SFC realize uma função bem definida. Por exemplo, o step Mov_Elevador seta o input Posição_Elevador.Movimenta como TRUE, logo é nesse step em que o elevador se move. Já os steps Fechando_Porta e Abrindo_Porta executam actions de fechar ou abrir as portas do elevador. Por fim, temos o step de Sobrecarga e uma série de 3 steps de Emergência: para o elevador, voltar para o térreo e esperar o rearme.

Finalmente, decidimos não alterar muito o jogo de variáveis globais a nós fornecidos, preferimos lá manter apenas as variáveis de interação com a IHM, que será discutida adiante. Isso mostrou-se acertado, pois um número reduzido de variáveis globais mantém a lógica de cada Step ou FB local, facilitando o debug e entendimento do código. A única exceção a isso foi a flag `F_EMG` que declaramos. Como a lógica de emergência é superior a todos os estados, faz sentido indicar que o programa está em um estado de emergência.

Interface Homem Máquina (IHM)

O papel de uma IHM (Interface Homem-Máquina) é facilitar a interação entre operadores humanos e sistemas de automação industrial. Ela atua como uma ponte, permitindo que os operadores monitorem, controlem e otimizem processos, tomando decisões com base em informações claras e acessíveis fornecidas pela IHM.

IHM é a sigla para Interface Homem Máquina. Como o próprio nome diz, o recurso atua como mediador da interação entre um operador e um sistema de automação. Na indústria,

ela geralmente é utilizada em linhas de produção e em máquinas de propósitos variados. Nestes casos, elas são equipadas com receitas, registros de eventos, sistema de vídeo, alarmes, entre outras informações que demandam acesso imediato pelo operador.

Em um sistema de controle típico, existe um CLP central responsável pela coleta de dados provenientes de sensores e demais dispositivos de campo. A função da IHM é acessar o CLP, ler os dados coletados e projetá-los de forma visual para que o operador do sistema possa tomar uma decisão com base na situação real da máquina ou do processo. Após tomar a decisão, o operador pode realizar um comando na IHM para que ela faça o caminho contrário, levando informações ao CLP para que ele, então, altere algum aspecto do sistema de controle.

O projeto do elevador, por definição, não se caracteriza como uma interface homem-máquina (IHM) convencional. Em vez disso, desenvolvemos uma representação gráfica abstrata do elevador, com o intuito de ilustrar o funcionamento do sistema. Essa representação foi projetada para demonstrar o comportamento do elevador, sem necessariamente seguir os padrões de uma IHM tradicional

Interface do Projeto Elevador

Figura: Interface Trabalho

A interface desenvolvida no ambiente AVEVA apresenta as principais informações operacionais e de segurança do sistema de elevador, dividida em quatro seções principais: **Parte Interna do Elevador**, **Porta do Elevador**, **Parte Externa**, e **Painel do Operador**.

Painel Interno do Elevador

Esta área simula os controles acessíveis pelos passageiros no interior da cabine do elevador. Os seguintes elementos estão presentes:

- **Botões de Andar:** Quatro botões numerados para seleção do destino, sendo eles: T (Térreo), 1, 2, 3 (andares).
- **Botão Emergência:** botão dedicado a situações de emergência, que ativa o protocolo de segurança e retorna o elevador ao térreo.
- **Display de Peso:** Um rótulo (label) serve como input para o peso atual dentro da cabine, em quilogramas (kg), para simular o peso da cabine
- **Andar atual:** Display indicando o andar atual

Porta do Elevador

Representação do status atual da porta da cabine, com indicação visual e sensores associados:

- **Estado da Porta:** Indicador visual informa se a porta está aberta ou fechada.
- **Sensor de Laser:** Detecta objetos ou pessoas na região de fechamento da porta.
- **Sensor de Esmagamento:** Segurança adicional que previne o fechamento da porta caso detecte qualquer tipo de resistência.

Painel Externo do Elevador (Andares)

Instalado em cada andar, representa os controles e informações acessíveis a quem está fora da cabine:

- **Indicador de Andar Atual:** Display que mostra o andar no qual o elevador se encontra no momento.
- **Botões de Chamada:** Cada andar possui um botão para chamar o elevador, permitindo que o usuário solicite a parada da cabine.
- **Alavanca de Emergência:** Dispositivo de segurança externo, utilizado para sinalizar situações de emergência e forçar o retorno da cabine ao térreo, se necessário.

Painel do Operador

Área dedicada à supervisão técnica e operação do sistema por um operador fictício.

- **Indicador de Velocidade:** Mostra em tempo real a velocidade do elevador (em m/min).
- **Sinalizador de Emergência (Buzina):** Lâmpada representando a buzina, que dispara quando alguém aciona o botão de emergência
- **Motores:** Indica os estados dos 4 motores do projeto, isto é, se estão ativados ou nao.
- **Gráfico de Monitoramento:** Na parte inferior da interface, há um gráfico dinâmico indicando a posição do elevador.

Testes Realizados

Seguimos um roteiro de teste com base fomos desenvolvendo cada módulo do trabalho. Isto é, testamos os estados e suas transições no desenvolvimento da Main; a posição vertical e a velocidade do elevador em Posicação_Elevador, o acionamento dos motores e sensores em Fecha_Porta, etc. Tais testes individuais garantiram que pudéssemos ter módulos individuais bem comportados, de como que, ao juntá-los na versão final do trabalho, não houve muito problema, pois cada FB, Action ou Tarefa tinha sido individualmente validada.

Como era de se esperar, durante os testes individuais, vários problemas ocorreram, e destacamos aqui aqueles que 'deram mais trabalho'.

Bugs encontrados

Velocidade no andar

Não foi trivial controlar a velocidade do elevador. A lógica que utilizamos, à princípio, fez com que o elevador sempre saísse do andar de partida em velocidade baixa e sem seguida entrasse em velocidade alta - até então correto. Todavia, o elevador *sempre* entrava novamente em velocidade baixa *de cada andar*, em vez de entrar em velocidade baixa *apenas próximo do andar de destino***.** A solução para isso foi sempre chegar a variável Atende_Chamadas.AndarDestino, quebrando um pouco o nosso ideal de módulos bem desacoplados.

Figura: Solução encontrada para a velocidade

```
// Controla a velocidade do elevador
IF Sobe THEN
    IF ZIC_1 < (UltimoAndar * 4 + 1) OR              // Elevador há menos de 1 metro do andar de partida
       ZIC_1 > (Atende_Chamadas.AndarDestino * 4 - 1)  // Elevador há menos de 1 metro do andar de chegada
    THEN
        Velocidade := Velocidade_Baixa;
        VEL_MOTOR  := 10;
    ELSE
        Velocidade := Velocidade_Alta;
        VEL_MOTOR  := 30;
    END_IF
ELSIF Desce THEN
    IF ZIC_1 > (UltimoAndar * 4 - 1) OR              // Elevador há menos de 1 metro do andar de partida
       ZIC_1 < (Atende_Chamadas.AndarDestino * 4 + 1)  // Elevador há menos de 1 metro do andar de chegada
    THEN
        Velocidade := Velocidade_Baixa;
        VEL_MOTOR  := 10;
    ELSE
        Velocidade := Velocidade_Alta;
        VEL_MOTOR  := 30;
    END_IF
END_IF
```

Problemas no Abre_Porta

Um bug muito simples, aparentemente muito comum, mas extremamente contra intuitivo, foi o funcionamento do Timer no FB Abre_Porta. Será simples entender o que aconteceu com base na versão antiga e na versão nova dessa FB:

Figura: Versão antiga e errada de Abre_Porta

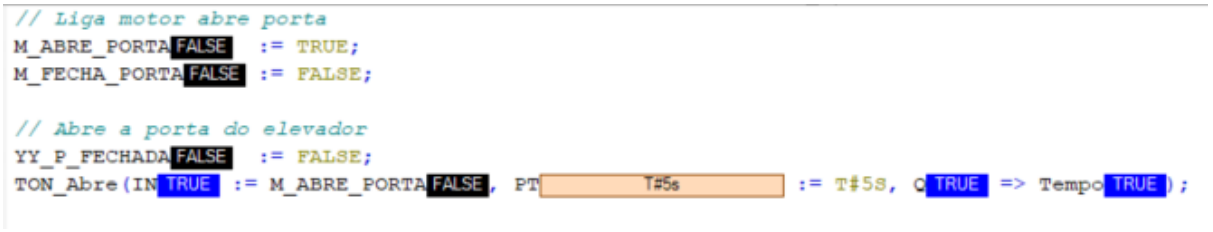


Figura: Versão nova e funcional de Abre_Porta

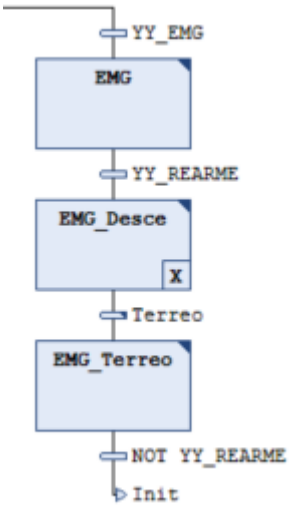
```
// Garante que a porta não está fechada
M_FECHA_PORTA := FALSE;
YY_P_FECHADA  := FALSE;

// Logica para temporizador do motor abre porta
IF NOT YY_P_ABERTA THEN
    M_ABRE_PORTA  := TRUE;
    TON_Abre(IN := TRUE, PT := T#3S);
ELSE
    M_ABRE_PORTA  := FALSE;
    TON_Abre(IN := FALSE, PT := T#3S);
END_IF;

// Abre a porta
IF TON_Abre.Q THEN
    YY_P_ABERTA := TRUE;
END_IF;
```

Basicamente, a versão antiga *não resetava o timer* a cada vez que o step da SFC era alterado, e isso gerava erros, fazendo com que o nosso projeto *mantesse o motor abre_porta acionado apenas uma vez*. Depois dessa primeira vez, a porta sempre era aberta, sem acionamento do motor.

Estado Emergência

A implementação do estado emergência também não foi trivial. Tivemos problemas em garantir que o elevador voltasse para o térreo. Depois, que ele *parasse* no térreo. Depois, que as chamadas fossem todas zeradas. Isso foi resolvido com um passo a passo de três Steps:

Figura: Sequência de steps de emergência



O primeiro Step aciona a buzina, seta a flag de emergência e trava o elevador. Isso foi intuitivo a princípio. O que não foi intuitivo foram os 2 próximos estados. Implementamos o EMG_Desce para usar o já feito Programa Atende_Chamadas, fazendo uma chamada para o térreo. Após isso foi preciso para o elevador no Térreo, através da transição Térreo, cujo estado de saída de EMG_Desce zerava todas as chamadas de Atende_Chamadas. Após isso, a chave de rearme voltava a Main para o step init

Conclusão

O desenvolvimento deste projeto permitiu a implementação bem-sucedida de um sistema de controle de elevador, validando a eficácia da abordagem modular adotada. A estruturação do controle em tarefas distintas, a Main, Atende_Chamadas e Posicao_Elevador, e o uso de Function Blocks (FB) para encapsular lógicas específicas, como o controle de portas, provaram ser cruciais para a organização, legibilidade e manutenção do código. Essa arquitetura não apenas facilitou o desenvolvimento e a depuração, mas também garantiu um funcionamento mais seguro e previsível. O maior desafio do projeto foi a criação do algoritmo Atende_Chamadas, que, devido à sua complexidade, foi implementado por meio de uma análise de todos os cenários possíveis, uma solução que se mostrou funcional para o escopo de quatro andares. Embora a interface gráfica desenvolvida não seja uma IHM convencional, ela foi fundamental para a visualização e teste do sistema. Ao final, o trabalho representou um valioso exercício de engenharia, reforçando a importância do planejamento estruturado e dos testes contínuos no desenvolvimento de sistemas de automação.