FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Upgrade do Simulador da Linha de Produção Flexível do Laboratório de Automação

Sebastião Cunha Reis

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Paulo Portugal

28 de Fevereiro de 2020

Resumo

A linha de produção flexível existente no laboratório de automação do DEEC é constituída por vários equipamentos (tapetes, máquinas...) e por peças que são processadas de diversas formas. Existe um simulador desta linha de produção que tem prestado um contributo muito importante no suporte ao desenvolvimento de aplicações de controlo da linha de produção por parte de estudantes do MIEEC.

O simulador ainda não dispõe de um módulo que permita simular corretamente o armazém automático da linha de produção flexível. Por esse motivo, e com o intuito de criar propostas de projeto mais ambiciosas, que explorem novos conceitos, surgiu o interesse de fazer um *upgrade* ao simulador.

Foi então desenvolvido um módulo que simula o funcionamento do armazém automático. Implementou-se também o conceito de código de barras na simulação das peças, o conceito de avaria e reparação de equipamentos e um painel com botões e indicadores luminosos que permitem ao utilizador interagir com o simulador.

Por fim, foram ainda implementadas novas funcionalidades que visam automatizar o processo de avaliação de aplicações de controlo da linha de produção do simulador. Permitem agir sobre o simulador (e.g. inserir peças, gerar avarias em equipamentos) e obter informação sobre os equipamentos e peças do simulador. Deste modo, podem ser desenvolvidos algoritmos que, através destas funcionalidades, permitam verificar um conjunto de objetivos predefinidos, avaliando o correto funcionamento de aplicações de controlo do simulador.

Agradecimentos

Queria agradecer ao meu orientador, professor Paulo Portugal, e à minha família e amigos pelo apoio que me deram ao longo de todo o processo de desenvolvimento desta dissertação.

Sebastião Cunha Reis

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

MIEEC	Mestrado Integrado em Engenharia Electrotécnica e de Computadores
DEEC	Departamento de Engenharia Electrotécnica e de Computadores
FEUP	Faculdade de Engenharia da Universidade do Porto
LED	Light-emitting diode
ID	Identifier
PLC	Programmable Logic Controller
IEEE	Instituto de Engenheiros Eletrotécnicos e Eletrónicos
TCP	Transmission Control Protocol
IP	Internet Protocol
PDU	Protocol Data Unit
ADU	Application Data Unit
MBAP	Modbus Application Protocol
SFC	Sequential Flow Chart
ST	Structured Text
UML	Unified Modeling Language

Capítulo 1

Introdução

Este documento descreve detalhadamente a dissertação "Upgrade do simulador do laboratório de automação". Neste primeiro capítulo é ilustrado, inicialmente, o contexto e motivação desta dissertação e os objetivos que foram propostos. Por fim, é descrita a organização do documento.

1.1 Contexto e Motivação

A linha de produção flexível existente no laboratório de automação do DEEC é constituída por vários equipamentos (tapetes, máquinas...) e por peças que são processadas de diversas formas.

Existe um simulador desta linha de produção que tem prestado um contributo muito importante no suporte ao desenvolvimento de aplicações de controlo por parte dos estudantes em várias unidades curriculares do MIEEC. É utilizado em unidades curriculares como Automação e Informática Industrial. Em Automação é usado apenas para que os estudantes possam fazer testes intermédios aos programas de controlo da linha de produção antes de testarem e serem avaliados na linha de produção flexível do laboratório. Em Informática Industrial, apenas se usa o simulador, tanto nos testes como na apresentação final do projecto, que visa também controlar uma linha de produção.

Apesar deste simulador representar corretamente quase todas as componentes da linha de produção flexível, pode ainda ser melhorado, não só de forma a que consiga simular todos os módulos da linha de produção flexível e seus componentes mas também para que disponha de novas funcionalidades que permitam a criação de propostas de projeto mais ambiciosas, que explorem novos conceitos.

O simulador atual não reproduz corretamente o funcionamento da célula do armazém automático da linha de produção flexível. Esta célula dispõe de uma unidade de transporte que permite inserir e retirar peças de um armazém para a linha de produção. No simulador não existe qualquer unidade de transporte e as peças são introduzidas de forma automática na linha de produção.

Uma outra limitação reside no facto de todas as avaliações de programas de controlo do simulador serem feitas de forma manual. Os docentes têm de acompanhar atentamente todos os processos para garantir que as operações são realizadas de forma correta. Desta forma, seria muito

útil desenvolver um conjunto de funcionalidades que permitissem interagir com o simulador de forma automática (e.g. inserir peças, avariar equipamentos) e verificar o estado dos equipamentos e peças da linha de produção. Deste modo, seria possível desenvolver algoritmos de verificação e avaliação automática dos programas de controlo do simulador.

1.2 Objetivos

Os objetivos definidos para esta dissertação são os seguintes:

a) Desenvolvimento do módulo de simulação do armazém automático;

b) Integração do conceito de código de barras na simulação das peças;

c) Desenvolvimento do conceito de avaria e reparação de equipamentos;

d) Integração de um painel com botões e indicadores luminosos na interface do simulador;

e) Desenvolvimento de um servidor Modbus que disponha de funcionalidades que permitam agir sobre a linha de produção do simulador (e.g. inserir e remover peças) e obter informação sobre o seu estado (equipamentos e peças), permitindo criar algoritmos de avaliação automática de programas de controlo do simulador.

1.3 Organização do documento

Este documento utiliza a formatação que está disponível na página web de dissertações da FEUP para *Latex*.

Para além deste capítulo introdutório, esta dissertação contém mais quatro capítulos. O capítulo 2 descreve a linha de produção flexível do laboratório e a versão anterior do seu simulador. No capítulo 3 é feita uma proposta de solução, que inclui uma análise dos requisitos e a uma descrição da sua arquitetura. O capítulo 4 descreve tudo o que foi implementado neste dissertação, tendo sempre em vista o cumprimento dos requisitos propostos, e os testes que foram realizados para validar o seu correto funcionamento. Por fim, no capítulo 5 são tiradas conclusões sobre o cumprimento dos objetivos propostos e é feita uma descrição do que poderá ser ainda implementado no futuro, como continuação desta dissertação.

Capítulo 2

Simulador da Fábrica - versão anterior

Neste capítulo faz-se, inicialmente, uma descrição da linha de produção flexível que se encontra no laboratório de Automação. De seguida, descreve-se de forma detalhada a versão anterior do simulador desta linha de produção. É feita ainda uma descrição da sua arquitetura e dos equipamentos constituintes. Para cada equipamento apresenta-se uma caracterização da sua representação gráfica e da forma como é controlado e configurado.

2.1 Linha de Produção Flexível

O laboratório I004 do DEEC da FEUP dispõe de uma linha de produção flexível que é utilizada em projetos de unidades curriculares do MIEEC, nomeadamente no ramo de Automação.

2.1.1 Descrição Geral

Esta linha de produção é constituída por 5 módulos básicos (figura 2.1), sendo estes da esquerda para a direita:

- Armazém automático;
- Célula de maquinação série (Plate MS, figura 2.2);
- Célula de maquinação paralela (Plate MP);
- Célula de montagem (Plate M);
- Célula de carga e descarga de peças (Plate C).

As figura 2.2 representa um esquema da linha de produção respetiva, com nomes para cada célula e equipamento. O armazém automático é apresentado num esquema simplificado, apenas com os postos de acesso AT1 e AT2 (tapetes).

O esquema dispõe de três eixos, XX, YY e ZZ, que permitem referenciar as direções e sentidos de qualquer movimento de peças ou equipamentos na linha de produção. O sentido positivo do eixo dos ZZ (ZZ+) aponta na direção do leitor.

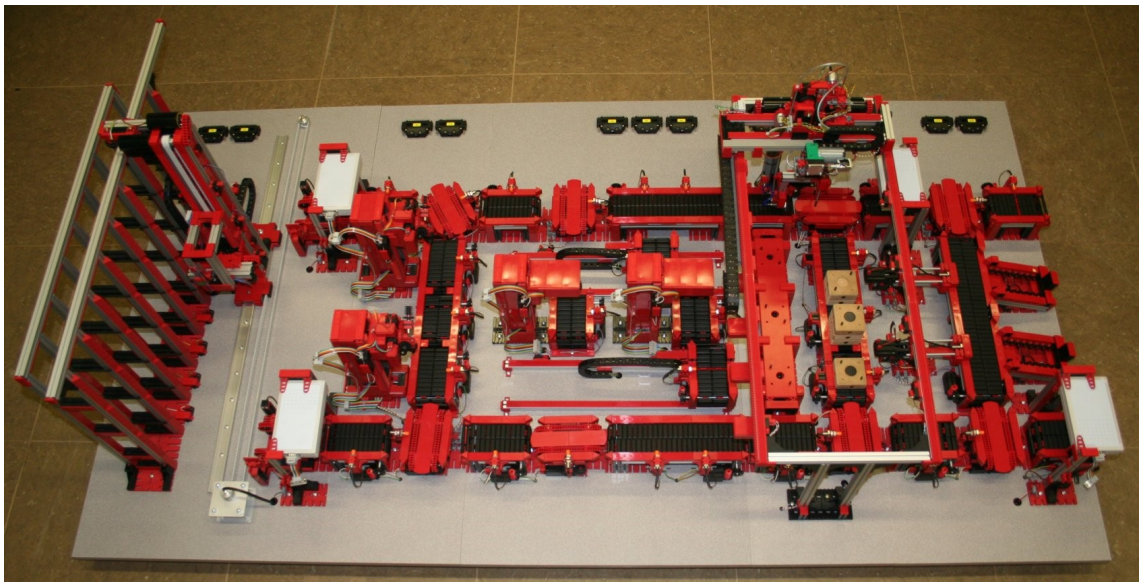

Figura 2.1: Linha de produção flexível

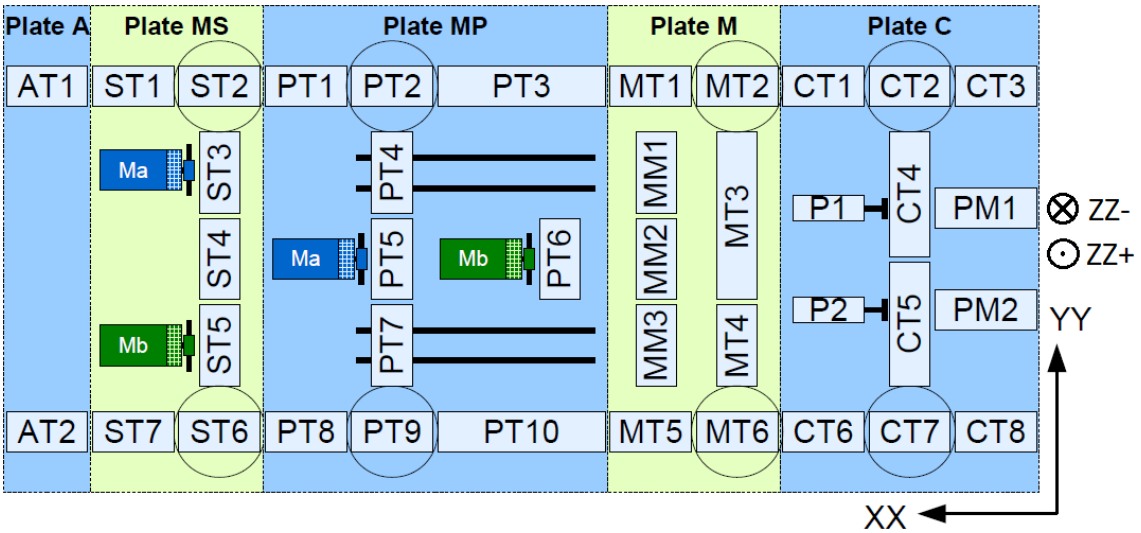

Figura 2.2: Esquema da linha de produção

As várias células da linha de produção permitem processar peças, tanto de forma virtual (e.g. as máquinas com ferramenta não alteram físicamente as peças) como de forma real (e.g. empilhamento de peças). As peças são pequenos blocos de madeira de forma quadrada, com um círculo preto na superfície que permite ativar sensores de presença.

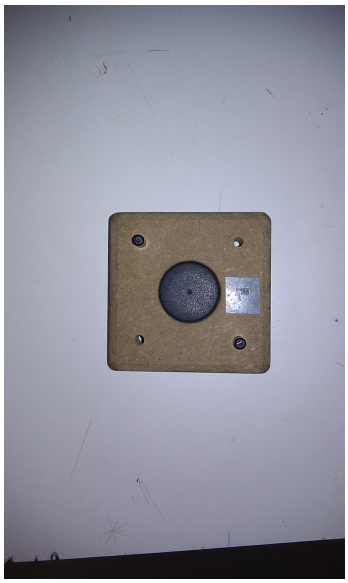

Figura 2.3: Peça utilizada na linha de produção flexível

O encaminhamento de peças entre as várias células é feito utilizando tapetes. Existem três tipos de tapetes:

- Tapetes lineares (figura 2.4): permitem transportar peças nos dois sentidos da sua orientação, que pode ser horizontal ou vertical;
- Tapetes rotativos: têm a mesma funcionalidade dos tapetes lineares mas podem também rodar sobre o seu eixo;
- Tapetes deslizantes: para além de funcionarem como tapetes lineares, podem também fazer movimentos de translação linear, deslizando sobre carris.

Figura 2.4: Tapete linear da linha de produção flexível

Os tapetes rotativos (figura 2.5) permitem transportar peças para o interior das células de fabrico (maquinação, montagem e carga/descarga) , onde poderão ser processadas. No esquema da figura 2.2 estes tapetes distinguem-se por apresentarem um círculo à sua volta (**ST2**, **PT2**...).

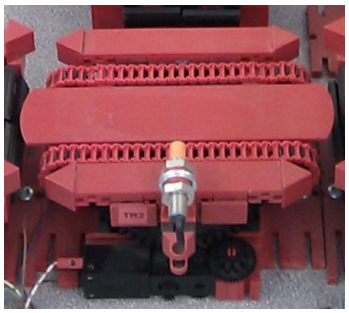

Figura 2.5: Tapete rotativo da linha de produção flexível

Os tapetes deslizantes (figura 2.6) distinguem-se por estarem apoiados sobre carris, como se pode ver nos tapetes **PT4** e **PT7** do esquema da figura 2.2. Deste modo, podem fazer movimentos de translação, ao longo dos carris, permitindo encaminhar peças até ao tapete **PT6**.

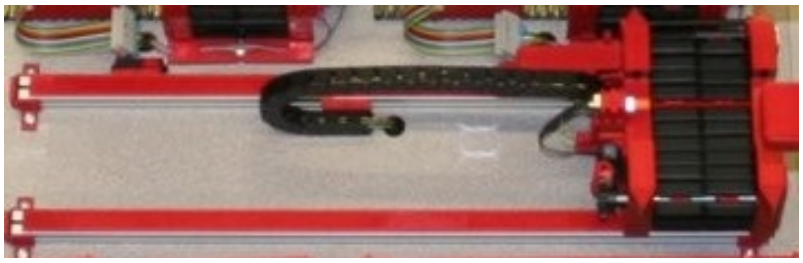

Figura 2.6: Tapete deslizante da linha de produção flexível

O acesso aos sinais de controlo da linha de produção (sensores e atuadores de cada equipamento) é feito através do protocolo Modbus. A linha de produção flexível dispõe de um servidor modbus que atende pedidos de leitura/escrita de dados. Cada equipamento da fábrica possui sensores e atuadores binários que estão associados a entradas e saídas modbus discretas. Cada entrada/saída é referenciada pelo seu endereço de memória no protocolo modbus. A linha de produção comporta-se então como um escravo modbus que poderá ser controlado por qualquer equipamento que suporte o protocolo. A secção 2.2.4 descreve em mais detalhe o protocolo de comunicação modbus.

2.1.2 Armazém Automático

O armazém automático dispõe de 24 posições organizadas em colunas e linhas (figura 2.7). Em cada posição pode ser armazenada uma peça simples ou composta (resultante do empilhamento de várias peças simples).

A colocação e retirada de peças das posições de armazenamento é realizada por intermédio de um *stacker* (torre vertical). Para realizar estas operações este dispositivo pode deslocar-se segundo os eixos XX, YY e ZZ.

Esta célula dispõe também de dois tapetes lineares que são utilizados para a entrada e saída de peças do armazém. O *stacker* é o meio de transporte das peças entre estes dois tapetes e o armazém automático.

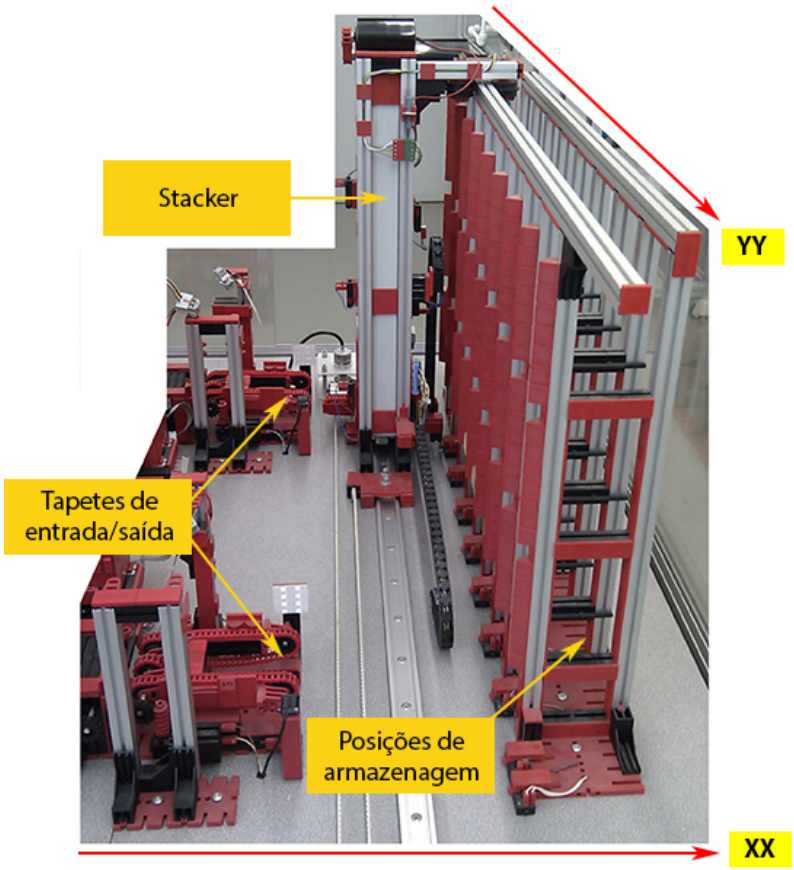

Figura 2.7: Armazém automático

2.1.3 Célula de Maquinação Série

A célula de maquinação série (figura 2.8) é constituída por duas máquinas ferramenta e por diversos tapetes.

O esquema da figura 2.9 ajuda a perceber melhor a disposição dos equipamentos desta célula e apresenta nomenclaturas para cada equipamento.

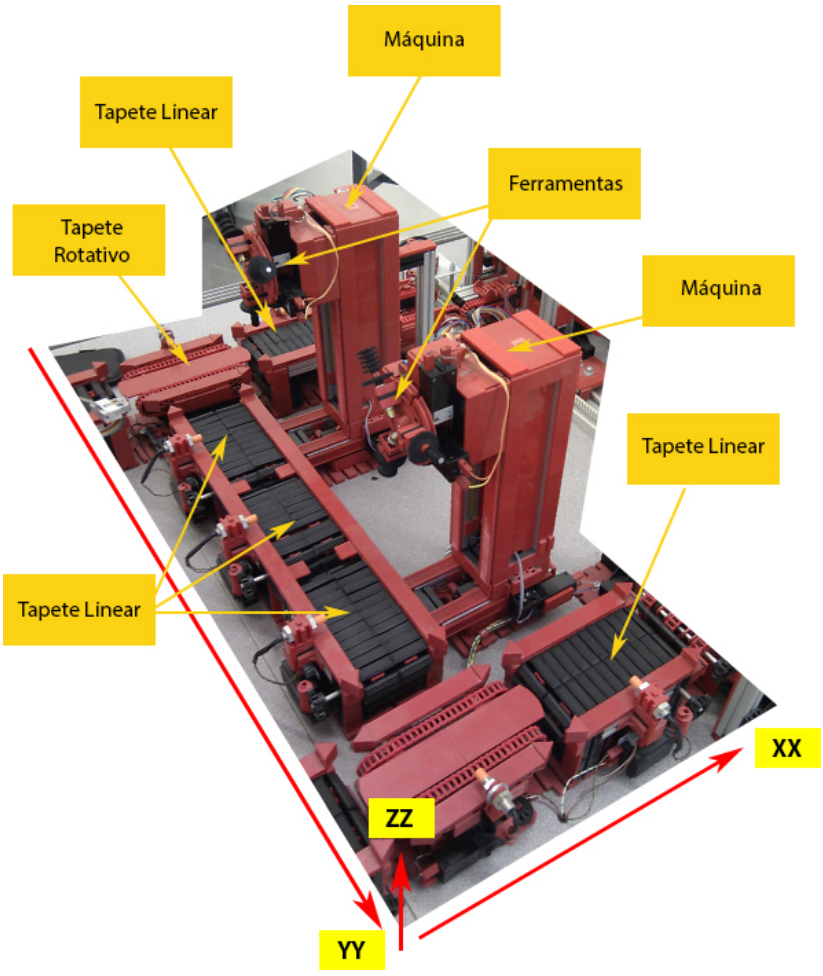

Figura 2.8: Célula de Maquinação Série

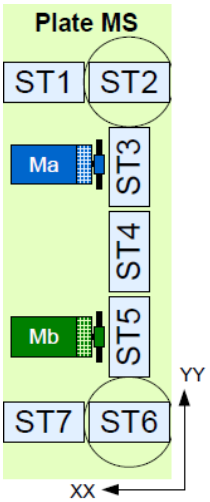

Figura 2.9: Célula de Maquinação Série (esquema)

Esta célula apenas contém tapetes lineares e tapetes rotativos. Os primeiros transportam peças na direcção da sua orientação e podem estar orientados segundo XX ou YY. Os tapetes rotativos, para além de funcionarem também como tapetes lineares, permitem também rodar sobre o seu próprio eixo (ZZ). Os tapetes **ST2** e **ST6** são rotativos e os restantes lineares.

As duas máquinas desta célula (**Ma** e **Mb**) encontram-se montadas em série, ou seja, uma peça que entre de uma lado da célula e saia pelo outro lado, passa sempre por ambas as máquinas. Cada máquina pode ser controlada de forma individual.

As máquinas são constituídas por uma torre e um suporte giratório com três ferramentas. A torre pode movimentar-se segundo o eixo XX, permitindo aproximar e afastar a máquina do tapete anexo, onde são colocadas as peças. O suporte giratório pode ser rodado para se selecionar a ferramenta desejada. Pode também movimentar-se ao longo do eixo ZZ, permitindo aproximar e afastar a ferramenta da peça que se prentende operar.

É de notar que no kit da fábrica as maquinações são virtuais, já que as ferramentas não alteram de qualquer forma as peças sobre as quais atuam.

2.1.4 Célula de Maquinação Paralela

A célula de maquinação paralela (figura 2.10) é constituída por duas máquinas com ferramenta e por vários tapetes, com a disposição esquematizada na figura 2.11.

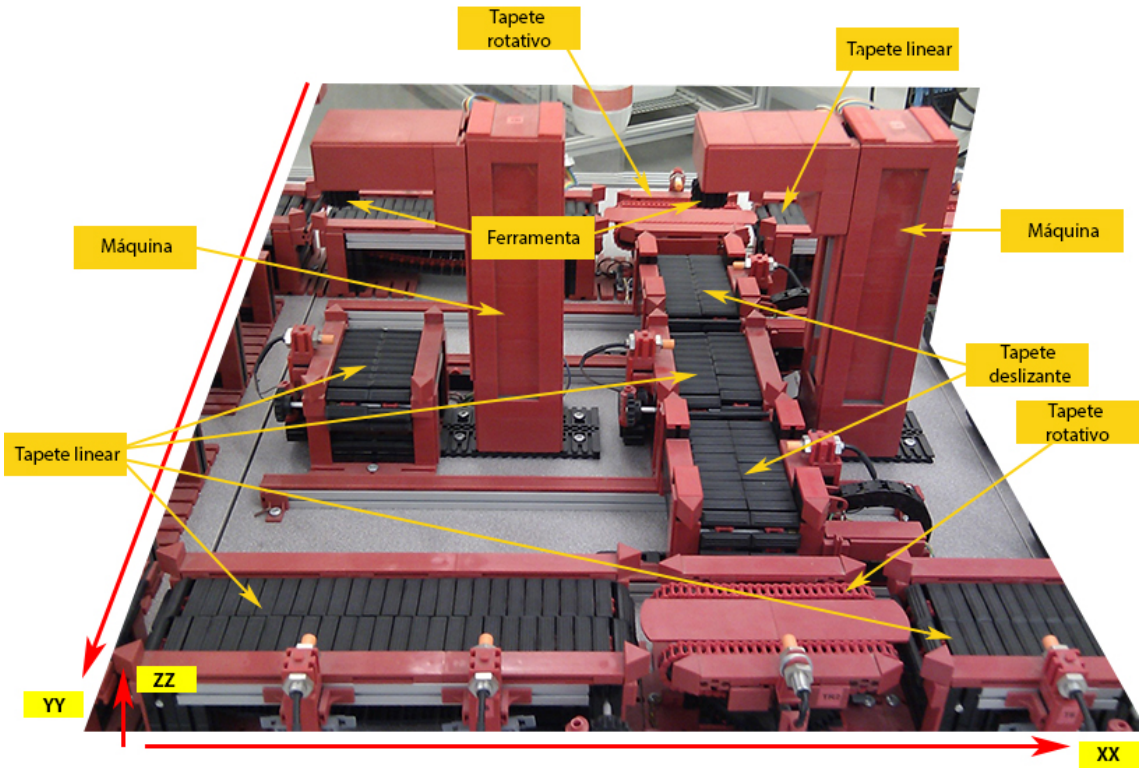

Figura 2.10: Célula de Maquinação Paralela

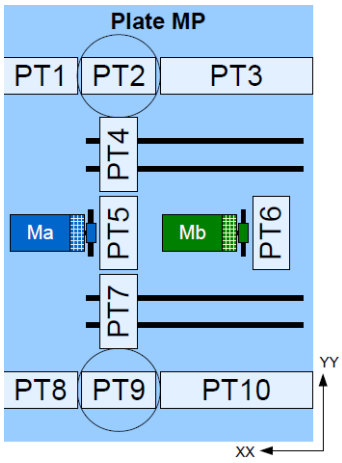

Figura 2.11: Célula de Maquinação Paralela (esquema)

Esta célula, assim como a célula de maquinação série, dispõe de tapetes lineares e rotativos. Para além destes, é também constituída por tapetes deslizantes (**PT4** e **PT7**, figura 2.11) que estão montados sobre carris e podem movimentar-se segundo o eixo XX, estando orientados segundo o eixo YY. Desta forma, as peças maquinadas por esta célula podem seguir diferentes percursos, consoante a necessidade.

No esquema da figura 2.11, a peça poderá passar pelos tapetes **PT4**, **PT5** e **PT7** e ser operada na máquina Ma ou, alternativamente, passar pelos tapetes **PT4**, **PT6** e **PT7** e ser operada na máquina Mb.

Ao contrário das máquinas da célula em série, as máquinas desta célula têm torres fixas, que não podem mover-se segundo o eixo dos XX, e dispõem de uma só ferramenta, montada num suporte não giratório. Este suporte pode também movimentar-se na direção do eixo dos ZZ, aproximando e afastando a ferramenta da peça que se pretende operar.

Assim como na célula de maquinação série, as operações sobre as peças são virtuais e não as alteram de qualquer forma.

2.1.5 Célula de Montagem

A célula de montagem (figura 2.13) é constituída por um robot de três eixos do tipo *"gantry"* e por diversos tapetes e mesas, com a disposição esquematizada na figura 2.12.

O robot desta célula possui uma garra que se pode deslocar nos três eixos (XX, YY ou ZZ) mas dentro de uma zona de operação limitada. Nesta zona existem diversos tapetes e ainda três mesas de montagem.

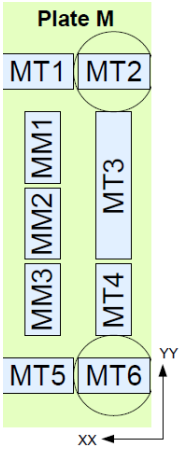

Figura 2.12: Célula de Montagem (esquema)

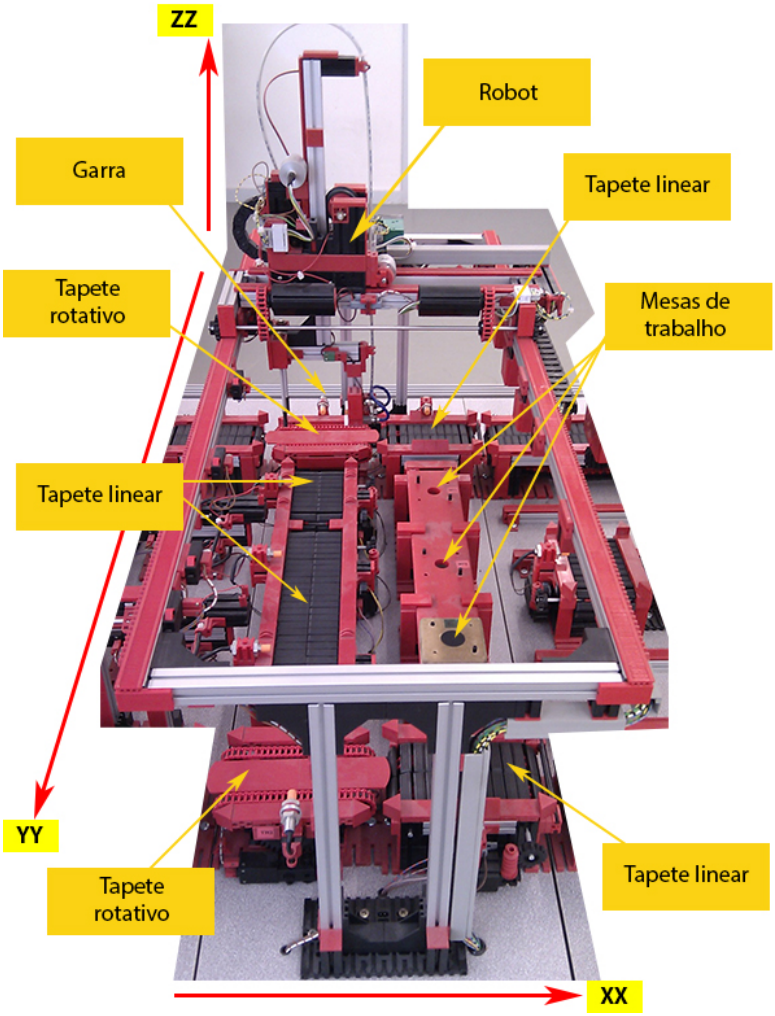

Figura 2.13: Célula de Montagem

As mesas de montagem são três (**MM1**, **MM2** e **MM3**) e servem apenas como suportes horizontais sobre os quais é possível colocar peças. As peças são empilhadas umas em cima das outras formando assim peças compostas (máximo de três peças simples).

A garra do robot permite segurar peças simples ou compostas. É utilizada para efetuar o empilhamento de peças simples, transformando-as em peças compostas, e para transferir as peças compostas para um dos tapetes de saída da célula (**MT4** ou **MT6**).

As peças compostas não podem ser colocadas no armazém devido ao seu tamanho, devendo ser encaminhadas para uma das células de carga/descarga, descritas na secção seguinte.

2.1.6 Célula de Carga e Descarga

A célula de carga/descarga (figura 2.15) permite efetuar a carga de peças do exterior para a linha de produção e a descarga de peças da linha de produção para o exterior. A sua disposição está esquematizada na figura 2.14.

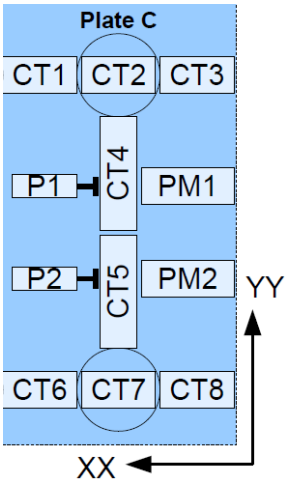

Figura 2.14: Célula de Carga/Descarga (esquema)

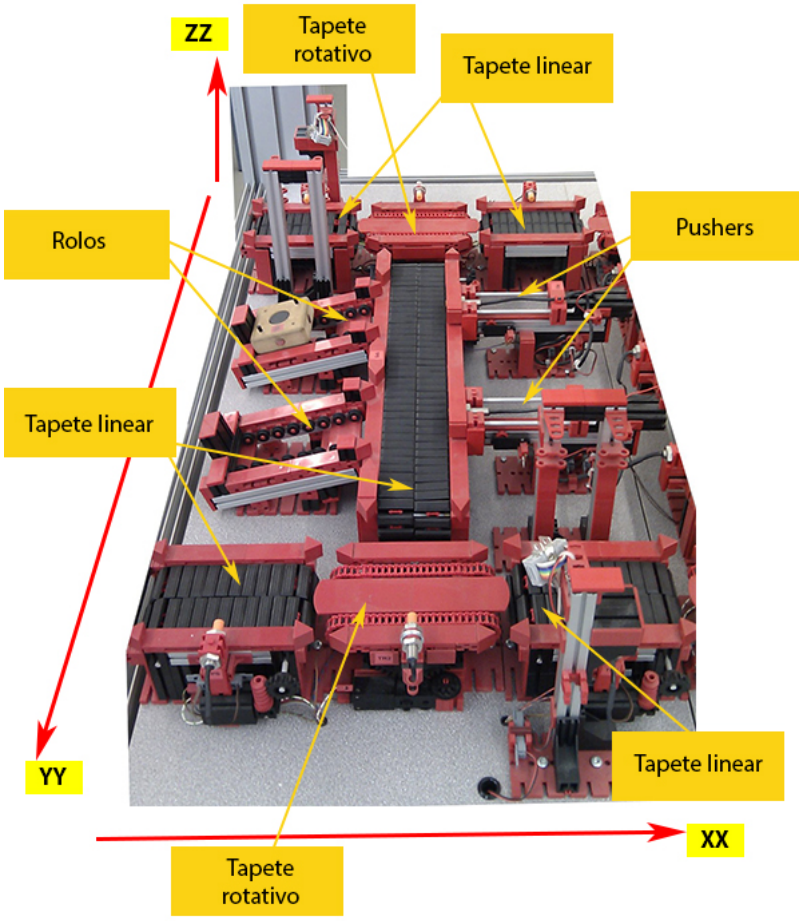

Figura 2.15: Célula de Carga/Descarga

A descarga de peças pode ocorrer por meio de um tapete (**CT3**) ou pelas zonas de armazenagem temporária (**PM1** e **PM2**), que dispõem de rolos sobre os quais as peças deslizam por ação da gravidade para fora da linha de produção. Neste último caso as peças são empurradas por ação de um pusher (**P1** e **P2**) que se movimenta segundo o eixo dos XX.

No caso da descarga por meio do tapete **CT3** a peça terá de ser removida manualmente pelo operador.

A carga de peças é feita de forma manual. Deste modo, o operador terá de colocar a peça no tapete de carga (**CT8**) para que esta seja encaminhada para o interior da célula.

2.2 Simulador da Linha de Produção Flexível

2.2.1 Introdução

O facto de apenas haver uma linha de produção flexível torna-se limitador e insuficiente quando o mesmo é utilizado em unidades curriculares com dezenas de estudantes. Seria impossível partilhar este equipamento com tantos estudantes, que precisam de realizar inúmeros testes de

validação na realização de projetos relacionados com o controlo desta linha de produção.

Com o intuito de resolver estas limitações, foi desenvolvido um simulador em Java que permite replicar quase na totalidade o funcionamento da linha de produção. Deste modo, cada estudante pode realizar todos os testes necessários, em qualquer lugar, sem ter de se deslocar ao laboratório. Para utilizar o simulador, apenas precisa de uma plataforma que disponha de uma máquina virtual Java (ex. Windows, Linux, OSX).

2.2.2 Descrição Geral

O simulador desenvolvido poderia ser uma réplica estática, rígida e não configurável da linha de produção, mas foi construído de forma flexível e configurável. Deste modo, para além de permitir simular o funcionamento da linha de produção já existente, permite também construir diferentes linhas de produção, com as mais diversas disposições e com qualquer número de sub-componentes ou equipamentos.

Assim, é possível definir-se a quantidade de equipamentos de cada tipo, a sua distribuição pela planta e as suas características (e.g. nomenclatura, tamanho, número de sensores). Isto é feito através de um ficheiro de configuração (*plant.properties*), que terá de estar no mesmo diretório do ficheiro de execução do simulador, para que este o consiga ler.

O simulador dispõe de diversos tipos de equipamentos, inspirados nos que já existem na linha de produção flexível:

- Tapete linear;
- Tapete rotativo;
- Tapete deslizante;
- Mesa de trabalho;
- Armazém automático;
- Máquina ferramenta;
- *Pusher*;
- Robot 3D.

A figura 2.16 apresenta um exemplo de uma configuração do simulador.

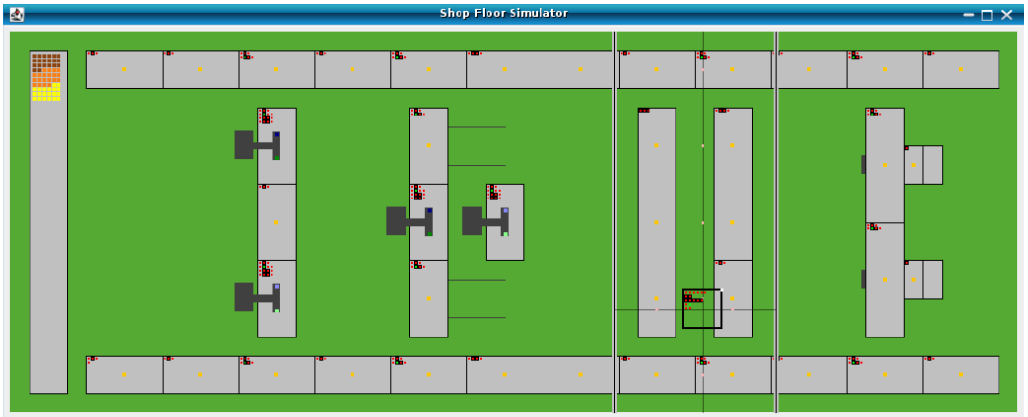

Figura 2.16: Exemplo de uma configuração do simulador da fábrica

2.2.3 Arquitetura de Controlo

A comunicação entre o programa de controlo e o simulador é feita através do protocolo Modbus TCP. O simulador funciona como um servidor modbus que atende pedidos do cliente modbus, o programa de controlo. O simulador define os sensores dos equipamentos como entradas modbus e os atuadores como saídas modbus, associando um endereço individual a cada um. As entradas modbus podem ser discretas (apenas dois estados: TRUE ou FALSE) ou registos (2 bytes de informação). As saídas podem também ser discretas (coils) ou registos de 2 bytes (registos *holding*).

É usada uma biblioteca Java modbus (*jamod*) que dispõe de funções que permitem definir as entradas/saídas do servidor Modbus e de funções que permitem ler/escrever essas entradas/saídas.

O diagrama de blocos da figura 2.17 representa a comunicação entre o programa de controlo e o simulador.

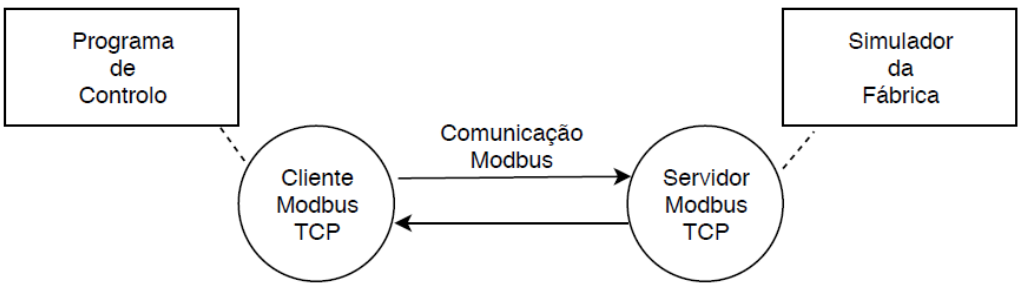

Figura 2.17: Arquitetura de Controlo do Simulador

O programa de controlo faz pedidos de leitura das entradas modbus associadas aos sensores e pedidos de leitura/escrita das saídas associadas aos atuadores. O programa de controlo pode, por exemplo, ler o estado de um sensor discreto de presença de peça para um tapete (TRUE se tiver

peça, FALSE se não tiver) ou escrever na saída discreta modbus (coil) associada a um atuador de movimento de um tapete.

O protocolo Modbus é descrito em mais detalhe na secção seguinte.

2.2.4 Protocolo Modbus

Modbus é um protocolo de comunicação de dados que foi publicado originalmente pela Modicon em 1979 para ser utilizado com os seus autómatos. Atualmente é utilizado globalmente e é considerado um protocolo *standard* para comunicação entre dispositivos industriais. É utilizado na aquisição de sinais de sensores e no comando de atuadores.

O protocolo Modbus é muito utilizado na área de Automação porque, para além de ter sido desenvolvido tendo em vista aplicações industriais, é livre de taxas de licenciamento. É também um protocolo que se adapta facilmente a diversos meios físicos, sendo utilizado em milhares de equipamentos existentes no mercado.

2.2.4.1 Modelo de Comunicação

O protocolo Modbus implementa um modelo de interação do tipo mestre-escravo (ou cliente-servidor). Deste modo, o escravo (servidor) não deve iniciar qualquer tipo de comunicação enquanto não tiver sido requisitado pelo mestre (cliente). Um exemplo comum deste modelo de comunicação é ter um autómato programável como mestre que envia pedidos solicitando aos escravos que enviem os dados lidos por um conjunto de sensores, ou que envia valores a serem escritos em saídas modbus, permitindo controlar atuadores.

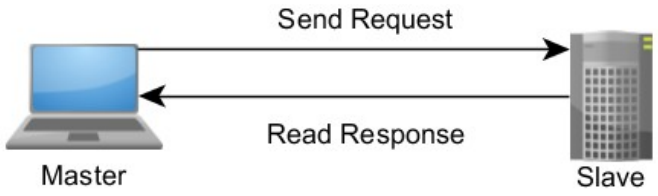

Figura 2.18: Modelo de Comunicação Modbus

2.2.4.2 Estrutura protocolar do Modbus

Nas implementações iniciais, o protocolo Modbus era bastante simples já que era criado no topo da comunicação série. Mais tarde, com a utilização massiva de TCP/IP, o protocolo foi portado para utilizar esta pilha protocolar. Isto levou a uma separação entre o protocolo básico, que define a unidade de dados de protocolo (PDU) e a camada de rede, que define a unidade de dados de aplicação (ADU).

2.2.4.3 Modelo de Dados

Os dados que podem ser acedidos através do protocolo Modbus são armazenados, de forma geral, num dos seguintes blocos de dados: coils, entradas discretas, registos *holding* e registos de entrada. Os registos *holding* podem também ser denominados registos de saída e as coils podem ser referidos como saídas digitais ou discretas.

As coils e as entradas discretas são binárias (um bit de informação) e os registos *holding* e de entrada são palavras com 16 bits de informação.

A tabela que se segue descreve cada um destes blocos de dados, indicando o seu tipo e os direitos de acesso por parte do cliente (mestre) e do servidor (escravo). O servidor é responsável por criar estes blocos de memória, e tem total acesso (leitura e escrita) a todos os blocos.

Tabela 2.1: Blocos do modelo de dados Modbus

Bloco de dados	**Tipo de dados**	**Acesso ao cliente**	**Acesso ao servidor**
Coils	Booleano	Leitura/escrita	Leitura/escrita
Entradas discretas	Booleano	Só leitura	Leitura/escrita
Registos *holding*	Palavra (2 bytes)	Leitura/escrita	Leitura/escrita
Registos de entrada	Palavra (2 bytes)	Só leitura	Leitura/escrita

2.2.4.4 Endereçamento de Dados

Cada bloco de dados contém um espaço de endereçamento distinto, compreendido entre 0 e 65535. O protocolo Modbus define o endereço de cada elemento[1] de dados na gama de 0 a 65535 e o número de referência do elemento vai de 1 a n, onde n tem o valor máximo de 65536. Deste modo, a coil 1 está no endereço 0 do bloco de coils, enquanto o registo *holding* 35 está no endereço 34 do bloco de memória reservado para registos *holding*.

2.2.4.5 Funções Modbus

Os pedidos de leitura e escrita nas posições de memória dos blocos de dados Modbus, descritos anteriormente, são realizados através de funções. Cada função é identificada através de um código.

A tabela que se segue apresenta os códigos das principais funções do protocolo Modbus, com a respetiva descrição. Os códigos de funções fazem parte do cabeçalho das tramas enviadas através deste protocolo. A trama PDU Modbus é descrita em detalhe na secção 2.2.4.6.

Estas funções permitem ler e escrever em posições de memória de qualquer bloco de dados (coils, entradas discretas, registos de entrada e registos *holding*).

[1] Daqui em diante um elemento de dados Modbus é referido como um elemento Modbus.

Tabela 2.2: Lista de funções Modbus

Código da função	**Descrição**
0x01	Lê um número variável de saídas discretas (coils)
0x02	Lê um número variável de entradas discretas
0x03	Lê um número variável de registos *holding*
0x04	Lê um número variável de registos de entrada
0x05	Altera o estado de uma saída discreta (coil)
0x06	Altera o estado de um registo *holding*
0x07	Lê o estado do registo de exceções (8 bits previamente configurados)
0x08	Funções de diagnóstico
0x0F	Altera o estado de uma quantidade variável de saídas discretas
0x10	Altera o estado de uma quantidade variável de registos *holding*

Para as funções que permitem escrever/ler uma quantidade variável de elementos Modbus, o número de elementos é definido na trama de pedido.

2.2.4.6 Trama PDU

A trama definida pelo PDU do Modbus é formada por um código de função de 1 byte seguido dos dados de função, que vão até 252 bytes.

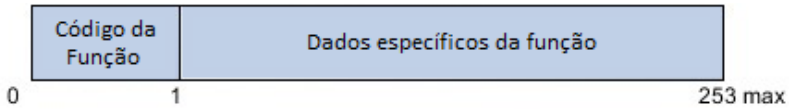

Figura 2.19: A PDU do Modbus

O código de função é validado pelo escravo, que responde com uma exceção se este não for reconhecido. Se for aceite, o escravo começa a decompor os dados consoante a definição da função.

2.2.4.7 Modbus TCP

O TCP (Transmission Control Protocol) é um dos formatos padrão para as ADUs (Application Data Unit) do Modbus. Pode ser usado em redes modernas TCP/IP ou UDP/IP.

As tramas (ADUs) do tipo TCP são constituídas pelo cabeçalho MBAP (Modbus Application Protocol) e pela PDU descrita anteriormente, como se pode ver na figura que se segue.

Figura 2.20: A ADU do tipo TCP/IP

O cabeçalho MBAP desta trama é constituído pelos seguintes campos de dados:

- ID da transação: identificador único de transações; é essencial em redes que possam ter vários pedidos a ser processadas em simultâneo (p.e. um cliente pode enviar os pedidos 1, 2 e 3 e obter respostas por parte do servidor na ordem 2, 1, 3, conseguindo identificar cada uma delas corretamente);
- Protocolo: identificador do protocolo; geralmente é zero, mas pode ser usado para expandir o comportamento do protocolo;
- Tamanho: indica o tamanho do restante pacote de dados;
- *Unit* ID: permite identificar o endereço do dispositivo escravo para o qual a PDU é destinada.

2.2.5 Peças do Simulador

As peças da linha de produção flexível são todas idênticas, tanto na sua forma como na sua cor. Por outro lado, as peças do simulador apresentam vários tipos, que são definidos por diferentes formas (quadrada, redonda...), cores e uma nomenclatura única (P1, P2...).

A figura que se segue apresenta vários exemplos de peças com formas e cores diferentes. As peças encontram-se sobre tapetes lineares, que são descritos na secção 2.2.6.

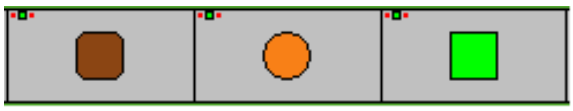

Figura 2.21: Exemplo de vários tipos de peça

2.2.5.1 Ficheiro de Configuração

Os tipos de peça são definidos no ficheiro de configuração (plant.properties) através das seguintes entradas (cada entrada é uma nova linha no ficheiro):

- *blocktype.ID.name = NAME*
  NAME: nomenclatura para o tipo de peça (p.e. P1);
- *blocktype.ID.name = COLOR*
  COLOR: código de cores RGB, em hexadecimal, para este tipo de peça;
- *blocktype.ID.shape = SHAPE*
  SHAPE: forma das peças deste tipo; pode tomar os valores *rounded* (forma arredondada), *square* (forma quadrada) ou *circle* (forma circular).

O "ID" em cada um dos parâmetros descritos é um número inteiro que permite identificar de forma única cada tipo de peça.

2.2.6 Tapete Linear

Um tapete linear permite movimentar uma peça nos dois sentidos da sua orientação (eixos XX ou YY, conforme a sua disposição). A velocidade de movimentação tem uma componente constante e uma componente variável, ambas definidas no ficheiro de configuração. Para cada tapete linear é gerado um número aleatório pertencente ao intervalo de valores [-*speedDelta*; +*speedDelta*], sendo *speedDelta* o valor da componente variável definido no ficheiro de configuração. Este valor é somado à componente constante e obtém-se assim a velocidade de cada tapete.

2.2.6.1 Representação Gráfica

Em termos visuais, o tapete linear tem uma forma retangular. Os tapetes curtos dispõem de um pequeno quadrado amarelo no centro (figura 2.22) e os tapetes longos de um quadrado amarelo em cada extremidade (figura 2.23), correspondendo a sensores de presença de peça. No canto superior esquerdo de cada tapete existem ainda pequenos quadrados que representam o estado lógico de cada sensor/atuador associado a este tipo de tapetes, tornando-se verdes quando o sinal está ativo (TRUE) e vermelhos quando está inativo (FALSE). É de notar que os quadrados dos sensores apresentam um fundo preto, para se distinguirem dos atuadores.

Os sensores e atuadores do tapete linear são descritos na subsecção seguinte.

Figura 2.22: Tapete linear curto

Figura 2.23: Tapete linear longo

2.2.6.2 Controlo

Os tapetes lineares possuem um ou mais sensores binários que permitem detetar a presença de uma peça sobre os mesmos. Nos tapetes mais curtos apenas há um sensor, que está colocado no seu centro. Nos tapetes mais longos existem dois ou mais sensores, posicionados de forma simétrica ao longo do mesmo.

Dispõem também de dois atuadores binários, um para cada sentido de movimentação. Quando um destes atuadores fica ativo (TRUE) o motor de movimento nesse sentido fica ligado. Se os atuadores estiverem ativos em simultâneo, o motor fica desligado.

Como já foi dito, estes tapetes apenas movimentam peças na direção da sua orientação.

A tabela 2.3 descreve cada um destes sensores/atuadores.

Tipo	Sigla	Nome
Comando Binário	mp	movimento no sentido positivo (XX ou YY)
Comando Binário	mm	movimento no sentido negativo (XX ou YY)
Sensor Binário 1	p1	Presença de peça
Sensor Binário 2	p2	Presença de peça
Sensor Binário 3		...

Tabela 2.3: Atuadores e sensores do tapete linear

2.2.6.3 Ficheiro de configuração

O ficheiro de configuração permite definir os seguintes parâmetros para cada tapete linear:

- *facility.ID.type = conveyor*
  O tipo de equipamento é obrigatoriamente "conveyor" para tapetes lineares;
- *facility.ID.length*
  Comprimento do tapete;
- *facility.ID.width*
  Largura do tapete;
- *facility.ID.orientation*
  Orientação do tapete (vertical ou horizontal);
- *facility.ID.center.x*
  Coordenada x do centro do tapete na frame do simulador;
- *facility.ID.center.y*
  Coordenada y do centro do tapete na frame do simulador;
- *facility.ID.sensors*
  Número de sensores de presença de peça.
- *configuration.conveyorspeed*
  Componente constante da velocidade do tapete linear;
- *configuration.conveyorspeeddelta*
  Componente variável da velocidade do tapete linear;

Todos os parâmetros definidos no ficheiro de configuração seguem o formato *"facility.ID.parameter = value"*, sendo "ID" um número inteiro que permite identificar de forma única cada equipamento.

De seguida apresenta-se um exemplo de atribuição de valores a cada um dos parâmetros que descreve um tapete linear no ficheiro de configuração:

- *facility.12.type = conveyor*
- *facility.12.length = 8*
- *facility.12.width = 2*
- *facility.12.orientation = horizontal*
- *facility.12.center.x = 36*
- *facility.12.center.y = 2*

- *facility.12.sensors = 2*

2.2.7 Tapete Rotativo

O tapete rotativo comporta-se de forma idêntica ao tapete linear, mas permite também fazer movimentos de rotação sobre o eixo dos ZZ.

Visualmente, o tapete rotativo é quase idêntico ao tapete linear, sendo a única diferença o facto de apresentar um maior número de quadrados no seu canto superior esquerdo. Isto verifica-se porque o tapete rotativo apresenta um maior número de sensores e atuadores, descritos na subsecção que se segue.

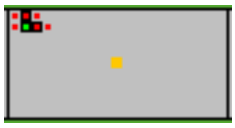

Figura 2.24: Tapete Rotativo

2.2.7.1 Controlo

De modo a controlar a sua rotação, o tapete dispõe de dois atuadores binários, um para cada sentido de rotação. Se estes sinais binários estiverem ativos em simultâneo o simulador tem um mecanismo de proteção que leva à paragem de ambos os motores. Só é permitido rodar o tapete até 90º e as duas extremidades destes 90º são sinalizadas por dois sensores binários de fim de curso.

A tabela 2.4 descreve os sensores e atuadores do tapete rotativo e a figura 2.25 descreve a sequência de comandos, movimentos e posições possíveis.

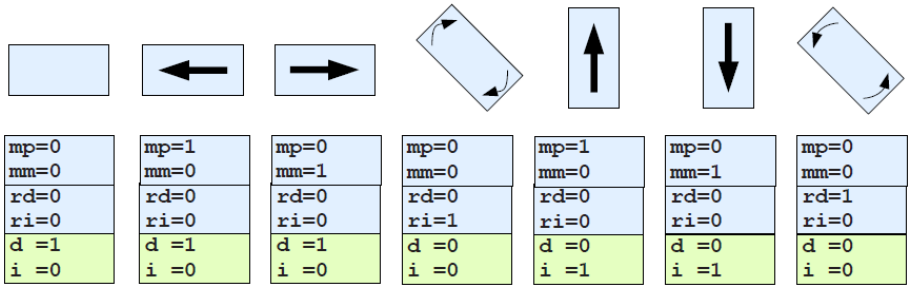

Figura 2.25: Movimentos do tapete rotativo

Tipo	Sigla	Nome
Comando Binário	mp	movimento no sentido positivo (XX ou YY)
Comando Binário	mm	movimento no sentido negativo (XX ou YY)
Comando Binário	rd	rotação no sentido directo
Comando Binário	ri	rotação no sentido inverso
Sensor Binário	p	presença de peça
Sensor Binário	d	fim de rotação no sentido directo
Sensor Binário	i	fim de rotação no sentido inverso

Tabela 2.4: Atuadores e sensores do tapete rotativo

2.2.7.2 Ficheiro de configuração

A configuração dos tapetes rotativos é idêntica à configuração dos tapetes lineares, descrita na secção 2.2.6.3. É de notar que o parâmetro que indica o tipo de equipamento (*facility.ID.type*) é distinto: para o tapete linear é *"conveyor"*, e para o tapete rotativo é *"rotator"*. Há ainda um novo parâmetro que permite configurar a velocidade de rotação dos tapetes rotativos (*configuration.rotationspeed*). Este é um parâmetro global que afeta todos os tapetes rotativos.

2.2.8 Tapete Deslizante

O tapete deslizante tem um comportamento muito semelhante a um tapete rotativo, sendo o movimento de rotação substituído por um movimento de translação linear ao longo de carris. Desta forma o tapete deslizante possui o mesmo número de sensores e atuadores que o tapete rotativo.

Visualmente é idêntico ao tapete rotativo, mas acrescido de dois carris, representados por linhas retas, como se pode ver na figura 2.26.

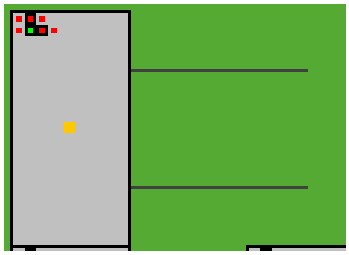

Figura 2.26: Tapete Deslizante

2.2.8.1 Controlo

Para além dos sinais idênticos ao tapete linear, o tapete deslizante dispõe ainda de dois atuadores binários para o movimento de translação nos dois sentidos e de dois sensores binários de fim de movimento de translação, em cada extremidade dos carris.

A tabela 2.5 apresenta cada um destes sinais binários.

Tipo	Sigla	Nome
Comando Binário	mp	movimento no sentido positivo (YY)
Comando Binário	mm	movimento no sentido negativo (YY)
Comando Binário	tp	translação no sentido positivo (XX)
Comando Binário	tm	translação no sentido negativo (XX)
Sensor Binário	p	presença de peça
Sensor Binário	fp	Fim translação no sentido positivo (XX)
Sensor Binário	fm	Fim translação no sentido negativo (XX)

Tabela 2.5: Atuadores e sensores do tapete deslizante

2.2.8.2 Ficheiro de configuração

A configuração dos tapetes deslizantes é idêntica à dos tapetes lineares e rotativos (ver secção 2.2.6.3), alterando-se apenas o valor do parâmetro que indica o tipo de equipamento (é *"rail"* em vez de *"conveyor"* ou *"rotator"*).

2.2.9 Mesa de Trabalho

A mesa de trabalho é apenas utilizada para posicionar peças de forma temporária. São normalmente utilizadas dentro da área de alcance do robot 3D, para que este possa pousar peças.

Assim como o tapete linear, a mesa pode ser curta, tendo apenas um sensor, ou longa, tendo dois ou mais sensores.

Visualmente, é idêntica ao tapete linear da figura 2.22, mas sem apresentar os quadrados do canto superior com o estado de atuadores, já que apenas dispõe de sensores de presença.

2.2.9.1 Controlo

Do ponto de vista do controlo comporta-se como um tapete sem movimento, tendo apenas um ou vários sensores binários de presença e não tendo qualquer atuador, como se pode ver na tabela 2.6.

A tabela apenas indica um sensor de presença, mas a mesa de trabalho poderá apresentar múltiplos sensores (a sua quantidade é indicada no ficheiro de configuração).

Tipo	Sigla	Nome
Sensor Binário	p	presença de peça

Tabela 2.6: Atuadores e sensores da mesa de trabalho

2.2.9.2 Ficheiro de configuração

A mesa de trabalho é configurada de forma semelhante ao tapete linear (ver secção 2.2.6.3). O valor do parâmetro que define o tipo de equipamento *(facility.ID.type)* é *"table"* em vez de *"conveyor"*.

2.2.10 Armazém Automático

O armazém automático do simulador é muito simplificado em relação ao armazém da linha de produção flexível do laboratório de Automação. Assume-se que este se encontra automatizado por um programa já existente e a interação com o armazém fica limitada a pedidos de armazenagem de peças, ou a pedidos de retirada de peças de um determinado tipo para a linha de produção.

A interface física entre a célula do armazém e as restantes células é feita através de dois tapetes. Um dos tapetes permite retirar peças do armazém, enquanto o outro tapete permite armazenar peças no armazém.

2.2.10.1 Representação Gráfica

Visualmente, o armazém do simulador é um retângulo com pequenos quadrados que representam as peças (figura 2.27). As diferentes cores estão associadas a cada tipo de peça, sendo definidas no ficheiro de configuração.

Na figura 2.27 estão também representados os dois tapetes que permitem aceder ao armazém. No tapete superior colocam-se as peças provenientes do armazém e no tapete inferior chegam peças que são armazenadas. O tapete inferior apresenta um quadrado adicional no seu canto superior esquerdo, que representa o estado de um atuador que permite inserir peças no armazém (fica a TRUE para inserir a peça). Quando a peça é armazenada é acrescentado um quadrado ao armazém com a cor desse tipo de peça.

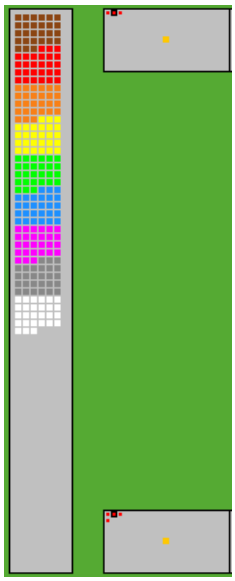

Figura 2.27: Armazém com os dois tapetes de interface

2.2.10.2 Controlo

Os tapetes de acesso ao armazém, assim como tapetes lineares, dispõem de duas saídas digitais para comandar a movimentação (para cada sentido) e uma entrada digital que indica a presença de uma peça sobre o tapete.

O tapete de entrada de peças no armazém (tapete de baixo na figura 2.27) dispõe de um atuador (saída discreta Modbus) adicional que, ao transitar de FALSE para TRUE, permite armazenar uma peça que se encontre no tapete. A operação termina quando a peça desaparece do tapete e é adicionado um quadrado ao armazém. Esta saída digital tem a sigla *"in"* na tabela 2.7.

O tapete de saída de peças do armazém (tapete de cima na figura 2.27) dispõe de um registo *holding* Modbus (saída de 16 bits) que permite identificar o tipo de peça que se pretende transferir do armazém para o tapete. Para esta operação é necessário que o registo transite do valor 0 para o valor correspondente ao tipo de peça que se pretende retirar do armazém. A operação termina logo que a peça apareça no tapete. O registo descrito tem a sigla *"tp"* na tabela 2.8.

Os sensores, atuadores e registos de cada um dos tapetes de acesso ao armazém são descritos nas tabelas que se seguem.

Tipo	Sigla	Nome
Comando Binário	mp	movimento no sentido positivo (XX ou YY)
Comando Binário	mm	movimento no sentido negativo (XX ou YY)
Comando Binário	in	inserir peça no armazém
Sensor Binário	p	presença de peça

Tabela 2.7: Atuadores e sensores binários do tapete de entrada de peças no armazém

Tipo	Sigla	Nome
Comando Binário	mp	movimento no sentido positivo (XX ou YY)
Comando Binário	mm	movimento no sentido negativo (XX ou YY)
Comando Word	tp	tipo de peça a remover
Sensor Binário	p	presença de peça

Tabela 2.8: Atuadores/sensores/registos do tapete de saída de peças do armazém

2.2.10.3 Ficheiro de configuração

O armazém automático e os dois tapetes de acesso acima descritos são também definidos através de um ficheiro de configuração (plant.properties).

O armazém é descrito pelos seguintes parâmetros:

- *warehouse.ID.length*
  Comprimento do armazém;
- *warehouse.ID.width*
  Largura do armazém;
- *warehouse.ID.orientation*
  Orientação: vertical ou horizontal;
- *warehouse.ID.center.x*
  Coordenada X da posição do centro do armazém na frame do simulador;
- *warehouse.ID.center.x*
  Coordenada Y da posição do centro do armazém;
- *warehouse.ID.block.TYPE.stock*
  Quantidade de peças do tipo "TYPE" a começar no armazém. Este parâmetro é adicionado para cada um dos tipos de peça existentes.

O ID em cada um dos parâmetros atrás descritos é um número inteiro que permite identificar o armazém (como só se usa um armazém é sempre 1).

O exemplo que se segue permite demonstrar a configuração de um armazém automático:

warehouse.1.length = 18

warehouse.1.width = 2
warehouse.1.orientation = vertical
warehouse.1.center.x = 2
warehouse.1.center.y = 10
warehouse.1.block.1.stock = 30
warehouse.1.block.2.stock = 10
warehouse.1.block.3.stock = 5

2.2.11 Máquina Ferramenta

A máquina ferramenta permite efetuar operações sobre uma peça que se encontre no tapete anexo à máquina. Considera-se que este tapete é parte integrante da máquina ferramenta, tendo sensores/atuadores idênticos aos de um tapete linear.

Esta máquina é constituída por um suporte giratório com três ferramentas. O tipo de cada ferramenta é definido no ficheiro de configuração (descrito na secção 2.2.11.5).

2.2.11.1 Representação Gráfica

Visualmente, a máquina do simulador apresenta cor cinzenta e é constituída por duas componentes: uma torre (retângulo cinzento) e um suporte giratório em forma de T com pequenos quadrados que representam as ferramentas.

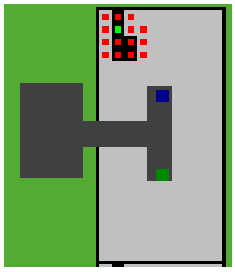

Figura 2.28: Máquina Ferramenta e tapete anexo

2.2.11.2 Controlo

As ferramentas encontram-se integradas no suporte giratório da máquina. Para trocar de ferramenta é utilizado um comando de rotação desse suporte, sempre no mesmo sentido, até que a ferramenta desejada se encontre na posição de operação. Há um sensor que é ativado sempre que qualquer uma das ferramentas se encontre nesta posição. Uma vez que este sensor não indica qual a ferramenta, mas apenas a presença de uma qualquer ferramenta, é da responsabilidade do programa de controlo guardar em memória o número de rotações efetuadas, e assim determinar qual a ferramenta que se encontra na posição de operação. Considera-se que inicialmente é sempre

a ferramenta T1 (com ID = 1) que se encontra na posição de operação e que as ferramentas se encontram montadas pela ordem T1, T2, T3.

O suporte giratório pode ainda deslocar-se segundo o eixo dos ZZ (cima-baixo) e segundo o eixo dos XX (esquerda-direita). Para o controlo da posição do suporte giratório em cada um destes eixos, existem dois comandos binários para movimentar a torre em cada um dos sentidos, e dois sensores que indicam a chegada a cada uma das posições extremas.

Para iniciar a operação da ferramenta é utilizado um simples comando binário, sendo a velocidade de rotação da ferramenta fixa. É de notar que a máquina só deve operar quando o suporte giratório se encontrar na posição de operação segundo X, estando chegado à frente (sobre o tapete anexo), e na posição de operação segundo Z (posição mais baixa), estando a ferramenta junto à peça que se pretende operar.

De forma a prevenir danos físicos ao equipamento, os movimentos nos eixos XX e ZZ nunca devem ultrapassar os limites. Este dano é representado através da alteração das margens do tapete anexo à máquina para vermelho.

A tabela 2.9 apresenta os sensores e atuadores associados à máquina ferramenta.

Tipo	Sigla	Nome
Comando Binário	mp	movimento no sentido positivo (XX ou YY)
Comando Binário	mm	movimento no sentido negativo (XX ou YY)
Comando Binário	tc	troca de ferramenta (tool change)
Comando Binário	--	troca de ferramenta (outro sentido)
Comando Binário	tr	rotação de ferramenta (tool rotate)
Comando Binário	yp	movimento torre eixo y (sentido positivo)
Comando Binário	ym	movimento torre eixo y (sentido negativo)
Comando Binário	zp	movimento torre eixo z (sentido positivo)
Comando Binário	zm	movimento torre eixo z (sentido negativo)
Sensor Binário	p	presença de peça
Sensor Binário	pt	presença de ferramenta
Sensor Binário	yp	Fim movimento eixo y (sentido positivo)
Sensor Binário	ym	Fim movimento eixo y (sentido negativo)
Sensor Binário	zp	Fim movimento eixo z (sentido positivo)
Sensor Binário	zm	Fim movimento eixo z (sentido negativo)

Tabela 2.9: Atuadores e sensores da máquina ferramenta

2.2.11.3 Transformação de Peças

A operação das máquinas sobre as peças permite transformá-las, alterando o seu tipo. Os diferentes tipos de peças são caracterizados por uma nomenclatura única (P1, P2, P3..), diferentes formas e cores, como descrito na secção 2.2.5 sobre as peças. Estes tipos de peças são definidos no ficheiro de configuração.

As várias sequências de transformação possíveis são também definidas no ficheiro de configuração e são caracterizadas pelos seguintes elementos:

- Tipo de ferramenta
- Tipo de peça inicial
- Tipo de peça final
- Duração da operação

O tipo de ferramenta é identificável através de um ID único (1, 2, 3..) e a cada máquina são atribuídos os IDs das ferramentas que a constituem. Tudo isto é definido no ficheiro de configuração (ver secção 2.2.11.5).

Se a operação sobre a peça durar menos tempo que o suposto, esta não se transforma, mantendo assim o seu tipo inicial. Se durar mais tempo que o suposto a peça fica danificada, mudando a sua cor para preto.

Em certos casos, para se transformar uma peça inicial num determinado tipo de peça, é necessário transformá-la primeiro numa peça intermédia. A peça inicial pode ser vista como a matéria prima, e a peça intermédia como um subproduto que ainda terá de ser processado para se obter a peça pretendida. O exemplo que se segue irá demonstrar uma situação em que isso acontece, sendo este o caso mais complexo de transformação.

2.2.11.4 Exemplo de Transformação de Peças

A tabela 2.10 apresenta todas as sequências de transformação existentes num simulador que se assume configurado dessa forma.

Matéria Prima	Peça final	Tipo de Máquina	Ferramenta	Tempo Processamento
P1	P3	Ma	T1	5 s
P3	P5	Ma	T2	10 s
P5	P7	Ma	T3	5 s
P2	P4	Mb	T1	10 s
P4	P6	Mb	T2	5 s
P6	P8	Mb	T3	5 s
P7	P8	Mc	T1	20 s
P8	P7	Mc	T2	20 s
P7	P9	Mc	T3	20 s
P8	P9	Mc	T3	20 s

Tabela 2.10: Sequências de transformação possíveis

Imaginemos que se pretende obter uma peça final do tipo P5 a partir de uma peça inicial do tipo P1. Observando a tabela anterior, rapidamente se conclui que não há qualquer transformação direta de peça P1 para P5.

Desta forma teremos de, numa primeira fase, transformar a peça P1 numa peça intermédia P3, atuando durante 5s com a ferramenta T1 da máquina de tipo Ma. Por fim, transforma-se a peça intermédia P3 na peça pretendida P5, atuando durante 10s com a ferramenta T2 da máquina de tipo Ma.

Poderá haver ainda transformações mais complexas, para as quais se terá de passar por vários tipos de peças intermédias antes de se obter a peça final pretendida. Um exemplo desta situação é uma transformação P2 $\rightarrow$ P8 que terá de seguir a seguinte sequência de transformações: P2 $\rightarrow$ P4 $\rightarrow$ P6 $\rightarrow$ P8.

2.2.11.5 Ficheiro de Configuração

Os tipos de ferramenta são definidos no ficheiro de configuração por apenas dois parâmetros: um identificador único (ID) e uma cor. São criados através da seguinte entrada no ficheiro:

- *tool.ID.color = COLORCODE*
  ID é substituído pelo número inteiro que identifica o tipo de ferramenta; COLORCODE é substituído pelo código de cor RGB, em hexadecimal.

As transformações de peças são definidas pelas seguintes entradas no ficheiro de configuração (plant.properties):

- *transformation.ID.tool = TOOL*
  TOOL: Número inteiro identificador do tipo de ferramenta;
- *transformation.ID.initial = INITIAL*
  INITIAL: Número inteiro identificador do tipo de peça origem;
- *transformation.ID.final = FINAL*
  FINAL: Número inteiro identificador do tipo de peça final (obtido após transformação);
- *transformation.ID.duration = DURATION*
  DURATION: Duração de operação da ferramenta.

O "ID" em cada um dos parâmetros acima descritos é um número inteiro que permite identificar cada uma das transformações definidas.

Os parâmetros que caracterizam uma máquina ferramenta com tapete anexo no ficheiro de configuração são os seguintes:

- *facility.ID.type = machine*
  O tipo de equipamento é obrigatoriamente "machine";
- *facility.ID.length = LENGTH*
  LENGTH: Comprimento do tapete anexo à máquina;
- *facility.ID.width = WIDTH*
  WIDTH: Largura do tapete anexo à máquina;
- *facility.ID.orientation = ORIENT*
  ORIENT: Orientação do tapete (vertical ou horizontal);
- *facility.ID.center.x = POSX*
  POSX: Coordenada x do centro do tapete na frame do simulador;

- *facility.ID.center.y = POSY*
  POSY: Coordenada y do centro do tapete na frame do simulador;
- *facility.ID.tool1 = TOOL1_ID*
  TOOL1_ID: Número identificador do tipo de ferramenta da 1ª ferramenta utilizada pela máquina;
- *facility.ID.tool2 = TOOL2_ID*
  TOOL2_ID: Número identificador do tipo de ferramenta da 2ª ferramenta utilizada pela máquina;
- *facility.ID.tool3 = TOOL3_ID*
  TOOL3_ID: Número identificador do tipo de ferramenta da 3ª ferramenta utilizada pela máquina;
- *facility.ID.alias = ALIAS*
  ALIAS: Nomenclatura para o tapete anexo à máquina.

O "ID" em cada um dos parâmetros da máquina ferramenta é um identificador único do equipamento.

2.2.12 *Pusher*

O *pusher*, como o próprio nome indica, permite empurrar uma peça para fora de um tapete. O tapete é considerado parte integrante do pusher e é comandado de forma idêntica ao tapete linear.

Visualmente, o pusher é cinzento e tem a forma de um T, como se pode ver na figura 2.29.

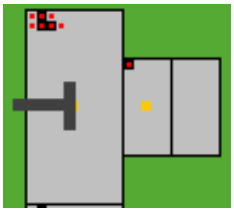

Figura 2.29: Pusher com tapete anexo, roller e mesa (simulador)

2.2.12.1 Controlo

O pusher utiliza dois comandos binários: um para o movimentar para a direita (sentido de empurrar a peça) e outro para o movimentar para a esquerda (sentido de retorno à posição de repouso). Tem associado dois sensores de fim de curso que indicam a posição limite de cada um destes movimentos.

De forma a prevenir a ocorrência de danos físicos ao equipamento, nenhum dos movimentos descritos deve ultrapassar os limites impostos pelos sensores. Este dano é representado através da alteração de cor das margens do tapete anexo ao *pusher* para vermelho.

É de notar que o tapete anexo ao pusher só deve ser movimentado quando este se encontra na posição de repouso.

A tabela 2.11 descreve os atuadores e sensores associados a este equipamento.

Tipo	Sigla	Nome
Comando Binário	mp	movimento no sentido positivo (XX ou YY)
Comando Binário	mm	movimento no sentido negativo (XX ou YY)
Comando Binário	pr	recolhe o pusher
Comando Binário	pe	extende o pusher
Sensor Binário	p	presença de peça
Sensor Binário	fr	fim de recolha do pusher
Sensor Binário	fe	fim de extensão do pusher

Tabela 2.11: Atuadores e sensores do Pusher

2.2.12.2 Ficheiro de configuração

A configuração do *pusher* é semelhante à configuração dos tapetes lineares, descrita na secção 2.2.6.3. É de notar que o parâmetro que indica o tipo de equipamento (*facility.ID.type*) toma o valor *"pusher"* em vez de *"conveyor"*, que é o valor atribuído para o tapete linear. Há ainda um novo parâmetro que permite configurar a velocidade do *pusher* (*configuration.pushspeed*). Este é um parâmetro global que afeta todos os *pushers*.

2.2.13 Robot 3D

O robot 3D é utilizado em células de montagem (empilhamento) de peças. Pode movimentar-se ao longo dos três eixos (XX, YY e ZZ) e possui uma garra para segurar em peças, permitindo transportá-las entre diferentes tapetes.

A sua movimentação está limitada a uma área que é definida no ficheiro de configuração (ver secção 2.2.13.3).

2.2.13.1 Representação Gráfica

Visualmente, o robot 3D é representado por um quadrado preto com duas linhas a atravessar o seu centro: uma linha horizontal e outra vertical. No seu canto superior esquerdo existem pequenos quadrados que representam os seus sensores e atuadores binários (descritos na secção seguinte). Os sensores (entradas) apresentam fundo preto, para se distinguirem dos atuadores (saídas). A cor destes quadrados é vermelha quando o sensor/atuador associado está inativo (FALSE) e verde quando está ativo (TRUE).

Sobre a aresta direita do quadrado que representa o robot 3D há também um pequeno quadrado branco. A posição deste quadrado sobre a aresta indica a posição segundo o eixo ZZ do robot 3D. O vértice superior desta aresta corresponde ao limite superior do eixo dos ZZ e o vértice inferior ao limite inferior. Esta componente gráfica permite adicionar uma perceção visual aos movimentos ao longo do eixo ZZ, para um simulador que é de duas dimensões.

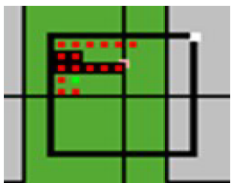

Figura 2.30: Robot3D no simulador

2.2.13.2 Controlo

O movimento do robot 3D ao longo dos três eixos (XX, YY e ZZ) é controlado por dois comandos binários para cada eixo, um para cada sentido de movimento.

Para o eixo ZZ existem dois sensores binários que indicam as duas posições extremas (em cima e em baixo). Para o eixo dos XX existem múltiplos sensores (tipicamente 2) que coincidem com o alinhamento correto com os tapetes/mesas que se encontram na célula onde o robot se encontra. Para o eixo dos YY existem também vários sensores que coincidem com o alinhamento correto com os tapetes/mesas da célula.

Para controlar a garra há apenas um atuador binário (0 -> abre a garra; 1 -> fecha a garra). Considera-se que um segundo após ser executado o comando de abertura ou fecho da garra esta se encontra aberta/fechada.

A garra dispõe de um sensor que indica a presença ou não presença de uma peça.

De forma a prevenir a ocorrência de colisões, todos os movimentos ao longo dos eixos XX e YY só podem ser efetuados quando a garra se encontra na posição superior do eixo ZZ. Para além disso, o movimento do robot é bloqueado quando este tenta ultrapassar os limites da área de operação (no plano XY), que estão definidos no ficheiro de configuração (ver secção 2.2.13.3). No caso do eixo dos ZZ, quando se força o movimento para além dos limites estabelecidos pelos dois sensores, o quadrado que representa o robot torna-se vermelho e o equipamento fica inutilizável.

Quando o simulador arranca, a posição do robot é aleatória (dentro da zona de operação) e desconhecida. É então necessário ter um procedimento de inicialização do mesmo, colocando-o numa posição conhecida.

A tabela 2.12 descreve todos os sensores e comandos associados ao robot 3D.

Tipo	Sigla	Nome
Comando Binário	xp	movimento no sentido positivo (XX)
Comando Binário	xm	movimento no sentido negativo (XX)
Comando Binário	yp	movimento no sentido positivo (YY)
Comando Binário	ym	movimento no sentido negativo (YY)
Comando Binário	zp	movimento no sentido positivo (ZZ)
Comando Binário	zm	movimento no sentido negativo (ZZ)
Comando Binário	g	Comando da garra
Sensor Binário	xp	Fim movimento no sentido positivo (XX)
Sensor Binário	xm	Fim movimento no sentido negativo (XX)
Sensor Binário	zp	Fim movimento no sentido positivo (ZZ)
Sensor Binário	zm	Fim movimento no sentido negativo (ZZ)
Sensor Binário	y1	Posição 1 do eixo YY
Sensor Binário	y2	Posição 2 do eixo YY
Sensor Binário	y3	Posição 3 do eixo YY
Sensor Binário	y4	Posição 4 do eixo YY
Sensor Binário	y5	Posição 5 do eixo YY
Sensor Binário	p	Presença de peça

Tabela 2.12: Atuadores e sensores de um Robot 3D

2.2.13.3 Ficheiro de Configuração

As características e disposição do robot 3D são também definidas no ficheiro de configuração através das seguintes entradas (cada entrada é uma linha no ficheiro de configuração):

- *facility.ID.type = portal3d*
  O tipo de equipamento é obrigatoriamente "portal3d";
- *facility.ID.width = WIDTH*
  WIDTH: largura da perímetro que limita a área que o robot pode ocupar (no plano XY);
- *facility.ID.height = HEIGHT*
  HEIGHT: altura da perímetro que limita a área que o robot pode ocupar (no plano XY);
- *facility.ID.center.x = POSX*
  POSX: Coordenada x do centro da área que o robot pode ocupar;
- *facility.ID.center.y = POSY*
  POSY: Coordenada y do centro da área que o robot pode ocupar;
- *facility.ID.sensorsx = SENSX*
  SENSX: número de sensores ao longo do eixo dos XX;
- *facility.ID.sensorsy = SENSY*
  SENSY: número de sensores ao longo do eixo dos YY;
- *facility.ID.alias = ALIAS*
  ALIAS: Nomenclatura para o robot3D.

2.2.14 Interação do Utilizador com o Rato

A interface do simulador permite ao utilizador interagir com a linha de produção através do rato.

É possível inserir peças em qualquer equipamento do simulador com tapete. Para isso é necessário clicar sobre o tapete pretendido com o botão direito do rato, selecionar o separador "Block" e clicar, com o botão esquerdo, sobre a opção com o tipo de peça que se pretende inserir. A figura 2.31 apresenta o menu de opções que permite inserir uma peça.

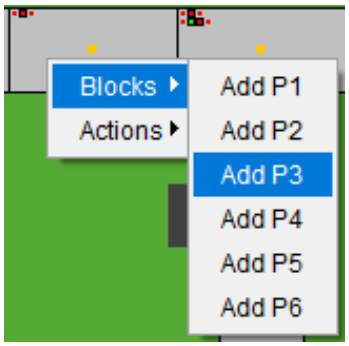

Figura 2.31: Menu de opções que permite inserir uma peça

É também possível remover peças da linha de produção. Para isso é necessário clicar sobre a peça pretendida com o botão direito do rato, selecionar o separador "Block" e clicar sobre a opção *"Remove"* com o botão esquerdo do rato. A figura 2.32 apresenta o menu de opções que permite remover uma peça.

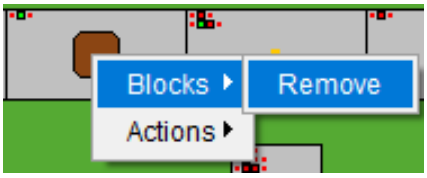

Figura 2.32: Menu de opções que permite remover uma peça

A interface do simulador também permite alterar o estado dos atuadores de qualquer equipamento da linha de produção. Para isso é necessário clicar sobre o equipamento pretendido com o botão direito do rato, selecionar o separador "Actions" e clicar, com o botão esquerdo, sobre a opção com o atuador que se pretende alterar. Isto faz com que a saída discreta Modbus (coil) associada a esse atuador altere o seu estado lógico (se for TRUE passa a FALSE, e vice-versa). A figura 2.33 apresenta o menu de opções com os atuadores de um tapete rotativo.

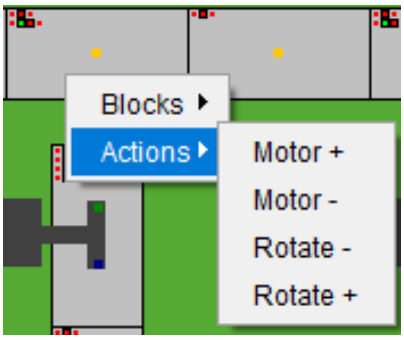

Figura 2.33: Menu de opções que permite atuar sobre um tapete rotativo

2.2.15 Arquitetura do Simulador

O simulador da linha de produção foi desenvolvido utilizando a linguagem de programação Java, que é uma linguagem baseada em classes e orientada a objetos.

A linguagem Java permite organizar as suas classes através de pacotes. Estes podem ser vistos como pastas onde são agrupadas classes que se relacionam de alguma forma.

O simulador é constituído por diversos pacotes e suas classes associadas. As subsecções que se seguem descrevem cada um destes pacotes e as respetivas classes desenvolvidas. A classe *Factory* (secção 2.2.15.6), pertencente ao pacote *"factory"*, é a classe principal do simulador, sendo responsável por ler o ficheiro de configuração e criar todos os objetos das classes existentes no simulador. A descrição desta classe ajuda a compreender o funcionamento interno geral do simulador.

2.2.15.1 Pacote *"modbus"*

O pacote *"modbus"* implementa o cliente Modbus (escravo) do simulador, para o qual se definem os diversos elementos Modbus (e.g. coils, entradas discretas) dos equipamentos do simulador, utilizados para a comunicação com o programa de controlo da linha de produção. Neste pacote apenas se inicializa o cliente Modbus, sendo que os elementos Modbus são adicionados pelas classes dos equipamentos.

O pacote é constituído por uma só classe, *"ModbusSlave"*, que utiliza uma biblioteca Modbus (*jamod*) e permite definir um escravo Modbus numa determinada porta, no endereço local (o programa de controlo e o simulador são executados na mesma máquina). A porta é indicada no ficheiro de configuração.

A figura 2.34 apresenta um diagrama UML (*Unified Modeling Language*) do pacote *"modbus"*, com os principais campos e métodos da sua classe *"ModbusSlave"*.

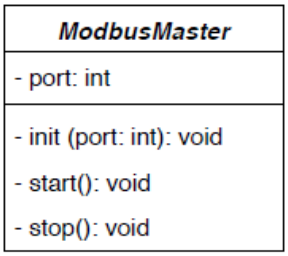

Figura 2.34: Diagrama UML do pacote *"modbus"*

O campo *"port"* corresponde à porta do cliente Modbus.

O método *init(port)* é apenas utilizado para definir o valor do campo *"port"*, o método *start()* permite definir e inicializar o cliente Modbus nessa porta e o método *stop()* permite parar o funcionamento desse mesmo cliente.

2.2.15.2 Pacote "block"

O pacote "block" é constituído por duas classes que permitem definir as peças do simulador: "BlockType" e "Block".

A figura 2.35 apresenta um diagrama UML deste pacote, com os principais campos e métodos das suas classes.

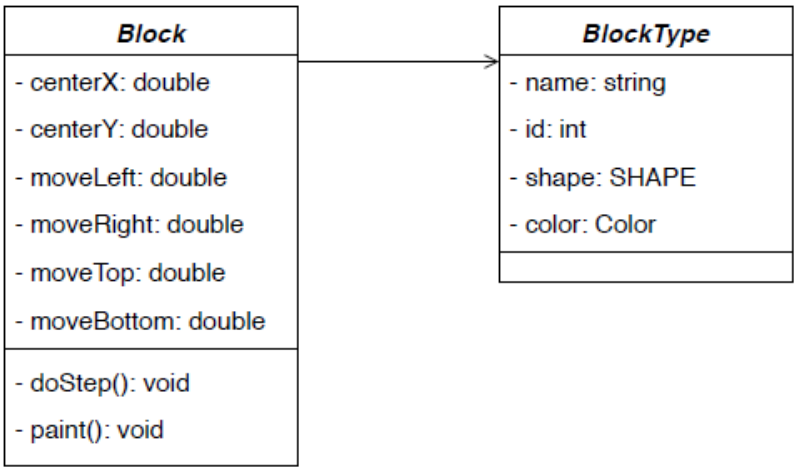

Figura 2.35: Diagrama UML do pacote *"block"*

A classe *"Block"* representa uma peça do simulador e é criado um objeto desta classe para cada peça. Os seus principais campos são:

- *centerX*: coordenada X do centro da peça na interface do simulador;
- *centerY*: coordenada Y do centro da peça na interface do simulador;

- *moveLeft*: permite simular a movimentação da peça para a esquerda; a coordenada X da posição central da peça é decrementada deste valor a cada ciclo de execução do simulador. Se o valor deste campo for superior a 0, a posição da peça fica mais à esquerda na interface do simulador. Este campo é controlado pelas classes dos equipamentos com tapetes, que no caso de haver uma peça presente e o motor que as move para a esquerda estar ativo, atribuem um valor superior a zero a este campo. O valor atribuído depende da velocidade dos motores dos tapetes, definida no ficheiro de configuração (uma velocidade superior implica valores superiores);
- *moveRight*: campo semelhante ao anterior, mas o seu valor é incrementado à posição da peça, de forma a simular movimentos para a direita.
- *moveTop*: semelhante aos campos *moveLeft* e *moveRight* mas o seu valor é decrementado da coordenada Y da posição da peça, simulando movimentos da peça para cima
- *moveBottom*: semelhante ao campo *moveTop*, mas o seu valor é incrementado à coordenada Y da posição da peça, simulando movimentos da peça para baixo;

A classe *"Block"* dispõe ainda de um campo que faz referência a um objeto da classe *"BlockType"*, que permite definir o tipo de peça. Os principais campos dessa classe são:

- *name*: nomenclatura para o tipo de peça (e.g. P1, P2);
- *id*: número único identificador do tipo de peça;
- *shape*: forma das peças deste tipo; o tipo de dados *SHAPE*, indicado no diagrama UML, é um objeto da classe Java *enum* que permite definir um grupo de constantes; neste caso, as constantes definidas representam as diferentes possibilidades para a forma das peças (arredondada, quadrada ou circular);
- *color*: cor das peças deste tipo; o tipo de dados *Color*, indicado no diagrama UML, corresponde a uma classe de uma biblioteca Java genérica, utilizada para a definição e manipulação de cores;

A classe *"Block"* dispõe de dois métodos de especial interesse: *doStep()* e *paint()*, que são chamados a cada ciclo de execução do simulador. O primeiro método permite atualizar a posição da peça na linha de produção. O segundo faz o desenho da peça na interface do simulador, em função do seu tipo e da posição calculada.

2.2.15.3 Pacote "facility"

O pacote *"facility"* é constituído pelas classes que definem os vários tipos de equipamentos.

A classe *"Facility"* é a classe principal deste pacote e define um equipamento de forma genérica. As restantes são subclasses de *"Facility"* e representam os vários tipos de equipamentos existentes.

A figura 2.36 representa um diagrama UML deste pacote, com os principais campos e métodos das classes constituintes.

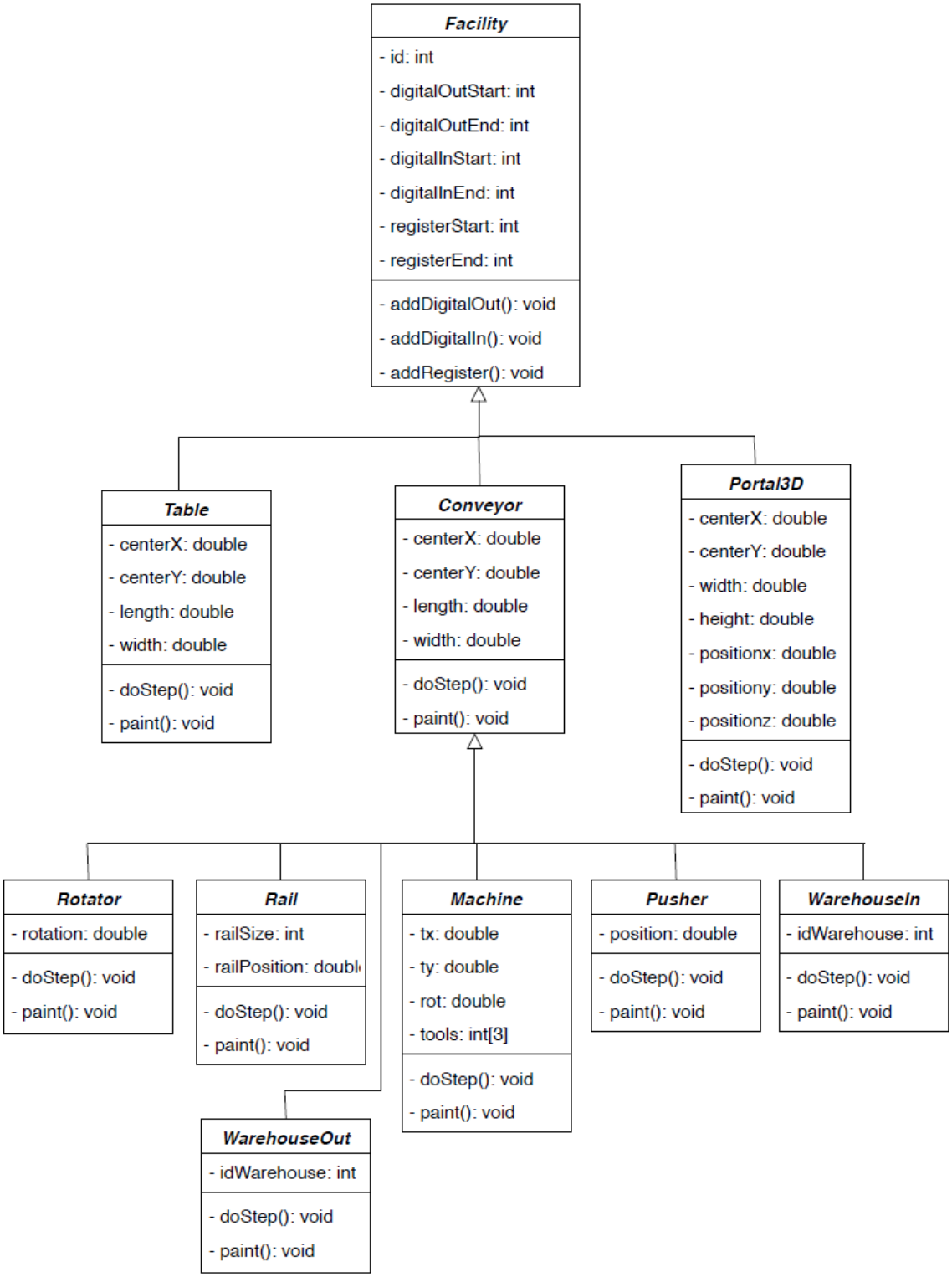

Figura 2.36: Diagrama UML do pacote *"facility"*

Os principais campos da classe genérica *"Facility"* são os seguintes:

- *id*: Identificador único do equipamento (número inteiro);

- *digitalOutStart*: endereço inicial das saídas discretas Modbus (coils) associadas ao equipamento;
- *digitalOutEnd*: endereço final das saídas discretas Modbus (coils) associadas ao equipamento;
- *digitalInStart*: endereço inicial das entradas discretas Modbus associadas ao equipamento;
- *digitalInEnd*: endereço final das entradas discretas Modbus associadas ao equipamento;
- *registerStart*: endereço Modbus inicial dos registos *holding* associadas ao equipamento;
- *registerEnd*: endereço Modbus final dos registos *holding* associadas ao equipamento;

Os três métodos da classe genérica *"Facility"* indicados no diagrama UML (*addDigitalOut()*, *addDigitalIn()* e *addRegister()*) permitem adicionar elementos Modbus aos diferentes espaços de endereçamento dos blocos de dados utilizados: saídas discretas (*coils*), entradas discretas e registos *holding*. O bloco de dados correspondente aos registos de entrada não é utilizado para qualquer equipamento.

As restantes classes são subclasses de "Facility" e representam cada um dos tipos de equipamentos existentes no simulador. Desta forma, para além de serem constituídas pelos campos e métodos da superclasse Facility, apresentam campos e métodos específicos do equipamento que representam. As subclasses existentes são as seguintes:

- *Conveyor*: implementa o tapete linear;
- *Rotator*: subclasse de *Conveyor* que implementa o tapete rotativo;
- *Rail*: subclasse de *Conveyor* que implementa o tapete deslizante;
- *Machine*: subclasse de *Conveyor* que implementa o tapete com máquina;
- *Pusher*: subclasse de *Conveyor* que implementa o *pusher*;
- *WarehouseIn*: subclasse de *Conveyor* que implementa o tapete onde são colocadas as peças que se pretende inserir no armazém;
- *WarehouseOut*: subclasse de *Conveyor* que implementa o tapete onde se colocam as peças provenientes do armazém;
- *Table*: implementa a mesa de trabalho;
- *Portal3D*: implementa o robot 3D.

A classe *"Conveyor"* é constituída pelos seguintes campos:

- *centerX*: coordenada X da posição central do tapete linear na interface do simulador;
- *centerY*: coordenada Y da posição central do tapete linear na interface do simulador;
- *length*: comprimento do tapete linear;
- *width*: largura do tapete linear.

A classe *"Rotator"*, que é uma subclasse de *"Conveyor"*, é constituída pelo seguinte campos adicional:

- *rotation*: indica o ângulo de rotação atual do tapete rotativo;

A classe *"Rail"*, que é uma subclasse de *"Conveyor"*, é constituída pelos seguintes campos adicionais:

- *railSize*: indica o comprimento dos carris sobre os quais o tapete deslizante se desloca;
- *railPosition*: indica a posição do tapete deslizante relativamente aos carris sobre os quais se desloca;

A classe *"Machine"*, que é uma subclasse de *"Conveyor"*, é constituída pelos seguintes campos adicionais:

- *tx*: representa a coordenada X da posição da torre de ferramentas; o valor deste campo vai desde -0.5 (torre recolhida) até 0.5 (torre estendida); para mover a torre para a direita incrementa-se este valor e para mover para a esquerda decrementa-se o valor;
- *ty*: representa a coordenada Y da posição da torre de ferramentas;
- *rot*: representa a posição de rotação da torre com as ferramentas da máquina;
- *tools*: vetor com os *IDs* das três ferramentas que constituem a máquina.

A classe *"Pusher"*, que é uma subclasse de *"Conveyor"*, é constituída pelo seguinte campo adicional:

- *position*: indica a posição do *pusher*; o seu valor vai desde a posição em que o *pusher* está totalmente recolhido (valor 0) até à posição em que o *pusher* está totalmente esticado, que coincide com o fim do tapete linear subjacente (valor igual à largura do tapete).

A classe *"WarehouseIn"*, que é uma subclasse de *"Conveyor"*, é constituída pelo seguinte campo adicional:

- *idWarehouse*: número identificador do armazém ao qual está associado.

A classe *"WarehouseOut"*, que é uma subclasse de *"Conveyor"*, é constituída pelo seguinte campo adicional:

- *idWarehouse*: número identificador do armazém ao qual está associado.

A classe *"Table"* é constituída pelos seguintes campos:

- *centerX*: coordenada X da posição central da mesa de trabalho na interface do simulador;
- *centerY*: coordenada Y da posição central da mesa de trabalho na interface do simulador;
- *length*: comprimento da mesa de trabalho;
- *width*: comprimento da mesa de trabalho.

Por fim, a classe *"Portal3D"* é constituída pelos seguintes campos:

- *centerX*: coordenada X do centro da área de ação do robot 3D na interface do simulador; esta é a área por onde o robot se pode movimentar;
- *centerY*: coordenada Y do centro da área de ação do robot 3D na interface do simulador;
- *width*: largura da área de ação do robot 3D;
- *height*: altura da área de ação do robot 3D;
- *positionx*: coordenada X da posição do robot 3D; esta posição é relativa ao ponto que se encontra no canto superior esquerdo do retângulo que define a área de ação do robot (ponto com positionx=0 e positiony=0); o seu valor vai desde 0 até *width*;

- *positiony*: coordenada Y da posição do robot 3D; esta posição é relativa ao ponto que se encontra no canto superior esquerdo do retângulo que define a área de ação do robot (ponto com positionx=0 e positiony=0); o seu valor vai desde 0 até *height*;
- *positionz*: coordenada Z da posição do robot 3D; o seu valor vai desde 0 até 1, sendo que 0 corresponde à posição em que o robot está mais baixo (junto à superfície dos tapetes) e 1 corresponde à posição em que o robot se encontra mais alto.

Cada objeto das classes descritas anteriormente possui um construtor (método que é executado quando se cria o objeto) que inicializa os elementos Modbus (e.g. coils, entradas discretas) associados ao respetivo equipamento. Para isso são utilizadas os métodos *addDigitalOut()*, *addDigitalIn()* e *addRegister()*, já descritos anteriormente para a classe genérica *"Facility"*.

Estas classes são também constituídas por um método *doStep()*, que atualiza o estado do equipamento a cada ciclo de execução do simulador. Este método é responsável por atualizar o estado das entradas modbus associadas aos sensores e por alterar o estado do equipamento (e.g. posição no simulador) em função do valor das saídas modbus, que podem ser *coils* ou registos *holding*. Por exemplo, um objeto da classe Conveyor (tapete linear) irá verificar se alguma peça se encontra suficientemente perto do sensor do tapete e atualizar o estado da entrada discreta Modbus associada a esse sensor. Por outro lado, se uma das saídas discretas Modbus (*coils*) associadas ao movimento do tapete estiver ativa, então qualquer peça que se encontre sobre o tapete será deslocada no respetivo sentido.

O método *paint()*, presente em todas as classes anteriormente descritas, é responsável por desenhar os equipamentos na interface do simulador em função do seu estado atual que, como já foi dito, é atualizado pelo método *doStep()*. É também chamado a cada ciclo de execução do simulador. Por exemplo, o método *paint()* do tapete rotativo terá de verificar o campo que indica a sua posição de rotação para o desenhar corretamente.

2.2.15.4 Pacote "transformation"

O pacote *"transformation* é constituído por duas classes: *"Tool* e *"Transformation*.

A figura 2.37 representa um diagrama UML deste pacote, com os principais campos das classes constituintes.

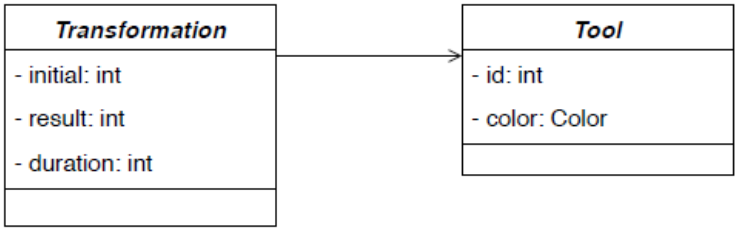

Figura 2.37: Diagrama UML do pacote *"transformation"*

A classe *"Transformation"* permite definir os vários tipos de transformação descritos no ficheiro de configuração. Os principais campos que as constituem são:

- *initial*: tipo de peça inicial;
- *result*: tipo de peça final, obtida quando a transformação é concluída;
- *duration*: Duração da operação de transformação.

A classe *"Transformation"* é ainda constituída por uma referência a um objeto da classe *"Tool"*, que implementa as ferramentas das máquinas. Esta referência permite indicar a ferramenta utilizada para cada transformação. A classe *"Tool"* dispõe dos seguintes campos:

- *id*: número inteiro que identifica o tipo de ferramenta;
- *color*: cor da ferramenta na interface do simulador;

2.2.15.5 Pacote "warehouse"

O pacote "warehouse" implementa o armazém automático do simulador e é constituído por quatro classes: "Order", "OrderIn", "OrderOut" e "Warehouse".

A figura 2.38 representa um diagrama UML deste pacote, com os principais campos e métodos das classes constituintes.

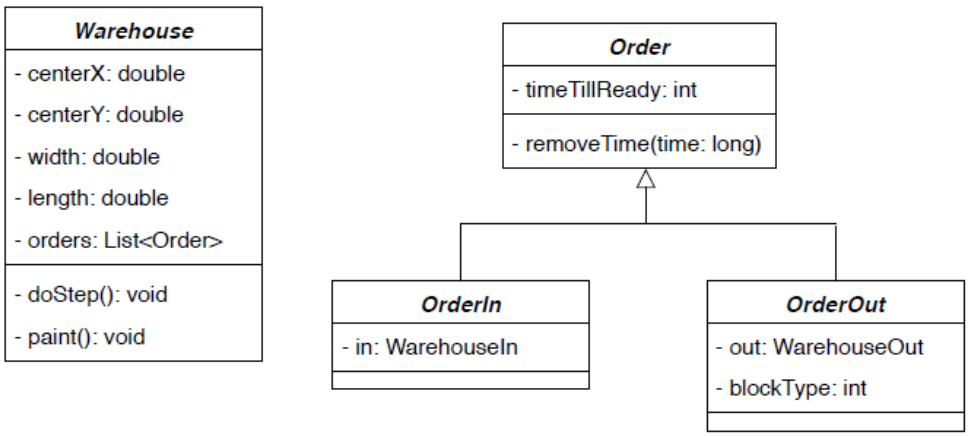

Figura 2.38: Diagrama UML do pacote *"warehouse"*

A classe "Order" implementa os pedidos de inserção ou retirada de peças do armazém automático para os tapetes de acesso. Contém apenas um campo inteiro, que corresponde ao tempo necessário até o pedido estar concluído. Este campo é decrementado pelo método *removeTime(time: long)*, que é chamado pelo método *doStep()* da classe "Warehouse" a cada ciclo de execução do simulador. O valor que é decrementado a cada ciclo corresponde a um valor aproximado do período de execução do simulador (duração de cada ciclo de execução). A peça só é inserida ou retirada do armazém quando o tempo chega a zero. Este intervalo simula o tempo que seria necessário para processar a peça num armazém automático real, e o seu valor é definido no ficheiro de configuração.

A classe "OrderIn" é uma subclasse de "Order" e implementa apenas os pedidos de entrada de peças no armazém. É constituída por um campo do tipo "WarehouseIn", que é a classe que implementa o tapete de entrada de peças no armazém. Este campo é utilizado para identificar o tapete do qual se retira a peça para o armazém.

A classe "OrderOut" é também uma subclasse de "Order" e implementa pedidos de retirada de peças do armazém para o tapete de saída. É constituída por um campo do tipo "WarehouseOut" que referencia o tapete de saída de peças do armazém para a linha de produção. Possui ainda um campo inteiro que indica o tipo de peça a ser retirada do armazém. O pedido fica concluído quando a peça aparece no tapete de saída.

A classe "Warehouse" implementa o armazém automático do simulador. Umas das suas tarefas é representar o armazém graficamente, através do método *paint()*, que é chamado a cada ciclo de execução do simulador. Esta classe possui também um método *doStep()* que faz a gestão dos pedidos de inserção e retirada de peças do armazém. Verifica se há pedidos e, se o tempo de duração do pedido tiver chegado ao fim, retira ou insere a peça no armazém, consoante o tipo de pedido. Os campos que constituem esta classe são:

- *centerX*: coordenada X da posição central do retângulo que representa o armazém;
- *centerX*: coordenada Y da posição central do retângulo que representa o armazém;
- *width*: largura do armazém;
- *length*: comprimento do armazém;
- *orders*: é uma lista com referências de todos os pedidos que ainda não foram executados; os tapetes de acesso ‘*"WarehouseIn"* e *"WarehouseOut"* são responsáveis por adicionar pedidos a esta lista quando os sinais Modbus respetivos são ativados pelo programa de controlo da linha de produção;

2.2.15.6 Pacote "factory"

O pacote *"factory"* contém apenas uma classe, *"Factory"*, que é a principal classe do simulador.

A figura 2.39 representa um diagrama UML desta classe, com os principais campos e métodos que a constituem.

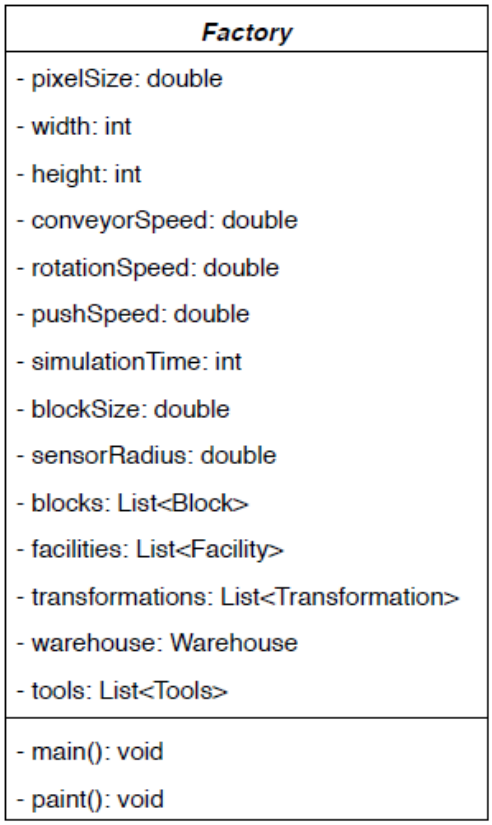

Figura 2.39: Diagrama UML do pacote *"factory"*

Esta classe contém o método *main()* e é responsável por ler o ficheiro de configuração e criar os objetos de todas as classes descritas anteriormente (peças, equipamentos, transformações...). Dispõe de um ciclo infinito que executa os métodos *doStep()* destes objetos, atualizando o estado lógico das componentes que representam.

Contém ainda um método *paint()* que faz o desenho do painel do simulador e chama os métodos *paint* de todos os objetos das classes que constituem o simulador. Desta forma, as diversas componentes gráficas associadas a estes objetos (peças, equipamentos..) são integradas no painel do simulador.

Os principais campos que constituem esta classe são:

- *pixelSize*: tamanho de cada pixel da interface do simulador; todos os valores de distâncias utilizados no simulador são divididos por este campo, obtendo-se assim o número de pixeis que essas distâncias representam na interface; definido no ficheiro de configuração;
- *width*: largura do painel de interface do simulador; definido no ficheiro de configuração;
- *height*: altura do painel de interface do simulador; definido no ficheiro de configuração;
- *conveyorSpeed*: velocidade de movimentação das peças ao longo de tapetes; definido no ficheiro de configuração;
- *rotationSpeed*: velocidade de rotação dos tapetes rotativos; definido no ficheiro de configuração;

- *pushSpeed*: velocidade de movimentação do *pusher*; definido no ficheiro de configuração;
- *simulationTime*: tempo de simulação, em milissegundos; a cada ciclo de execução o simulador fica a "dormir" durante este intervalo de tempo e assume-se que este é o tempo necessário para executar cada ciclo (o tempo de processamento real é desprezado); este campo é utilizado para qualquer movimento de peças ou equipamentos: multiplica-se a velocidade de movimentação (e.g. *conveyorSpeed*) por este valor e obtém-se a distância que deve ser percorrida em cada ciclo de execução; definido no ficheiro de configuração;
- *blockSize*: tamanho de cada peça do simulador; definido no ficheiro de configuração;
- *sensorRadius*: distância mínima para que qualquer sensor de presença ou posição fique ativo; definido no ficheiro de configuração;
- *blocks*: lista com todos os objetos da classe *Block*, que como já foi dito representam as peças do simulador; o tipo de campo *List* é uma classe pertencente a uma biblioteca Java genérica que permite criar listas com vários elementos, sendo cada elemento uma referência para um objeto da classe que for definida para a lista;
- *facilities*: lista com todos os objetos da classe *Facility*, que representam os equipamentos existentes;
- *transformations*: lista com todos os objetos da classe *Transformation*, que representam os tipos de transformação existentes;
- *tools*: lista com todos os objetos da classe *Tools*, que representam os tipos de ferramenta existentes;
- *warehouse*: objeto da classe *Warehouse*, que representa o armazém automático do simulador;

Capítulo 3

Solução proposta

Neste capítulo é feita uma proposta de solução, tendo em vista o cumprimento dos objetivos definidos para a dissertação.

Na primeira secção é feita uma descrição dos requisitos estabelecidos para cada objetivo. Na segunda secção é feita uma descrição da arquitetura geral e da arquitetura interna (alterações no código do simulador) da solução.

3.1 Análise de requisitos

Através das várias interações com o cliente da dissertação, foi estabelecido um conjunto de requisitos para cada objetivo definido.

As tabelas que se seguem apresentam os requisitos funcionais associados a cada um dos objetivos da dissertação:

a) Desenvolvimento do módulo de simulação do armazém automático;

b) Integração do conceito de código de barras na simulação das peças;

c) Desenvolvimento do conceito de avaria e reparação de equipamentos;

d) Integração de um painel com botões e indicadores luminosos no simulador;

e) Desenvolvimento de um servidor Modbus que disponha de funcionalidades que permitam agir sobre a linha de produção do simulador (e.g. inserir e remover peças) e obter informação sobre o seu estado (equipamentos e peças), permitindo criar algoritmos de avaliação automática de programas de controlo do simulador.

Tabela 3.1: Requisitos funcionais para o objetivo a)

Código	**Requisito**
a1	O armazém automático desenvolvido deve permitir simular o funcionamento do armazém automático da linha de produção flexível.

a2	O armazém automático deve ter um número de posições de armazenamento configurável (número de linhas e colunas configurável).
a3	As posições de armazenamento devem ter sensores de presença.
a4	A unidade de transporte deve poder deslocar-se segundo os eixos X, Y e Z, através de atuadores de movimento.
a5	Para o movimento segundo X, a unidade de transporte deve dispor de 3 sensores: um alinhado com as posições de armazenamento, um intermédio e um alinhado com os tapetes de acesso ao armazém.
a6	Para o movimento segundo Y, a unidade de transporte deve dispor de um sensor alinhado com cada uma das colunas do armazém.
a7	Para o movimento segundo Z, a unidade de transporte deve dispor de dois sensores para cada linha do armazém: um alinhado com a parte inferior de cada linha, outro alinhado com a parte superior.
a8	Deve haver uma linha que representa o carril sobre o qual a unidade de transporte se movimenta, ao longo do eixo Y.
a9	A unidade de transporte deve permitir deslocar peças entre o armazém e os tapetes de acesso da mesma forma que na linha de produção flexível.
a10	Para cada posição de armazenamento deve ser possível definir, no ficheiro de configuração, se a mesma começa com ou sem peça, e no caso de começar, definir qual o tipo de peça.
a11	Os sensores e atuadores do módulo do armazém automático devem estar associados a elementos Modbus (e.g. coils, entradas discretas) pertencentes ao servidor já existente, como acontece para os equipamentos da versão anterior do simulador.
a12	Os atuadores do módulo do armazém automático devem poder ser controlados manualmente, utilizando o rato.
a13	O estado dos sensores e atuadores do módulo do armazém automático deve ser representado visualmente, como acontece com os equipamentos da versão anterior (pequenos quadrados que mudam de cor consoante o estado lógico)

Tabela 3.2: Requisitos funcionais para o objetivo b)

Código	**Requisito**
b1	As peças devem passar a ter um código de barras associado.
b2	Deve ser possível definir o código de barras das peças inicializadas no armazém automático

Tabela 3.3: Requisitos funcionais para o objetivo c)

Código	**Requisito**
c1	Deve ser possível avariar ou reparar qualquer equipamento.
c2	A avaria de um equipamento deve provocar alguma alteração nítida na sua interface gráfica. A reparação deve reverter essa alteração e o equipamento deve voltar ao seu estado visual normal.
c3	A avaria e reparação de equipamentos deve poder ser gerada através de botões na interface do simulador.
c4	A avaria e reparação de equipamentos deve poder ser gerada de forma automática, através do servidor Modbus.

Tabela 3.4: Requisitos funcionais para o objetivo d)

Código	**Requisito**
d1	Deve existir no simulador um painel com botões e indicadores luminosos, definidos no ficheiro de configuração;
d2	A cor dos botões e indicadores luminosos deve ser configurável.
d3	Os botões e indicadores luminosos devem ter duas tonalidades diferentes, uma para quando estão ativos, outra para quando estão inativos.
d4	Deve haver botões do tipo *latch*, que ficam ativos ao ser pressionados e inativos ao ser novamente pressionados, e botões do tipo *pulse*, que só estão ativos enquanto estão a ser pressionados.
d5	Um botão do tipo *latch* deve poder ser configurado para estar associado à avaria de um equipamento, que deve ficar avariado quando o botão é ativo e ser reparado quando o botão fica inativo.
d6	Os botões devem estar associados a saídas discretas Modbus (coils), para que possam também ser controlados de forma automática e para que o programa de controlo (cliente Modbus) consiga ler o seu estado.
d7	Os indicadores luminosos devem estar associados a saídas discretas Modbus (coils), para que o programa de controlo (cliente Modbus) possa controlar o seu estado.
d8	Deve haver um botão *restart* que permita reiniciar a execução do simulador.

Tabela 3.5: Requisitos funcionais para o objetivo e)

Código	Requisito
e1	O servidor Modbus deve permitir inserir uma peça sobre qualquer equipamento com tapete.
e2	O servidor Modbus deve permitir remover uma peça que se encontre sobre algum equipamento.
e3	O servidor Modbus deve permitir gerar avarias e repará-las para qualquer equipamento.
e4	O servidor Modbus deve permitir obter informação sobre o estado de equipamentos.
e5	O servidor Modbus deve permitir obter informação sobre peças.

3.2 Arquitetura da solução

3.2.1 Arquitetura Geral

A figura 3.1 representa um diagrama da arquitetura de controlo da versão anterior do simulador (a preto) com os novas módulos a ser implementadas (a azul).

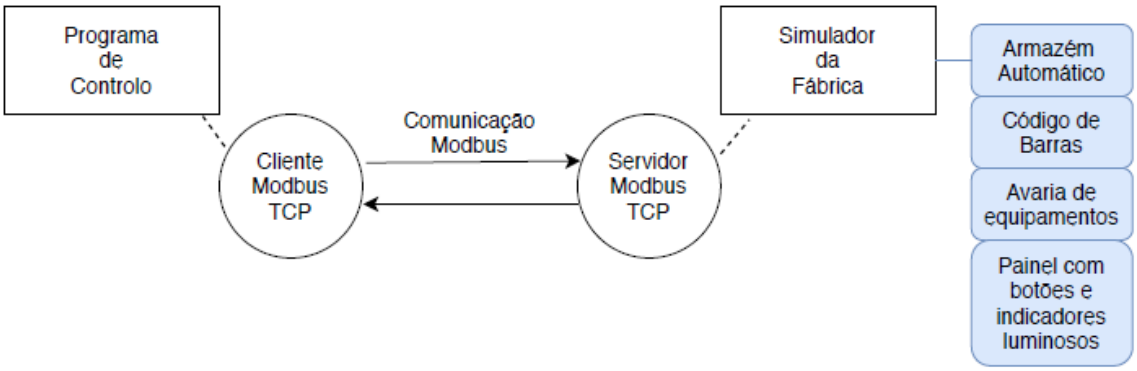

Figura 3.1: Arquitetura Geral da nova versão do simulador

Os novos módulos a ser implementadas no simulador são:

- Armazém Automático: módulo da linha de produção do simulador, que será constituído por um armazém onde se colocam peças, uma unidade de transporte e dois tapetes de acesso; deve simular corretamente o funcionamento do armazém da linha de produção flexível;
- Código de Barras: conjunto de caracteres que poderá ser atribuído a cada peça de forma única; apenas será utilizado pelo servidor automático;

- Avaria de equipamentos: será implementado o conceito de avaria e reparação de equipamentos; um equipamento poderá ser avariado através de um botão ou através do servidor automático;
- Painel com botões e indicadores luminosos: os botões irão estar associados a saídas discretas Modbus (coils) e serão utilizados para gerar avarias/reparar equipamentos ou para funcionalidades arbitrarias (e.g. botão que suspende as operações na linha de produção); os indicadores luminosos irão também estar associados a saídas discretas Modbus (coils) e poderão ser controlados pelo programa de controlo, também com finalidades arbitrarias (e.g. indicador luminoso a piscar informa que as operações na linha de produção estão suspensas);
- Novas Funcionalidades do Servidor Modbus: irão ser implementadas novas funcionalidades que permitem interagir com o simulador de forma automática (sem se utilizar o rato), através do servidor Modbus existente; irão permitir agir sobre a linha de produção (e.g. gerar avarias, inserir e remover peças) e obter informação sobre o seu estado (e.g. estado de equipamentos, informação sobre peças); a sua finalidade é automatizar o processo de avaliação de programas de controlo da linha de produção;

3.2.2 Arquitetura Interna

Nesta secção descreve-se, de forma geral, as alterações que se pretende fazer ao código do simulador, de modo a implementar os módulos descritos na secção anterior.

Como já foi dito anteriormente, o código do simulador está organizado em pacotes Java, que contêm diversas classes. As subsecções que se seguem apresentam as alterações que terão de ser feitas nas classes já existentes e as classes que devem ser desenvolvidas para implementar cada um dos novos módulos do simulador.

3.2.2.1 Armazém Automático

Para implementar o armazém deste módulo, irá manter-se a estrutura de classes já descrita anteriormente. O método *paint()* da classe "Warehouse" (pacote "warehouse") será alterado para que o armazém tenha uma interface gráfica que permita representar o armazém da linha de produção flexível. A esta classe terão de ser adicionados novos campos que permitam indicar o número de linhas e colunas de armazenamento, definidos através do ficheiro de configuração do simulador.

Para implementar a unidade de transporte do armazém, vai ser desenvolvida uma nova classe, que será integrada no pacote "facility" como uma subclasse de "Facility" (classe genérica para os vários tipos de equipamento). Assim como para os restantes equipamentos, a unidade de transporte irá dispor de um método *paint()* que desenha a sua interface gráfica e de um método *doStep()* que atualiza o estado do equipamento.

3.2.2.2 Código de Barras

Para implementar o conceito de código de barras, será adicionado um novo campo do tipo String à classe "Block", que implementa as peças do simulador.

3.2.2.3 Avaria de equipamentos

Para implementar o conceito de avaria de equipamentos, será adicionado um novo campo booleano à classe "Facility", que representa os equipamentos do simulador. Quando este campo tomar o valor booleano TRUE, será gerada uma avaria no equipamento respetivo, que ficará com um contorno de cor amarela. Para isso, o método *paint()* das classes que implementam cada tipo de equipamento (e.g. "Conveyor", "Portal3D") terá de verificar o estado lógico deste campo e alterar a cor de contorno para amarelo no caso de se verificar uma avaria.

3.2.2.4 Painel com botões e indicadores luminosos

Para implementar o painel com botões e indicadores luminosos vai ser criado um novo pacote com seguintes classes:

- Classe que implementa os botões: contém informação sobre os botões (e.g. cor, forma, avaria de equipamento), inicializa a saída discreta Modbus associada e atualiza o seu estado lógico;
- Classe que implementa os indicadores luminosos: contém informação sobre os indicadores luminosos (e.g. cor), inicializa a saída discreta Modbus associada e atualiza o seu estado lógico;
- Classe que integra os botões num painel que é adicionado à interface gráfica do simulador;
- Classe que integra os indicadores luminosos num painel que é adicionado à interface gráfica do simulador.

3.2.2.5 Novas funcionalidades do servidor Modbus

De forma a implementar as novas funcionalidades do servidor Modbus, vai ser criado um novo pacote Java, que deverá ser constituído por uma só classe. Essa classe irá definir métodos que implementam as novas funcionalidades e irá inicializar os elementos de memória do servidor Modbus (e.g. coils, registos) que permitem fazer uso das mesmas. Irá também dispor de um método *doStep()*, que é executado a cada ciclo de execução do simulador, e permite atender pedidos de utilização das suas funcionalidades.

Para funcionalidades que permitam agir sobre o simulador (e.g. inserir peça, gerar avaria), vão ser utilizadas saídas discretas Modbus (coils) para ler os pedidos (e.g. uma coil, ao ficar ativa, indica que se pretende inserir uma peça no simulador) e registos para obter informação adicional sobre esses pedidos (p.e. ID do equipamento onde se pretende inserir a peça).

Para funcionalidades que permitam obter informação sobre o estado da linha de produção (equipamentos e peças), vão ser utilizados registos que são preenchidos pelo simulador com essa informação.

Desta forma, qualquer programa externo ou equipamento que disponha de uma interface Modbus (cliente Modbus) pode fazer pedidos ou verificar o estado da linha de produção.

Capítulo 4

Implementação

Neste capítulo é feita uma descrição detalhada da implementação realizada de acordo com os objetivos e requisitos propostos.

4.1 Armazém Automático

4.1.1 Introdução

O armazém existente na versão anterior do simulador é muito rudimentar e não corresponde à realidade do armazém da linha de produção flexível, que se encontra no laboratório de Automação. Dispõe de uma simples interface gráfica com a representação das peças e de dois tapetes que, através de dois comandos, permitem inserir e retirar peças do armazém. Assim, toda a interação com o armazém está limitada a pedidos de armazenagem ou retirada de peças.

Pretende-se então criar um armazém mais elaborado, que possa simular corretamente o armazém do laboratório.

Nesta secção é feita, inicialmente, uma descrição geral dos vários elementos que constituem a célula do armazém. De seguida, são descritos de forma individualizada cada um desses elementos: a sua representação gráfica, a forma como são controlados e configurados (através do mesmo ficheiro descrito na versão anterior do simulador).

4.1.2 Descrição Geral

A célula do armazém automático desenvolvida para o simulador é flexível e configurável, e não apenas uma réplica rígida do armazém da linha de produção flexível. Permite definir parâmetros como o número de linhas e de colunas que o constituem, entre outros (p.e. coordenadas da posição central do armazém na interface do simulador). Estes parâmetros de configuração são descritos nas secções de cada um dos elementos que constituem a célula.

Esta célula é constituída pelos seguintes elementos:

- O próprio armazém: tem capacidade para armazenar um número configurável de peças (p.e. 24 peças, se tiver 8 colunas e 3 linhas);

- Estação de entrada de peças (**ST1/ST2**): tapete linear onde se colocam as peças que se pretende armazenar;
- Estação de saída de peças (**ST1/ST2**): tapete linear onde são colocadas as peças que se retiram do armazém para a linha de produção;
- Unidade de transporte: é constituída por uma transportadora (**AR**) montada sobre uma torre vertical e efetua o transporte de peças entre o armazém e as duas estações de acesso.

A figura 4.1 apresenta a representação gráfica desta célula no simulador. Neste caso o armazém está configurado para ter 3 linhas e 8 colunas.

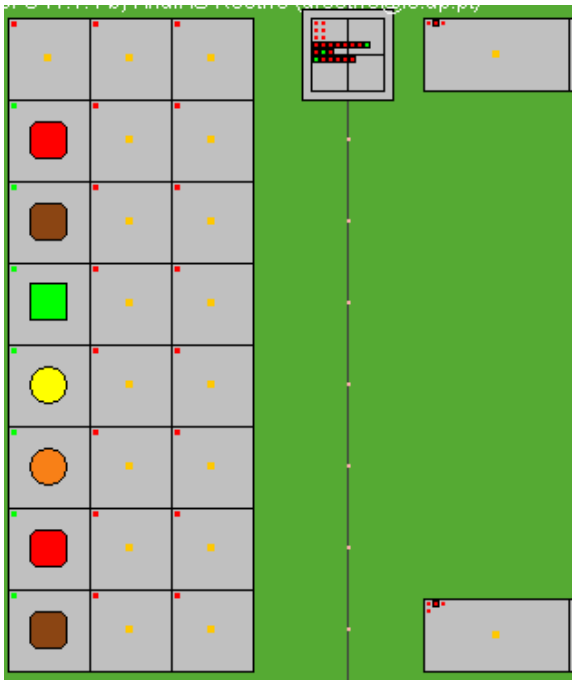

Figura 4.1: Célula do armazém automático no simulador (3 linhas, 8 colunas)

4.1.3 Armazém

O armazém do simulador permite armazenar peças, sendo organizado por linhas e colunas. Considera-se que as linhas têm direção paralela ao eixo dos YY e as colunas direção paralela ao eixos dos ZZ, como se pode ver no esquema da figura 4.3.

4.1.3.1 Representação Gráfica

A versão anterior do simulador apresenta uma vista de cima da linha de produção. Deste modo, se o armazém fosse representado com esta vista seria impossível ter visibilidade de todas as posições de armazenamento, sendo que apenas se conseguiriam ver as peças no topo de cada coluna. Para solucionar esta limitação, optou-se por rebater o eixo das colunas do armazém (ZZ)

sobre o eixo dos XX. Assim, as colunas passam a apresentar uma direção paralela ao eixo dos XX e todas as posições de armazenamento ficam visíveis.

Cada posição do armazém é delimitada por um retângulo e dispõe de um pequeno quadrado amarelo no seu centro, que representa um sensor de presença de peça. No armazém original da linha de produção flexível este sensor não existia, mas foi acrescentado para que fosse possível saber se existe uma peça em cada posição.

Existe também um pequeno quadrado no canto superior esquerdo de cada posição do armazém que representa o estado lógico do sensor de presença, ficando verde se o mesmo estiver ativo (com peça) e vermelho se estiver inativo (sem peça).

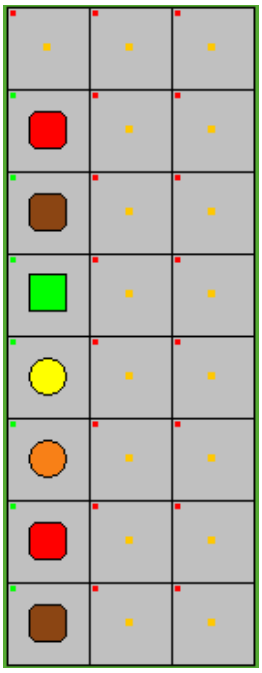

Figura 4.2: Armazém automático do simulador (3 linhas x 8 colunas)

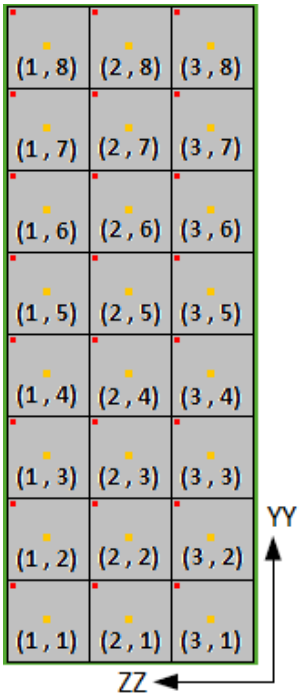

Figura 4.3: Esquema do armazém com coordenadas para cada posição (linha, coluna) e eixos

A altura dos retângulos que representam as posições de armazenamento depende do número de colunas que for definido e das posições dos tapetes de entrada/saída de peças do armazém. O simulador posiciona os sensores da primeira e da última coluna de forma a que fiquem alinhados com os sensores de presença dos tapetes de acesso. Por fim, distribui a distância vertical que existe entra os sensores dos tapetes de acesso pelas colunas, de modo a que tenham todas a mesma altura.

A largura das posições de armazenamento é definida no ficheiro de configuração.

4.1.3.2 Controlo

Cada posição do armazém pode armazenar uma peça simples e dispõe de um sensor binário que indica a presença ou não presença de uma peça. É de notar que não é permitido armazenar

peças compostas, que são peças constituídas por duas ou mais peças empilhadas (resultantes da célula de montagem). Estas peças continuam a ter de ser encaminhadas para uma das células de descarga.

Cada sensor binário de presença tem uma entrada discreta Modbus associada, que permite ler o seu estado lógico. A tabela 4.1 apresenta o conjunto de sensores de presença existentes num armazém com 2 linhas e 4 colunas.

Tabela 4.1: Sensores de um armazém com 2 linhas e 4 colunas

Tipo	**Sigla**	**Descrição**
Sensor binário	pln1col1	Presença de peça na linha 1, coluna 1 do armazém
Sensor binário	pln1col2	Presença de peça na linha 1, coluna 2 do armazém
Sensor binário	pln1col3	Presença de peça na linha 1, coluna 3 do armazém
Sensor binário	pln1col4	Presença de peça na linha 1, coluna 4 do armazém
Sensor binário	pln2col1	Presença de peça na linha 2, coluna 1 do armazém
Sensor binário	pln2col2	Presença de peça na linha 2, coluna 2 do armazém
Sensor binário	pln2col3	Presença de peça na linha 2, coluna 3 do armazém
Sensor binário	pln2col4	Presença de peça na linha 2, coluna 4 do armazém

4.1.3.3 Ficheiro de Configuração

O ficheiro de configuração permite definir os seguintes parâmetros para o armazém automático:

- *warehouse.ID.center.x = CENTERX*
  CENTERX: coordenada X da posição central do armazém; a coordenada Y não é necessária porque é calculada em função do número de colunas e das posições dos tapetes de acesso;
- *warehouse.ID.nlines = NLINES*
  NLINES: número de linhas do armazém;
- *warehouse.ID.ncolumns = NCOL*
  NCOL: número de colunas do armazém;
- *warehouse.ID.line.LN.column.COL.block = BLKID*
  LN: número da linha; COL: número da coluna;
  BLKID: id do tipo de bloco que será colocado na posição de armazenamento (LN, COL);
  Existe um parâmetro com este formato para cada posição do armazém que começa com peça. Se este não estiver definido para uma determinada posição ou se estiver definido com BLKID a zero, então essa posição é inicializada sem peça.

- *warehouse.ID.line.LN.column.COL.blockbarcode = BLKBC*
  LN: número da linha; COL: número da coluna; BLKBC: código de barras da peça (se houver peça).

O ID em cada um dos parâmetros descritos corresponde a um número que identifica o armazém. Como apenas se utiliza um armazém, o ID poderá tomar sempre o valor 1.

4.1.4 Estações de Entrada/Saída de Peças

A estação de entrada de peças no armazém automático (ST1) é um tapete para onde se encaminham as peças que se pretende armazenar. A estação de saída de peças do armazém (ST2) é também um tapete onde são colocadas as peças que se pretende retirar do armazém para a linha de produção.

Estes tapetes de acesso têm um funcionamento idêntico ao funcionamento dos tapetes lineares. Contudo, no ficheiro de configuração terão de ser definidos com um diferente tipo, uma vez que os sensores de presença da primeira e última coluna do armazém são alinhados especificamente com estes dois tapetes. O tipo de equipamento do tapete de entrada terá de ser "warehousein" (facility.ID.type = warehousein) e o tipo de equipamento do tapete de saída terá de ser "warehouseout".

4.1.5 Unidade de transporte

A célula do armazém automático da linha de produção flexível é constituída por uma unidade de transporte que permite transportar peças entre o armazém e os tapetes de acesso (inserir e retirar peças do armazém). Esta unidade corresponde a um *stacker*, que é constituído por uma torre vertical móvel e uma transportadora integrada nessa torre. A torre encontra-se sobre um carril e apenas se pode movimentar ao longo do mesmo. A transportadora pode movimentar-se verticalmente (ao longo da torre) e perpendicularmente à orientação da torre, para poder aceder às posições de armazenamento e aos tapetes de acesso.

Deste modo, o *stacker* tem movimentos ao longo dos três eixos (XX, YY e ZZ). Para o eixo dos YY o *stacker* move-se como um todo (torre + transportadora) e para movimentos ao longo de YY e ZZ só a transportadora é que se move.

A figura 4.4 apresenta o *stacker* da linha de produção flexível, para que melhor se perceba a sua constituição (transportadora e torre vertical) e a sua disposição na célula do armazém automático.

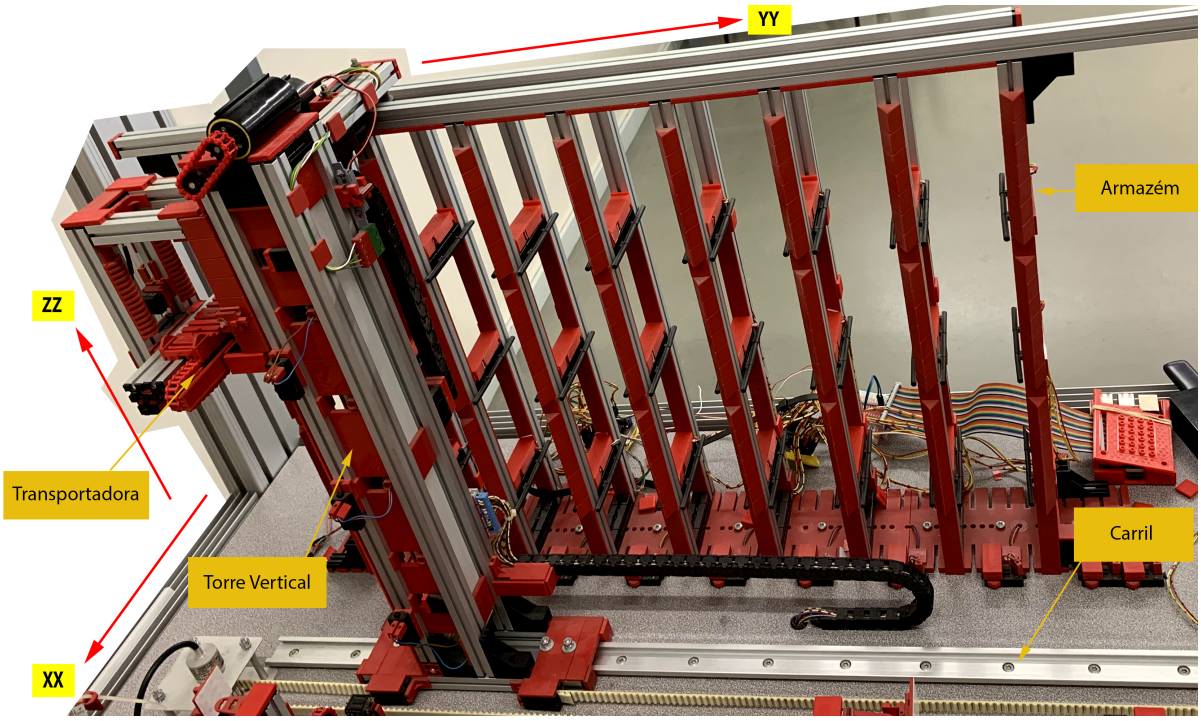

Figura 4.4: *Stacker* do armazém automático da linha de produção flexível

As secções que se seguem descrevem a unidade de transporte que foi implementada no simulador, incluindo a sua representação gráfica e a forma como é controlada.

4.1.5.1 Representação Gráfica

A transportadora do *stacker* no simulador é representada por um quadrado preto, que se encontra sobre um quadrado preenchido a cor cinzenta, a torre vertical. Para os movimentos ao longo do eixo dos YY a torre e a transportadora movem-se em conjunto e para movimentos ao longo do eixo dos XX apenas a transportadora se move. Os movimentos ao longo do eixo dos ZZ não são visíveis, já que o simulador apresenta uma vista de cima.

Existe também uma linha vertical que corresponde ao carril sobre o qual o *stacker* (torre + transportadora) se move, segundo o eixo dos YY. Ao longo desta linha existem pequenos quadrados que representam os sensores discretos que estão alinhados com cada uma das colunas do armazém.

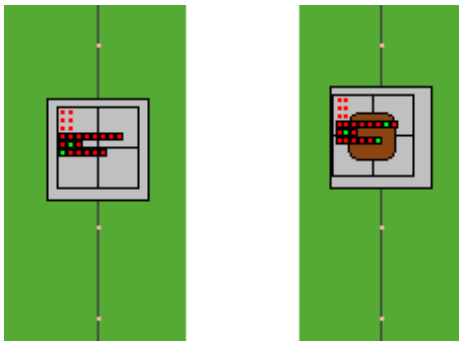

Figura 4.5: Unidade de transporte do armazém, com peça (direita) e sem peça (esquerda)

Sobre o quadrado preto que representa a transportadora existem também pequenos quadrados que representam o estado dos sensores e atuadores binários associados à unidade de transporte, como já acontecia com os restantes equipamentos do simulador. A sua cor fica verde quando estão ativos e vermelha quando estão inativos. Os quadrados dos sensores têm fundo preto para se distinguirem dos atuadores.

4.1.5.2 Controlo

A figura 4.6 apresenta um esquema de perfil e um esquema com vista de cima da célula do armazém automático, com os vários sensores que a constituem (excluindo os sensores de presença de peça nas posições de armazenamento, descritos anteriormente na secção sobre o armazém). Os sensores são representados através de pequenos quadrados azuis e apresentam nomenclatura.

A figura 4.7 é um exemplo de um armazém e um *stacker* na interface do simulador, com numeração das linhas/colunas do armazém e identificação dos sensores de posição do *stacker* ao longo do carril (y1, y2, y3...).

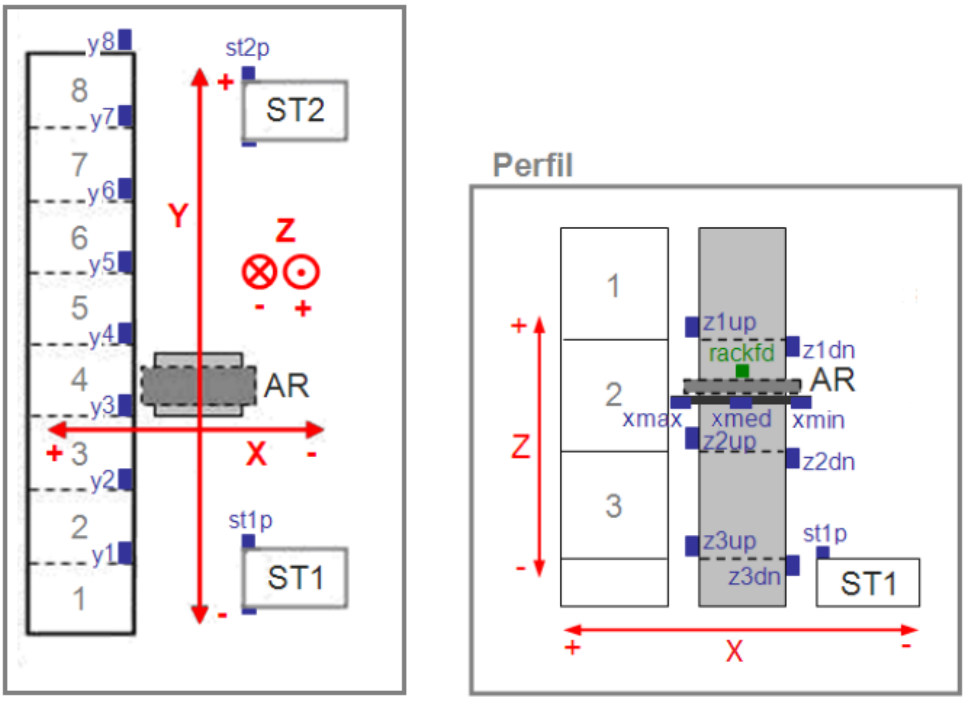

Figura 4.6: Esquema com sensores da célula do armazém automático

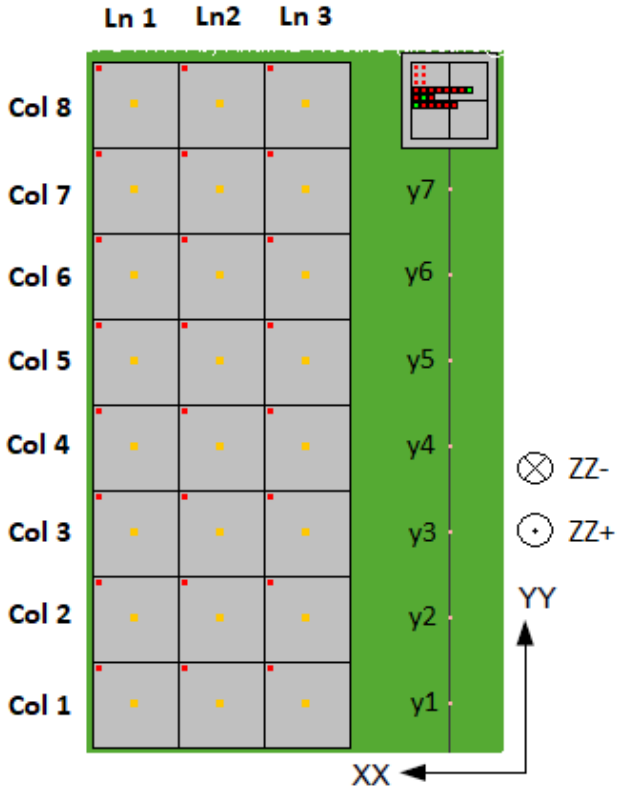

Figura 4.7: Armazém e unidade de transporte do simulador

A unidade de transporte (*stacker*) do armazém automático dispõe de dois atuadores discretos para cada direção de movimentação (XX, YY e ZZ), um para cada sentido de deslocação. Para movimentos ao longo do eixo dos XX e ZZ, apenas a transportadora (**AR**) se move, enquanto que para movimentos ao longo do eixo dos YY toda a unidade de transporte se move (transportadora + torre vertical).

Para movimentos ao longo do eixo dos ZZ existem múltiplos sensores discretos, que permitem indicar a posição da transportadora (**AR**, fig. 4.6) relativamente à torre vertical (retângulo cinzento). Existem dois sensores (*zup* e *zdn*) para cada linha do armazém (3 linhas e 6 sensores, neste caso). Cada sensor *zup* está posicionado ligeiramente acima do sensor *zdn* da mesma linha.

Para movimentos ao longo do eixo dos XX existem três sensores discretos que indicam a posição da transportadora (**AR**) ao longo de X: *xmin*, *xmed* e *xmax*. O sensor *xmax* corresponde à posição de alinhamento com o armazém. O sensor *xmed* corresponde à posição de repouso de **AR**, e a torre só poderá movimentar-se (ao longo do eixo dos YY) quando este estiver ativo. Por fim, o sensor *xmin* corresponde à posição de alinhamento com os sensores de presença dos tapetes de acesso (sensores *st1p* e *st2p* dos tapetes **ST1** e **ST2**).

A posição do sensor *xmax* na interface gráfica simulador não é sempre a mesma. Como já foi dito, o armazém está rebatido no plano XY e, deste modo, a posição deste sensor terá de depender da posição de **AR** segundo Z. Se **AR** estiver alinhado com algum dos sensores (*zup* ou *zdn*) associados a uma determinada linha do armazém, então o sensor *xmax* estará alinhado com o centro dessa linha. Por exemplo, observando a figura 4.7, se a transportadora estiver alinhada com o sensor *z3dn* ou *z3up*, então o sensor *xmax* estará alinhado com o centro linha 3 (Ln 3) do armazém. Se, por outro lado, a transportadora estiver alinhada com o sensor *z1dn* ou *z1up*, então *xmax* estará alinhado com o centro da linha 1 (Ln 1) do armazém.

Para movimentos ao longo do eixo dos YY (ao longo da linha que representa o carril), existe um sensor discreto de posição para cada coluna do armazém, como se pode ver na figura 4.7 (*y1, y2, y3...*). Estes sensores estão alinhados com o centro das colunas do armazém.

Há ainda um sensor discreto de presença de peça (*rackfd*) sobre a transportadora **AR**. O sensor fica ativo se **AR** tem peça e inativo se não tem peça.

A posição inicial da unidade de transporte é sempre a mesma. A transportadora encontra-se na posição de repouso segundo X (sensor *xmed* ativo) e na posição mais baixa do eixo Z (sensor *z3dn* na figura 4.6). Em relação à posição segundo Y, a unidade de transporte (torre + transportadora) encontra-se alinhada com a última coluna do armazém (sensor *y8*).

Na secção 4.1.5.4 são apresentados dois exemplos de como se insere e retira peças do armazém. Estes exemplos permitem perceber de que forma a unidade de transporte é controlada, através dos sensores e atuadores descritos anteriormente.

Os sensores descritos são controlados através de entradas discretas Modbus e os atuadores através de saídas discretas Modbus (coils). As tabelas que se seguem apresentam todos os sensores e atuadores discretos associados à unidade de transporte de um armazém com 3 linhas e 8 colunas.

Tabela 4.2: Sensores discretas da unidade de transporte do Armazém

Sensor	Função
ari_y1	AR na posição y1, no eixo Y
ari_y2	AR na posição y2, no eixo Y
ari_y3	AR na posição y3, no eixo Y
ari_y4	AR na posição y4, no eixo Y
ari_y5	AR na posição y5, no eixo Y
ari_y6	AR na posição y6, no eixo Y
ari_y7	AR na posição y7, no eixo Y
ari_y8	AR na posição y8, no eixo Y
ari_xmin	AR na posição xmin, no eixo X
ari_xmed	AR na posição xmed, no eixo X
ari_xmax	AR na posição xmax, no eixo X
ari_z1up	AR na posição z1up (pode colocar peça no 1º andar do armazém)
ari_z1dn	AR na posição z1dn (pode retirar peça do 1º andar do armazém)
ari_z2up	AR na posição z2up (pode colocar peça no 2º andar do armazém)
ari_z2dn	AR na posição z2dn (pode retirar peça do 2º andar do armazém)
ari_z3up	AR na posição z3up (pode colocar peça no 3º andar do armazém)
ari_z3dn	AR na posição z3dn (pode retirar peça do 3º andar do armazém)
ari_rackfd	AR tem peça

Tabela 4.3: Atuadores discretos (coils) da unidade de transporte do Armazém

Actuador	Função
Aro_xp	AR move-se no sentido X+
aro_xm	AR move-se no sentido X-
aro_yslow	Redução da velocidade de AR no eixo Y
aro_yp	AR move-se no sentido Y+
aro_ym	AR move-se no sentido Y-
aro_zp	AR move-se no sentido Z+
aro_zm	AR move-se no sentido Z-

4.1.5.3 Ações Ilegais

Foi definido um conjunto de ações ilegais que levam ao bloqueio da unidade de transporte, imobilizando-a. Este bloqueio é representado através da alteração de cor da transportadora para vermelho, como se pode ver na figura que se segue.

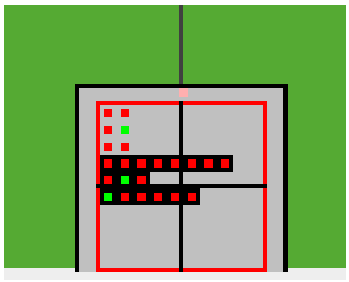

Figura 4.8: Unidade de transporte bloqueada

As ações consideradas ilegais são as seguintes:

- Mover a unidade de transporte para além dos sensores das colunas nas extremidades (e.g. y1 e y8, se tiver 8 colunas); a unidade de transporte não pode ultrapassar estas extremidades;
- Mover a transportadora segundo o eixo X para além das posições extremas *xmin* ou *xmax*;
- Mover a transportadora segundo o eixo X sem que qualquer dos sensores que estão alinhados com as colunas do armazém (y1, y2...) esteja ativado; a transportadora só poderá aceder ao armazém ou aos tapetes de acesso se estiver alinhada com alguma posição de armazenamento ou algum dos tapetes.

O bloqueio da unidade de transporte só ocorre se a ação ilegal se prolongar por mais de dois segundos. Este atraso é necessário porque a comunicação do programa de controlo com o simulador também acarreta atrasos que podem acionar a ação ilegal por breves momentos. Por exemplo, se o programa de controlo pretender mover a transportadora segundo o eixo X até *xmax*, poderá haver uma pequena deslocação adicional para além de *xmax* que decorre do tempo de resposta do programa de controlo e do tempo de comunicação.

4.1.5.4 Exemplos de Retirada e Inserção de Peça no Armazém

Os exemplos que se seguem baseiam-se na configuração do armazém e unidade de transporte da figura 4.7.

Imagine-se que se pretende inserir uma peça na posição (Ln 2 , Col 3), assumindo que esta se encontra no tapete de entrada de peças (tapete de baixo), pronta para ser inserida. O programa de controlo terá de seguir o seguinte conjunto de procedimentos, pela ordem apresentada:

1. Mover torre no sentido negativo do eixo dos YY (*ym*) até *y1*;
2. Descer **AR** (*zm*) até *z3dn*;
3. Mover **AR** no sentido negativo do eixo dos XX (*xm*) até *xmin*;
4. Subir **AR** (*zp*) até *z3up* (a peça é colocada em AR);
5. Esperar que o sensor *rackfd* fique ativo (**AR** com peça);

6. Mover **AR** até *xmed* (*xp*);
7. Mover torre até *y3* (*yp*);
8. Subir **AR** (*zp*) até *z2up*;
9. Mover **AR** até *xmax* (*xp*);
10. Descer **AR** (*zm*) até *z2dn* (peça fica armazenada);
11. Recolher **AR** até à posição de repouso (ativar *xm* até *xmed* ficar ativo).

Seguidamente é demonstrado o processo de retirada de uma peça do armazém. Para este exemplo, considera-se que se pretende retirar uma peça armazenada na posição (Ln 1 , Col 2). O conjunto de procedimentos é o seguinte:

1. Mover torre no sentido negativo do eixo dos YY (*yp*) até *y2*;
2. Subir **AR** (*zp*) até *z1dn*;
3. Mover **AR** para a esquerda (*xp*) até *xmax*;
4. Subir **AR** (*zp*) até *z1up* (peça é colocada em AR);
5. Esperar que o sensor *rackfd* fique ativo (**AR** com peça);
6. Mover **AR** para a direta (*xm*) até *xmed*;
7. Mover torre no sentido positivo do eixo dos YY (*yp*) até *y8*;
8. Mover **AR** para a direita (*xm*) até *xmin*;
9. Descer **AR** (*zm*) até *z3dn* (peça é colocada no tapete de saída);
10. Recolher **AR** até à posição de repouso (ativar *xp* até *xmed* ficar ativo).

4.1.5.5 Ficheiro de configuração

O ficheiro de configuração permite definir os seguintes parâmetros para a unidade de transporte do armazém:

- *facility.ID.type = warehouseCarrier*
  O tipo de equipamento terá de tomar o valor "warehouseCarrier";
- *facility.ID.warehouse = WHID*
  WHID: número inteiro identificador do armazém.
- *facility.ID.towersize = TOWERSIZE*
  TOWERSIZE: tamanho da torre da unidade de transporte, que tem forma quadrada;
- *facility.ID.carriersize = CARRSIZE*
  CARRSIZE: tamanho da transportadora da unidade de transporte (forma quadrada).

O parâmetro "ID" em cada uma das entradas do ficheiro descritas anteriormente corresponde a um número inteiro que identifica de forma única o equipamento.

4.1.6 Arquitetura Interna

De modo a implementar o novo armazém automático, fizeram-se alterações à classe "Warehouse" do pacote "warehouse", descrita anteriormente. Foram adicionados os seguintes campos:

- *whOutCenterY*: coordenada Y da posição central do tapete de saída de peças do armazém; este campo é utilizado para alinhar a primeira ou última coluna do armazém com o tapete de saída; se o valor deste campo for superior ao valor do campo *whInCenterY* (tapete de saída numa posição superior ao tapete de entrada), então o tapete de saída ficará alinhado com a última coluna; se o contrário se verificar, o tapete de saída ficará alinhado com a primeira coluna;
- *whOutCenterY*: coordenada Y da posição central do tapete de entrada de peças do armazém; este campo é utilizado alinhar a primeira ou última coluna do armazém com o tapete de entrada; se o valor deste campo for superior ao valor do campo *whInCenterY* (tapete de entrada numa posição superior ao tapete de saída), então o tapete de entrada ficará alinhado com a última coluna; se o contrário se verificar, o tapete de saída ficará alinhado com a primeira coluna;
- *nlines*: número de linhas do armazém;
- *ncolumns*: número de colunas do armazém.

O método *paint()* foi alterado para desenhar a nova interface gráfica do armazém e o método *doStep()* foi alterado para atualizar o estado das entradas discretas Modbus associadas aos sensores de presença de peças.

Para implementar a unidade de transporte do armazém automático foi desenvolvida uma nova classe,*"Carrier"*, como subclasse de "Facility" (classe que representa os equipamentos, descrita anteriormente).

A figura 4.9 apresenta um diagrama UML com os principais campos e métodos desta classe.

Os campos que constituem esta classe são os seguintes:

- *warehouseID*: ID do armazém;
- *centerX*: coordenada X da posição central inicial da unidade de trasnporta (torre + transportadora);
- *centerY*: coordenada Y da posição central inicial da unidade de transporte;
- *positionx*: valor do desvio segundo X da transportadora em relação à posição inicial; quando se pretende mover a transportadora segundo X incrementa-se ou decrementa-se este valor, dependendo do sentido de movimentação;
- *positionx*: valor do desvio segundo Y da unidade de transporte (torre + transportadora) em relação à posição inicial; quando se pretende mover a unidade de transporte segundo Y incrementa-se ou decrementa-se este valor, dependendo do sentido de movimentação; como já foi dito, a torre e a transportadora movimentam-se em conjunto ao longo do eixo Y;
- *positionz*: posição da transportadora segundo Z; o seu valor vai desde 0 (posição inferior) até 1 (posição superior); inicialmente a transportadora encontra-se na posição inferior,

junto ao solo (*positionz* = 0).

- *towerSize*: tamanho da torre, que tem forma quadrada;
- *carrierSize*: tamanho da transportadora, que tem forma quadrada.

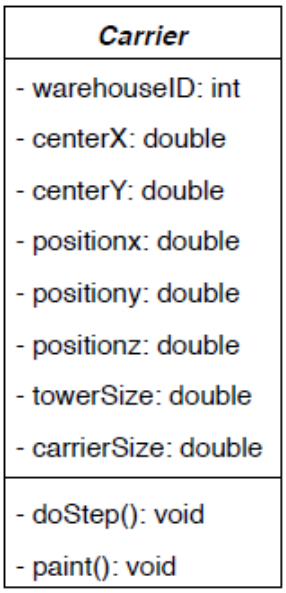

Figura 4.9: Diagrama UML da classe *Carrier*

Os principais métodos da classe *"Carrier"* são *paint()* e *doStep()*. O método *paint()* permite desenhar a unidade de transporte do armazém em função do seu estado atual, incluindo o carril sobre o qual se desloca e os quadrados que sinalizam o estado dos sensores e atuadores. O método *doStep()* permite atualizar o estado lógico dos sensores e respetivas entradas discretas Modbus e atualizar o estado geral da unidade de transporte, em função do estado lógico das saídas discretas Modbus, que representam atuadores. Estes métodos são chamados a cada ciclo de execução do simulador.

4.1.7 Validação e Testes

De forma a validar o funcionamento do armazém automático desenvolvido foram realizados vários testes.

Inicialmente foram feitos testes para verificar se as entradas e saídas (coils) Modbus discretas desta célula estavam bem configuradas. Foi utilizado o software *Codesys*, que é um ambiente de desenvolvimento para controladores programáveis que dispõe de uma interface Modbus. Permite usar linguagens de programação da norma IEC 61131-3 (ST, IL, SFC, LD, FBD). Após configurar o cliente Modbus que permite controlar o simulador, foi utilizada a linguagem SFC (Sequential Flow Chart) e ST (Structured Text) para testar as entradas e saídas Modbus discretas. Para as entradas, alterou-se o seu valor no servidor e verificou-se se a alteração era visível no *Codesys*.

Para as saídas, alterou-se o seu valor no *Codesys* e verificou-se no servidor do simulador se as alterações ocorreram.

Depois dos primeiros testes serem validados, foi necessário verificar se os sensores e atuadores associados às entradas e saídas Modbus estavam a funcionar. Estes testes foram feitos utilizando apenas a interface gráfica com o simulador. Para os sensores, bastou verificar-se se os pequenos quadrados que indicam o seu estado lógico apresentavam a cor correta (verde se TRUE, vermelho se FALSE). Para os atuadores, utilizou-se a interface com o rato para ativá-los e verificar se funcionavam (e.g. ativar o atuador *yp* da transportadora e verificar se esta se movimenta ao longo do eixo Y, no sentido correto).

Por fim, foram feitos dois testes de validação completos: um para a inserção de uma peça no armazém e um para a retirada de uma peça do armazém. Foi utilizado novamente o software *Codesys* e desenvolveram-se dois algoritmos de controlo, um para cada teste, com a linguagem de programação SFC. Para o teste de inserção de peça no armazém, era colocada manualmente (com o rato) uma peça no tapete de entrada e a unidade de transporte colocava-a na posição desejada. Para o teste de retirada de peça, a unidade de transporte movia uma peça de uma posição do armazém (definida no ficheiro de configuração) até ao tapete de saída.

4.2 Código de Barras na Simulação das Peças

O simulador da linha de produção flexível integra agora o conceito de código de barras. Cada peça da linha de produção poderá ter um código de barras associado, que permite identificá-la de forma única.

Os códigos de barras têm um formato alfanumérico e a sua dimensão (número de caracteres) é definida no ficheiro de configuração.

As peças que são inicializadas no armazém através do ficheiro de configuração podem ser definidas com um código de barras, que deve ser único para cada peça. Por outro lado, as peças que são inseridas manualmente, com o rato, ficam sem código de barras associado.

O código de barras é utilizado para diversas funcionalidades que foram implementadas no simulador. Estas funcionalidades permitem interagir com o simulador de forma automática (sem uso de rato), através de elementos de memória do servidor Modbus, e visam automatizar o processo de avaliação de programas de controlo da linha de produção.

Uma das funcionalidades permite inserir uma peça na linha de produção do simulador, de forma automática (sem uso de rato). É utilizado um conjunto de registos *holding* Modbus para que o cliente Modbus possa informar o simulador (servidor Modbus) sobre o código de barras da peça que pretende inserir. O número de registos Modbus alocados para o código de barras depende do número de caracteres, definido no ficheiro de configuração. Cada registo tem 2 bytes e permite armazenar o código ASCII de dois caracteres (cada caractere ASCII necessita de 1 byte). Para esta funcionalidade, existe ainda um conjunto de registos que permitem indicar o ID do equipamento sobre o qual se pretende inserir a peça e uma saída discreta Modbus (*coil*) que informa o simulador que o pedido está pronto para ser processado.

De seguida apresenta-se um exemplo de uma sequência de interações entre um cliente Modbus e o simulador (servidor Modbus) que permite demonstrar a inserção de uma peça:

1. O cliente preenche o conjunto de registos *holding* que permitem indicar o código de barras da peça;

2. O cliente preenche o conjunto de registos *holding* que permitem indicar o ID do equipamento onde se pretende inserir a peça;

3. O cliente ativa a saída discreta que informa que o pedido de inserção de peça está pronto;

4. O simulador (servidor), ao verificar que esta saída discreta foi ativa, insere a peça com o código de barras indicado, sobre o equipamento indicado (lê os registos respetivos).

Uma outra funcionalidade que foi implementada permite obter informações sobre qualquer peça (e.g. ID do equipamento onde se encontra, tipo atual da peça) e o código de barras é utilizado para identificar a peça pretendida. Assim como na inserção de peças, é utilizado um conjunto de registos *holding* para que o cliente possa indicar o código de barras da peça sobre a qual pretende obter informação. Há ainda um registo *holding* (registo principal) que possui 2 bits para interação entre o cliente e o simulador (servidor) e os restantes bits para preencher com informação sobre a peça. Um dos bits de interação é ativo pelo cliente para informar que o pedido está pronto para ser processado e o outro bit é ativo pelo servidor para informar que a informação sobre a peça pode ser lida.

De seguida apresenta-se um exemplo de uma sequência de interações entre um cliente Modbus e o simulador (servidor Modbus) que permite demonstrar esta funcionalidade:

1. O cliente preenche o conjunto de registos *holding* que permitem indicar o código de barras da peça;

2. O cliente ativa o bit do registo principal que informa o simulador que o pedido está pronto para ser processado;

3. O simulador (servidor), ao verificar que o bit de pedido foi ativo, preenche os bits de informação do registo principal com a informação sobre a peça com o código de barras indicado (lê os registos respetivos);

4. O simulador (servidor), após ter preenchido a informação da peça, ativa o bit de resposta do registo principal;

5. O cliente, ao verificar que o bit de resposta foi ativo, lê a informação sobre a peça pretendida.

Qualquer programa externo ou equipamento que disponha de uma interface de cliente Modbus pode utilizar as funcionalidades descritas. Na secção 4.5 é feita uma descrição mais detalhada destas funcionalidades.

4.2.1 Ficheiro de Configuração

No ficheiro de configuração é definido o número de caracteres que cada código de barras terá de apresentar. A entrada (linha) no ficheiro que define este parâmetro é a seguinte:

- *block.barcode.numchars = N*
  N: número de caracteres do código de barras;

4.2.2 Arquitetura Interna

De forma a implementar o conceito de código de barras nas peças, foi adicionado um novo campo (*"barcode"*) do tipo *String* à classe *"Block"*, que como já foi dito representa as peças do simulador. O construtor desta classe, que é o método executado quando se criam novos objetos, dispõe agora de um parâmetro com o código de barras da peça. As peças que são adicionadas sem indicação do código de barras ficam com um valor "0" para esse campo, que significa que não têm código de barras atribuído.

4.2.3 Validação e Testes

De forma a validar a implementação do código de barras para peças, foram realizados dois testes.

O primeiro consistiu em verificar se as peças inicializadas no armazém automático através do ficheiro de configuração apresentavam o código de barras especificado. Para isto, foi apenas necessário imprimir na consola o campo de dados com o código de barras para os objetos das peças inicializadas no armazém.

O segundo teste apenas foi realizado após o desenvolvimento do cliente e servidor automáticos, descritos em detalhe na secção 4.5. Uma das funcionalidades do servidor automático, que faz parte do simulador, é inserir peças com determinado código de barras na linha de produção. Através do cliente (programa externo) desenvolvido para interagir com este servidor e utilizar as suas funcionalidades, fizeram-se testes de inserção de peças e verificou-se se as peças apresentavam o código de barras pretendido. Esta verificação foi feita de forma idêntica ao primeiro teste, imprimindo na consola o campo de dados com o código de barras da peça inserida.

4.3 Avaria e Reparação de Equipamentos

Foi implementado o conceito de avaria de equipamentos no simulador. Desta forma, é possível simular uma situação de avaria e verificar como reage o programa de controlo, que deverá deixar de utilizar o equipamento avariado. Quando o equipamento deixa de estar avariado, pode voltar a ser utilizado normalmente.

Inicialmente, a avaria foi implementada de forma a bloquear o funcionamento do equipamento. Deste modo, qualquer tentativa de comandá-lo através dos seus atuadores não iria funcionar. Contudo, optou-se por deixar que os equipamentos continuassem a funcionar normalmente, passando para o programa de controlo a responsabilidade de não utilizar equipamentos avariados.

4.3.1 Representação Gráfica

Visualmente, a avaria é sinalizada por um contorno de cor amarela envolvendo o equipamento em questão, como se pode ver na figura seguinte.

Figura 4.10: Tapete Linear com avaria (lado esquerdo) e sem avaria (lado direito)

4.3.2 Controlo

O controlo das avarias pode ser feito de duas formas: na própria interface do simulador, através de botões, ou através do servidor Modbus, utilizando um conjunto de elementos Modbus (coils e registos) que foram adicionados para gerar avarias em equipamentos de forma automática.

O novo simulador dispõe de um painel de botões, que são definidos no ficheiro de configuração e podem ser associados à avaria de equipamentos específicos. Estes botões permitem ao utilizador simular a avaria de equipamentos de forma rápida e não automática. Quando o botão é pressionado com o rato gera-se uma avaria no equipamento associado. Se o mesmo botão voltar a ser premido, o equipamento é reparado e poderá voltar a ser utilizado. Estes botões são usados pelo utilizador do simulador para verificar se o programa de controlo respeita a avaria de equipamentos, deixando de utilizá-los. O painel de botões é descrito em maior detalhe na secção 4.4.

O simulador dispõe também de uma nova funcionalidade que permite gerar avarias de forma automática, através de elementos de memória do servidor Modbus. Os elementos que foram adicionados ao servidor Modbus para implementar as avarias são os seguintes:

- Conjunto de registos *holding* utilizados para indicar a nomenclatura ou o ID do equipamento que se pretende avariar; cada registo permite armazenar informação de dois caracteres e o número de registos necessários depende do número máximo de caracteres permitido para nomes de equipamentos, que é definido no ficheiro de configuração;
- Saída discreta (coil) que é ativa para informar o simulador que se pretende gerar uma avaria no equipamento especificado nos registos descritos anteriormente;

De seguida apresenta-se um exemplo de uma sequência de interações entre um cliente Modbus e o simulador (servidor Modbus) que permite demonstrar a avaria automática de equipamentos:

1. O cliente preenche o conjunto de registos *holding* que permitem indicar o ID do equipamento que se pretende avariar;

2. O cliente ativa a saída discreta que informa o simulador (servidor) que o pedido de avaria está pronto para ser processado;

3. O simulador (servidor), ao verificar que esta saída discreta foi ativa, gera uma avaria no equipamento especificado pelos registos *holding*.

Qualquer programa externo ou equipamento que disponha de uma interface de cliente Modbus pode gerar avarias em equipamentos.

4.3.3 Arquitetura Interna

De forma a implementar a avaria de equipamentos, foi adicionado um novo campo booleano *"malfunction"* à classe *"Facility"*, que é a super classe que representa os diversos tipos de equipamento. Em cada uma das subclasses específicas dos vários tipos de equipamento (e.g. tapete, máquina, pusher) é feita uma verificação do estado de *"malfunction"* no método *paint()*, que é responsável por desenhar o equipamento. Se este campo estiver ativo, o equipamento encontra-se com avaria e a sua margem é desenhada com cor amarela. Se estiver inativo, o equipamento é desenhado com margem de cor preta.

Como já foi dito, o controlo das avarias pode ser feito através da interface do simulador (com botões) ou através do servidor Modbus. Para ambas as formas de controlo a avaria ou reparação de um equipamento é conseguida alterando-se o valor do campo *"malfunction"* do objeto correspondente ao equipamento que se pretende avariar ou reparar.

A arquitetura interna da funcionalidade do simulador que permite gerar avarias através do servidor Modbus é descrita na secção 4.5.7.

4.3.4 Validação e Testes

Foram feitos diversos testes para validar o bom funcionamento das avarias e das diferentes formas de as gerar, seja através de botões ou através do servidor Modbus.

Numa primeira fase foi apenas testado o comportamento interno da avaria, ativando diretamente o campo booleano *malfunction* dos vários tipos de equipamento e verificando se a sua margem mudava de cor para amarelo.

Posteriormente testaram-se botões que geram avarias. Como já foi dito, cada botão pode ter um equipamento associado, através do ficheiro de configuração, que permanece avariado sempre que o botão estiver ativo. Após configurar os botões, verificou-se se era gerada uma avaria no equipamento respetivo quando os mesmos eram pressionados. A avaria era visível por alteração da cor da margem do equipamento para amarelo. Verificou-se também se o equipamento voltava ao estado normal (sem avaria) quando o botão era novamente pressionado.

Por fim, fizeram-se testes para verificar se era possível gerar avarias de forma automática, através do servidor Modbus do simulador. Os primeiros testes foram feitos executando diretamente as funções do simulador que permitem avariar e reparar equipamentos e verificando se os equipamentos eram avariados e reparados devidamente. Posteriormente foram também realizados testes para verificar se as funções eram executadas corretamente através de pedidos Modbus de um programa externo (cliente Modbus).

4.4 Painel de Botões e Indicadores Luminosos

Foi desenvolvido um painel de botões e indicadores luminosos que permite criar uma nova interface de interação com o operador, que poderá ativar/desativar botões e verificar o estado lógico de indicadores luminosos.

Visualmente, a nova versão do simulador é constituída por um painel vertical de botões, um painel vertical de indicadores luminosos e a linha de produção, organizados por esta ordem, da esquerda para a direita.

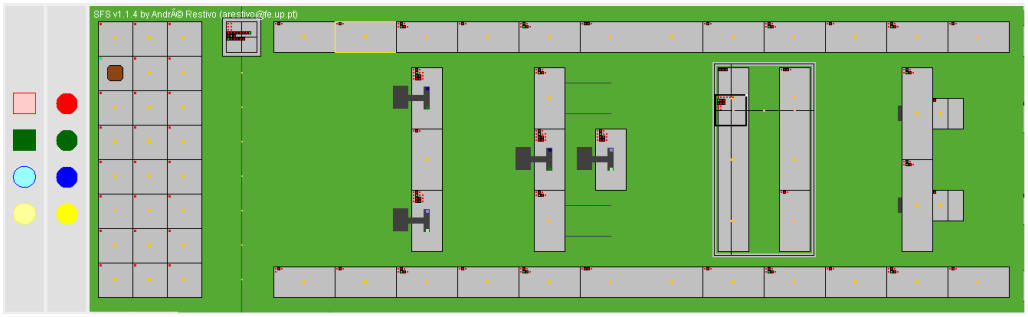

Figura 4.11: Simulador com painel de botões e indicadores luminosos

4.4.1 Botões

O painel de botões pode ser utilizado pelo operador do simulador para diversas finalidades, como por exemplo:

- Botão de Emergência que ao ser ativado indica ao programa de controlo que deve cessar todas as operações do simulador;
- Botão que permite gerar uma avaria num equipamento ao qual está associado.

Os botões são ativos através do clique do rato. Há dois tipos de botões:

- Botão "Pulse": é ativo quando o rato é pressionado e fica inativo quando o clique é libertado;
- Botão "Latch": é ativo quando o rato é pressionado e permanece ativo até que seja pressionado novamente.

4.4.1.1 Representação Gráfica

Visualmente, os botões podem apresentar duas formas, quadrada e circular, e as seguintes cores: azul, vermelho, verde e amarelo.

As cores disponíveis são limitadas porque é necessário haver um tom de cor menos vivo para quando o botão está inativo e um tom de cor mais vivo para quando está ativo. Foi testado um mecanismo que determinava as tonalidades de qualquer cor para botões ligados e desligados mas verificou-se que para certas cores a diferença de tom não era suficiente. Assim, cada cor foi

formatada individualmente e será necessário adicionar novas cores ao código do simulador para que possam ser utilizadas. Na versão atual do simulador estão disponíveis as seguintes cores: vermelho, amarelo, azul e verde.

A figura que se segue apresenta um painel com quatro botões de formas e cores diferentes, em ambos os estados lógicos (ligado e desligado).

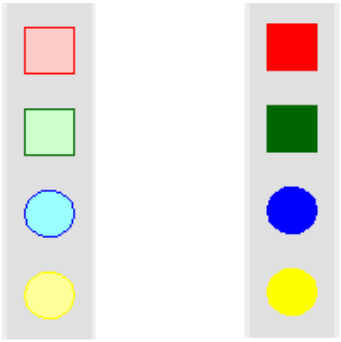

Figura 4.12: Painel com botões ligados (lado direito) e desligados (lado esquerdo)

4.4.1.2 Controlo

Cada botão está associado a uma entrada discreta do servidor Modbus do simulador. Esta entrada contém o estado lógico do botão, estando ativa quando o botão está ligado e inativa quando está desligado.

Desta forma, o programa de controlo poderá ler o estado lógico de qualquer botão do simulador. Poderá, por exemplo, verificar o estado de um botão de emergência e cessar todas as operações da linha de produção quando o mesmo for pressionado.

4.4.1.3 Avarias

Os botões podem também estar associados a avarias, sendo indicado no ficheiro de configuração o *ID* do equipamento que se pretende avariar.

Quando o botão é ativo gera-se uma avaria no equipamento associado. Quando volta a ficar inativo, o mesmo equipamento é reparado. Deste modo, fará mais sentido utilizar botões do tipo *"Latch"* para avarias, que ficam ativos ao serem pressionados e inativos ao serem novamente pressionados. Os botões do tipo *"Pulse"* não são muito adequados já que para manter um equipamento avariado é necessário manter o rato premido, e soltá-lo apenas quando se pretende repará-lo.

4.4.1.4 Botão *Restart*

Foi implementado um botão *Restart* que permite reiniciar a execução do simulador. Desta forma, quando o utilizador pressiona o botão, o simulador é encerrado e volta a ser inicializado.

Este botão foi implementado para simplificar a interação do utilizador com o simulador. Na versão anterior, cada vez que o utilizador quisesse reiniciar o simulador, tinha de fechar a janela da sua interface, ir ao diretório com o seu executável e iniciá-lo novamente.

A figura 4.13 apresenta um painel de botões com o botão *restart* incluído. Este botão apresenta um tom cinzento escuro, um contorno azul e tem a expressão "rst" escrita sobre o seu preenchimento. Desta forma, é facilmente distinguível dos restantes botões.

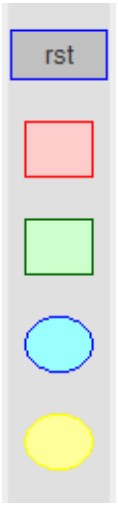

Figura 4.13: Painel de botões com o botão *restart* incluído

Ao contrário dos restantes botões, não é possível configurar a cor e a forma do botão *restart*. Apenas é possível escolher se se pretende ou não utilizá-lo. Para utilizá-lo é necessário definir a seguinte entrada no ficheiro de configuração:

- *button.restart = true*

Se esta entrada não for definida ou se o parâmetro que a define tomar outro valor que não *"true"*, o botão *restart* não é integrado no painel de botões.

4.4.1.5 Ficheiro de Configuração

Os botões são definidos no ficheiro de configuração através das seguintes entradas (cada entrada é uma linha no ficheiro):

- *button.ID.color = COLOR*
  ID: número inteiro que identifica de forma única o botão;
  COLOR: cor do botão em inglês (e.g. yellow, green)
- *button.ID.type = TYPE*
  TYPE: tipo de botão, pode ser "pulse" ou "latch";
- *button.ID.shape = SHAPE*
  SHAPE: forma do botão em inglês, pode ser "square" (quadrado) ou "circle" (circular);

- *button.ID.alias = ALIAS*
  ALIAS: nomenclatura para o botão (parâmetro opcional);
- *button.ID.malfunctionID = MALFUNCTIONID*
  MALFUNCTIONID: identificador do equipamento que será avariado e reparado através do botão; este parâmetro é opcional.

4.4.2 Indicadores Luminosos

Foi também desenvolvido um painel com indicadores luminosos, junto ao painel de botões. Estes indicadores podem ser utilizados para diversas finalidades, como por exemplo:

- Indicador luminoso que quando pisca informa o operador que a linha de produção do simulador se encontra com operação suspensa;
- Indicador luminoso que quando está ligado indica que a linha de produção se encontra num estado de emergência.

4.4.2.1 Representação Gráfica

Os indicadores luminosos têm forma circular e podem apresentar diversas cores. Assim como para os botões, as cores disponíveis para os indicadores estão limitadas às cores que foram formatadas no código do simulador. Cada cor apresenta duas tonalidades diferentes, uma para quando o indicador luminoso está ligado (tom mais vivo) e outra para quando está desligado (tom menos vivo). Na versão atual do simulador estão disponíveis as seguintes cores: vermelho, amarelo, azul e verde.

A figura que se segue apresenta um painel de quatro indicadores luminosos com as quatro cores disponíveis e nos dois estados lógicos (ligado e desligado).

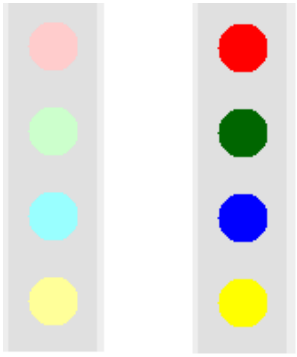

Figura 4.14: Painel com indicadores luminosos ligados (lado esquerdo) e desligados (lado direito)

4.4.2.2 Controlo

Cada indicador luminoso está associado a uma saída discreta Modbus (coil). Deste modo, o programa de controlo do simulador poderá controlar o seu estado através desta saída discreta. Se a

saída estiver ativa o indicador luminoso permanece ligado e se estiver inativa o indicador luminoso permanece desligado.

4.4.2.3 Ficheiro de Configuração

Os indicadores luminosos são definidos no ficheiro de configuração através de uma só entrada, que configura a sua cor:

- *led.ID.color = COLOR*
  ID: número inteiro que identifica de forma única o indicador luminoso;
  COLOR: cor do indicador luminoso em inglês (e.g. yellow, green, blue).

4.4.3 Arquitetura Interna

De modo a implementar o painel de botões e indicadores luminosos foi desenvolvido um novo pacote Java, *"button"*, que é constituído por quatro classes: *"Button"*, *"ButtonPanel"*, *"Led"* e *"LedPanel"*.

A figura 4.15 apresenta um diagrama UML deste pacote, com os principais campos e métodos das classes constituintes.

A classe *"Button"* permite definir os botões do simulador. É criado um objeto desta classe para cada botão indicado no ficheiro de configuração. Os principais campos que a constituem são:

- *id*: número identificador do botão;
- *alias*: nomenclatura para o botão;
- *colorON*: cor do botão quando está ativo;
- *colorOFF*: cor do botão quando está inativo;
- *shape*: forma do botão; o tipo de campo *SHAPE* é um conjunto de constantes com as diferentes formas possíveis (quadrada ou circular), que foi definido através da classe *enum*; esta classe pertence a uma biblioteca genérica Java e permite definir um tipo de campo que pode tomar valores de um conjunto de constantes definidas;
- *presstype*: tipo de botão; o tipo de campo *PRESSTYPE* é um conjunto de constantes com os diferentes tipos de botão (*pulse* ou *latch*);
- *digitalOutAddr*: endereço da saída discreta Modbus (coil) associada ao botão;
- *state*: estado lógico do botão; quando está ligado toma o valor TRUE e quando está desligado toma o valor FALSE;
- *malfunctionID*: quando o botão está associado à avaria de um equipamento, este campo representa o ID desse equipamento; se não estiver associado a uma avaria, toma o valor 0;

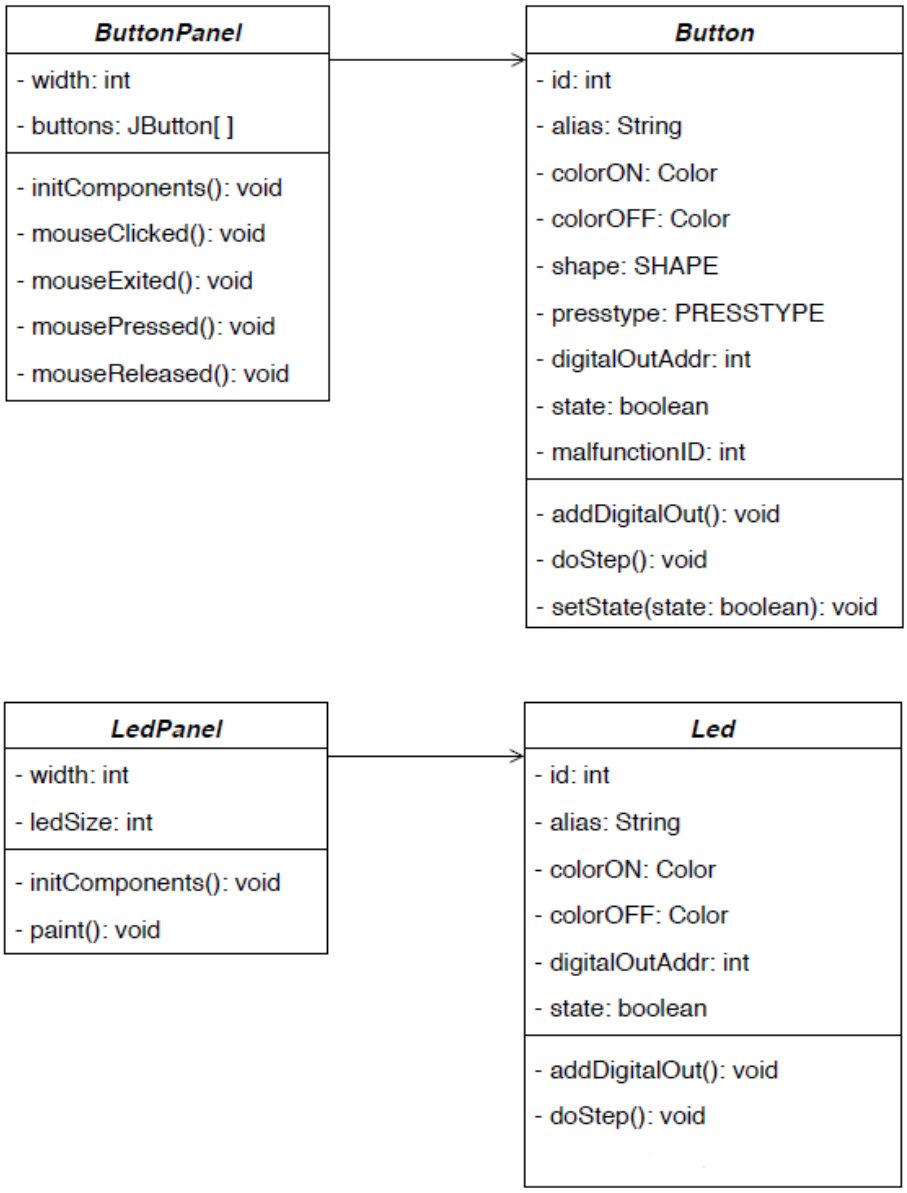

Figura 4.15: Diagrama UML do pacote *"button"*

Os principais métodos da classe *"Button"* são:

- *addDigitalOut()*: permite adicionar uma saída discreta (coil) ao servidor Modbus; é chamado quando o objeto de um botão é criado, para inicializar a saída discreta associada ao seu estado lógico;
- *doStep()*: atualiza o estado do botão com o valor lógico da saída discreta Modbus associada e atualiza o estado de avaria do equipamento associado; é utilizado para ativar o botão de forma automática, através da sua saída Modbus em vez de o ativar manualmente, através do rato; é chamado a cada ciclo de execução do simulador;
- *setState(state: boolean)*: permite alterar o campo *state* que indica o estado lógico do

botão e, se o botão estiver associado a uma avaria, atualizar o estado de avaria do equipamento respetivo;

A classe *"ButtonPanel"* é utilizada para desenhar o painel na interface do simulador e integrar os botões definidos na classe "Button". Apenas é definido um objeto para esta classe. Os principais campos que a constituem são:

- *width*: largura do painel de botões; a altura do painel coincide com a altura da interface do simulador;
- *buttons*: *array* com objetos da classe *JButton*, que pertence a uma biblioteca genérica Java e permite definir botões, para poder integrá-los na interface gráfica;

Os principais métodos da classe *"ButtonPanel"* são:

- *initComponents()*: integra o painel de botões na interface do simulador (lado esquerdo);
- *mouseClicked(), mouseExited(), mousePressed() e mouseReleased()*: estes métodos são chamados cada vez que há interações com os botões (e.g. botão pressionado, botão libertado) e atualizam o seu estado lógico em função dessas interações;

A classe *"Led"* permite definir os indicadores luminosos do simulador. É criado um objeto desta classe para cada indicador luminoso definido no ficheiro de configuração. Os seus principais campos são:

- *id*: número identificador do indicador luminoso;
- *alias*: nomenclatura para o indicador luminoso;
- *colorON*: cor do indicador quando está ativo;
- *colorOFF*: cor do indicador quando está inativo;
- *digitalOutAddr*: endereço da saída discreta Modbus (coil) associada ao indicador luminoso;
- *state*: estado lógico do indicador luminoso; quando está ligado toma o valor TRUE e quando está desligado toma o valor FALSE;

Os principais métodos desta classe são:

- *addDigitalOut()*: permite adicionar uma saída discreta (coil) ao servidor Modbus; é chamado quando o objeto de um indicador luminoso é criado, para inicializar a saída discreta associada ao seu estado lógico;
- *doStep()*: atualiza o estado do indicador luminoso com o valor lógico da saída discreta Modbus associada; é chamado a cada ciclo de execução do simulador.

A classe *"LedPanel"* é utilizada para desenhar um painel na interface do simulador e integrar os indicadores luminosos definidos na classe "Led". Apenas é definido um objeto para esta classe. Os principais campos que a constituem são:

- *width*: largura do painel de indicadores luminosos; a altura do painel coincide com a altura da interface do simulador;
- *ledSize*: diâmetro dos indicadores luminosos, que apresentam forma circular;

Os seus principais métodos são:

- *initComponents()*: desenha um painel na interface do simulador, junto ao painel de botões (lado direito);
- *paint()*: desenha os indicadores luminosos sobre o painel, com as cores correspondentes ao seu estado atual (cores vivas para indicadores ligados e cores menos vivas para indicadores desligados); é chamado a cada ciclo de execução do simulador.

Os objetos das classes descritas anteriormente são instanciados na classe *"Factory"*, que é a classe principal do simulador (descrita na secção 2.2.15.6).

4.4.4 Validação e Testes

Foram realizados vários testes para verificar o correto funcionamento dos botões e indicadores luminosos.

Os principais testes realizados para os botões foram os seguintes:

- Verificar se o botão mudava o seu estado lógico através da interface com o rato, alterando a cor (cor viva para botões ativos e cor menos viva para botões inativos);
- Verificar se o estado lógico da saída discreta Modbus correspondia ao estado lógico do botão, ligando-o e desligando-o múltiplas vezes através da interface com o rato;
- Verificar se o estado lógico do botão correspondia sempre ao estado lógico da saída discreta Modbus associada; para isso, alterou-se o valor da saída discreta e verificou-se se o botão se ligava e desligava devidamente na interface; este teste permitiu confirmar se o botão era controlado através da saída discreta, e não apenas pela interface do simulador.

Os principais testes realizados para os indicadores luminosos foram os seguintes:

- Verificar se os indicadores luminosos mudavam de cor em função do seu estado lógico (dado pelo campo *state* na classe "Led");
- Verificar se o estado lógico do indicador luminoso correspondia sempre ao estado lógico da saída discreta Modbus associada; para isso, alterou-se o valor da saída discreta e verificou-se se o indicador luminoso se ligava e desligava devidamente na interface.

4.5 Novas Funcionalidades do Servidor Modbus

Foi desenvolvido um módulo com diversas funcionalidades que permitem interagir com o simulador de forma automática (sem se utilizar o rato). Permitem agir sobre a linha de produção (e.g. gerar avarias, inserir e remover peças) e obter informação sobre o seu estado (e.g. estado de equipamentos, informação sobre peças).

Estas funcionalidades foram implementadas com a finalidade de automatizar o processo de avaliação de programas de controlo da linha de produção. Desta forma, poderão ser construídos algoritmos que permitam substituir as ações do utilizador e, ao mesmo tempo, verificar o cumprimento de um conjunto de objetivos ou requisitos predefinidos, de modo a validar o correto funcionamento de programas de controlo.

Foram adicionados novos elementos de memória ao servidor Modbus existente no simulador (e.g. coils, registos *holding*) para cada função implementada. Desta forma, qualquer programa externo ou equipamento que disponha de uma interface Modbus pode utilizar as novas funcionalidades do servidor.

Para testar as funções implementadas no simulador, foi ainda desenvolvido um programa externo com uma interface Modbus (cliente Modbus).

Neste capítulo faz-se uma descrição individual de cada funcionalidade implementada. É também feita uma descrição geral da arquitetura interna deste novo módulo e dos testes efetuados para validar o seu correto funcionamento.

4.5.1 Inserir Peça

Uma das funcionalidades implementadas no simulador permite inserir peças em qualquer equipamento com tapete (e.g. tapetes lineares, *pusher*, máquina ferramenta).

Os elementos de memória que foram adicionados ao servidor Modbus para implementar esta funcionalidade são os seguintes:

- *"Block barcode"*: conjunto de registos *holding* que são preenchidos pelo cliente Modbus com o código de barras da peça que se pretende inserir. O número de registos Modbus alocados para o código de barras depende do seu número de caracteres, definido no ficheiro de configuração. Cada registo tem 2 bytes e permite armazenar o código ASCII de dois caracteres (cada caractere ASCII necessita de 1 byte);
- *"Equipment Alias"*: conjunto de registos *holding* que são preenchidos pelo cliente Modbus com a nomenclatura ou ID do equipamento onde se pretende inserir a peça. O número de registos alocados depende do máximo número de caracteres para nomenclaturas de equipamentos, definido no ficheiro de configuração;
- *"Insert Block"*: saída discreta Modbus (*coil*) que é ativa pelo cliente para informar o simulador (servidor) de que o pedido de inserção de peça está pronto para ser processado.

De seguida, apresenta-se um exemplo da sequência de interações entre um cliente Modbus e o simulador (servidor Modbus), que permite demonstrar a inserção de uma peça num equipamento:

1. O cliente preenche o conjunto de registos *holding "Block barcode"*, que permitem indicar o código de barras da peça;

2. O cliente preenche o conjunto de registos *holding "Equipment Alias"* que permitem indicar o ID do equipamento onde se pretende inserir a peça;

3. O cliente ativa a saída discreta *"Insert Block"*, informando que o pedido de inserção de peça está pronto;

4. O simulador (servidor), ao verificar que esta saída discreta foi ativa, insere a peça com o código de barras indicado, sobre o equipamento indicado (lê os registos *"Equipment Alias"*).

4.5.2 Remover Peça

Foi também implementada uma funcionalidade que permite remover uma peça de qualquer equipamento do simulador (a peça desaparece na interface do simulador).

Os elementos de memória que foram adicionados ao servidor Modbus para implementar esta funcionalidade são os seguintes:

- *"Equipment Alias"*: é o mesmo conjunto de registos utilizados na funcionalidade que permite inserir peças. São preenchidos pelo cliente Modbus com a nomenclatura ou ID do equipamento de onde se pretende remover uma peça.
- *"Remove Block"*: saída discreta Modbus (*coil*) que é ativa pelo cliente para informar o simulador (servidor) de que o pedido de remoção de peça está pronto para ser processado.

De seguida, apresenta-se um exemplo da sequência de interações entre um cliente Modbus e o simulador (servidor Modbus), que permite demonstrar a remoção de uma peça:

1. O cliente preenche o conjunto de registos *"Equipment Alias"*, que permitem indicar o ID do equipamento de onde se pretende remover a peça;

2. O cliente ativa a saída discreta *"Remove Block"*, informando que o pedido de remoção de peça está pronto;

3. O simulador (servidor), ao verificar que esta saída discreta foi ativa, remove a peça do equipamento indicado (lê os registos *"Equipment Alias"*).

4.5.3 Gerar Avaria em Equipamento

Foi implementada uma funcionalidade que permite gerar avarias em qualquer equipamento.

Os elementos de memória que foram adicionados ao servidor Modbus para implementar esta funcionalidade são os seguintes:

- *"Equipment Alias"*: é o mesmo conjunto de registos utilizados nas funcionalidades descritas anteriormente. São preenchidos pelo cliente Modbus com a nomenclatura ou ID do equipamento que se pretende avariar;
- *"Malfunction"*: saída discreta Modbus (*coil*) que é ativa pelo cliente para informar o simulador (servidor) de que o pedido de avaria de equipamento está pronto para ser processado.

De seguida, apresenta-se um exemplo da sequência de interações entre um cliente Modbus e o simulador (servidor Modbus), que permite demonstrar a avaria de um equipamento:

1. O cliente preenche o conjunto de registos *"Equipment Alias"*, que permitem indicar o ID do equipamento que se pretende avariar;

2. O cliente ativa a saída discreta *"Malfunction"*, informando o simulador que o pedido de avaria está pronto;

3. O simulador (servidor), ao verificar que esta saída discreta foi ativa, gera uma avaria no equipamento indicado (lê os registos *"Equipment Alias"*).

4.5.4 Reparar equipamentos

Foi implementada uma funcionalidade que permite reparar equipamentos avariados. Há duas opções de utilização para esta funcionalidade: uma permite reparar um equipamento específico, outra permite reparar todos os equipamentos que se encontrem com avaria.

Os elementos de memória que foram adicionados ao servidor Modbus para implementar esta funcionalidade são os seguintes:

- *"Equipment Alias"*: é o mesmo conjunto de registos utilizados nas funcionalidades descritas anteriormente. São preenchidos pelo cliente Modbus com a nomenclatura ou ID do equipamento que se pretende reparar. São utilizados para a opção de reparação de um equipamento específico;
- *"Fix malfunction"*: saída discreta Modbus (*coil*) que é ativa pelo cliente para informar o simulador (servidor) de que o pedido de reparação de um equipamento específico está pronto para ser processado.
- *"Fix all malfunctions"*: saída discreta Modbus (*coil*) que é ativa pelo cliente para informar o simulador (servidor) de que o pedido de reparação de todos os equipamentos está pronto para ser processado.

De seguida, apresenta-se um exemplo da sequência de interações entre um cliente Modbus e o simulador (servidor Modbus), que permite demonstrar a reparação de um equipamento específico:

1. O cliente preenche o conjunto de registos *"Equipment Alias"*, que permitem indicar o ID do equipamento que se pretende reparar;
2. O cliente ativa a saída discreta *"Fix malfunction"*, informando que o pedido de reparação de um equipamento específico está pronto;
3. O simulador (servidor), ao verificar que esta saída discreta foi ativa, repara o equipamento indicado (lê os registos *"Equipment Alias"*).

O exemplo que se segue permite demonstrar a reparação de todos os equipamentos com avaria:

1. O cliente ativa a saída discreta *"Fix all malfunctions"*, informando que pretende reparar todos os equipamentos com avaria;
2. O simulador (servidor), ao verificar que esta saída discreta foi ativa, repara todos os equipamentos com avaria.

4.5.5 Obter Informação de um Equipamento

Foi implementada uma funcionalidade que permite a um cliente Modbus obter informação sobre o estado de qualquer equipamento do simulador (e.g. estado de avaria, presença de peça, estado de movimento).

Para esta funcionalidade é adicionado um registo *holding* ao servidor Modbus para cada equipamento do simulador (registos *"Equipment State"*). Os registos são atualizados a cada ciclo de execução do simulador com informação sobre o estado dos equipamentos respetivos.

Desta forma, um cliente Modbus poderá ler qualquer um destes registos, obtendo informação sobre qualquer equipamento do simulador.

A secção que se segue descreve o formato destes registos, indicando o tipo de informação de cada bit ou conjunto de bits que o constituem.

4.5.5.1 Registo *"Equipment State"*

O registo *"Equipment State"* segue o mesmo formato para todos os tipos de equipamento e, deste modo, alguns bits apenas são utilizados para determinados equipamentos (e.g. bit que informa sobre o estado de movimento de um *pusher*).

O primeiro byte do registo tem a estrutura que se encontra na figura 4.16.

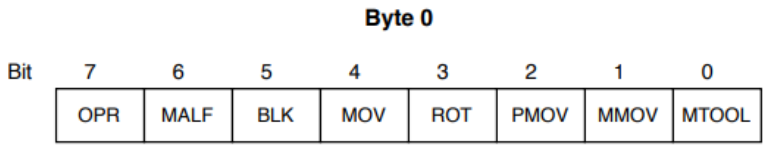

Figura 4.16: Estrutura do registo com o estado de um equipamento (byte 0)

Descrição de bits do primeiro byte do registo:

- Bit 7 - OPR: indica se o equipamento está operacional (1 - operacional, 0 - não operacional); um equipamento encontra-se no estado operacional se não estiver com avaria nem erro devido a ações ilegais (e.g. forçar movimento para além dos limites);
- Bit 6 - MALF: estado de avaria do equipamento (1 - com avaria, 0 - sem avaria);
- Bit 5 - BLK: indica se há alguma peça presente (1 - com peça, 0 - sem peça); só é utilizado para equipamentos com tapete; uma peça considera-se presente se pelo menos metade da sua área se encontrar sobre o tapete;
- Bit 4 - MOV: estado de movimento de equipamentos com tapete (1 - tapete em movimento, 0 - tapete parado);
- Bit 3 - ROT: estado do movimento de rotação de tapetes rotativos (1 - tapete a rodar, 0 - tapete sem rotação);
- Bit 2 - PMOV: estado do movimento de translação linear de tapetes deslizantes (1 - com movimento, 0 - sem movimento);
- Bit 1 - MMOV: estado de movimento de máquinas segundo o eixo X ou Y (1 - com movimento, 0 - sem movimento);
- Bit 0 - MTOOL: indica se a máquina está a trocar de ferramenta (1 - a trocar de ferramenta, 0 - sem trocar de ferramenta);

O segundo byte do registo apresenta a estrutura da figura que se segue.

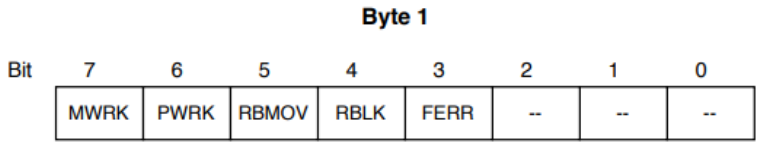

Figura 4.17: Estrutura do registo com o estado de um equipamento (byte 1)

Descrição de bits do segundo byte do registo:

- Bit 7 - MWRK: indica se a ferramenta de uma máquina está a atuar (1 - ferramenta a atuar, 0 - ferramenta inativa);
- Bit 6 - PWRK: indica se um equipamento do tipo *pusher* está a movimentar-se, seja no sentido de empurrar uma peça ou no sentido de regressar à posição de repouso, retraindo-se (1 - a movimentar-se, 0 - sem movimento);
- Bit 5 - RBMOV: estado de movimento de um robot 3D ao longo de X, Y ou Z (1 - a movimentar-se, 0 - sem movimento);
- Bit 4 - RBLK: indica se um robot 3D está a carregar uma peça (1 - com peça, 0 - sem peça);
- Bit 3 - FERR: estado de bloqueio do equipamento que ocorre quando são realizadas ações ilegais (e.g. robot 3D move-se para além dos limites da sua zona de ação); toma o valor 1 quando o equipamento está bloqueado e 0 quando não está.

Os 3 bits menos significativos do segundo byte não são utilizados e estão livres para ser preenchidos com informação adicional sobre equipamentos que possa ser útil para o processo de avaliação automática de programas de controlo.

4.5.6 Obter Informação sobre Peça

Foi implementada uma funcionalidade que permite a um cliente Modbus obter informação sobre uma peça do simulador.

Ao contrário dos registos que contêm informação sobre equipamentos (*"Equipment State"*), o registo utilizado para obter informações sobre uma peça é singular e é preenchido pelo simulador (servidor Modbus) apenas quando há um pedido por parte de um cliente Modbus.

Os elementos de memória que foram adicionados ao servidor Modbus para implementar esta funcionalidade são os seguintes:

- *"Block barcode"*: conjunto de registos *holding* que são preenchidos pelo cliente Modbus com o código de barras da peça sobre a qual se pretende obter informação. O número de registos Modbus alocados para o código de barras depende do seu número de caracteres, definido no ficheiro de configuração. Cada registo tem 2 bytes e permite armazenar o código ASCII de dois caracteres (cada caractere ASCII necessita de 1 byte);
- *"Block Info"*: registo *holding* que é preenchido pelo simulador (servidor) com informação sobre a peça pretendida. Para além dos bits com informação sobre a peça, este registo

dispõe de 2 bits para interação entre o cliente e o simulador: um dos bits é ativo pelo cliente para informar o simulador que há um pedido de informação sobre uma peça; o outro bit é ativo pelo simulador para informar o cliente que a informação foi preenchida e pode ser lida.

De seguida, apresenta-se um exemplo da sequência de interações entre um cliente Modbus e o simulador (servidor Modbus), que permite demonstrar esta funcionalidade:

1. O cliente preenche o conjunto de registos *"Block barcode"*, que permitem indicar o código de barras da peça;

2. O cliente ativa o bit do registo *"Block Info"* que informa o simulador que o pedido está pronto para ser processado;

3. O simulador (servidor), ao verificar que o bit de pedido foi ativo, preenche o registo *"Block Info"* com informação sobre a peça com o código de barras indicado (lê os registos *"Block barcode"*);

4. O simulador (servidor), após ter preenchido a informação da peça, ativa o bit de resposta do registo *"Block Info"*, para informar o cliente que a informação foi preenchida;

5. O cliente, ao verificar que o bit de resposta foi ativo, lê a informação sobre a peça pretendida.

A secção que se segue descreve o formato do registo *"Block barcode"*, indicando o tipo de informação de cada bit ou conjunto de bits que o constituem.

4.5.6.1 Registo *"Block info"*

A figura que se segue apresenta a estrutura do registo *"Block info"*, que é constituído por 2 bits de interação cliente-servidor e os restantes bits com informação sobre uma peça.

Byte 0

Bit	7	6	5	4	3	2	1	0
	REQ	RESP	TYPE3	TYPE2	TYPE1	TYPE0	INF2	INF1

Byte 1

Bit	7	6	5	4	3	2	1	0
	INF0	EQID6	EQID5	EQID4	EQID3	EQID2	EQID1	EQID0

Figura 4.18: Estrutura do registo com informação sobre uma peça

Descrição de bits exclusivos do primeiro byte do registo:

- Bit 7 - REQ: informa o simulador (servidor) que foi feito um pedido de informação sobre uma peça por parte do cliente Modbus (1 - pedido pendente, 0 - sem pedido); o servidor coloca este bit a zero quando preenche o registo com a informação;
- Bit 6 - RESP: informa o cliente Modbus que fez o pedido de que o registo já se encontra preenchido com a informação sobre a peça (1 - resposta pronta, 0 - sem resposta);
- Bits 5:2 - TYPE[3:0]: usados para indicar o número inteiro identificador do tipo atual da peça;

Descrição de bits exclusivos do segundo byte do registo:

- Bits 6:0 - EQID[6:0]: utilizados para indicar o ID do equipamento onde se encontra a peça; se esta não estiver sobre qualquer equipamento estes bits tomam o valor 0.

O conjunto de 3 bits INF[2:0] pertence a ambos os bytes do registo (bits 1:0 do byte 0 e bit 7 do byte 1). Permite informar o cliente Modbus sobre o estado genérico da peça especificada e pode tomar os seguintes valores:

- 000: valor por defeito, não tem qualquer significado;
- 001: informa que a peça se encontra sobre algum equipamento;
- 010: informa que a peça está a ser carregada pelo robot 3D;
- 011: informa que a peça está a ser carregada pela transportadora do armazém automático;
- 100: informa que a peça se encontra posicionada no armazém automático;
- 101: informa que a peça foi removida da linha de produção;
- 111: indica que o código de barras indicado pelo conjunto de registos "Block Barcode" não foi encontrado.

4.5.7 Arquitetura Interna

De modo a implementar as novas funcionalidades do simulador, foi adicionado um novo pacote ao código Java do simulador, que permite inicializar os novos elementos Modbus e gerir os pedidos de clientes Modbus. Este pacote é constituído por três classes: *"Automated_server"*, *"BlockInfoBitMap"* e *"EquipStateBitMap"*.

A classe *"BlockInfoBitMap"* é um mapa de bits, sendo apenas constituída por um conjunto de campos inteiros constantes que representam as posições de cada um dos bits no registo com informação sobre peças (*"Block info"*), descrito anteriormente.

A classe *"EquipStateBitMap"* é semelhante à classe anterior mas corresponde a um mapa de bits dos registos que contêm informação sobre o estado dos equipamentos do simulador (*"Equipment state"*).

A classe *"Automated_server"* é a classe principal deste pacote. É responsável por inicializar os elementos Modbus descritos anteriormente e por atender pedidos do cliente Modbus. Para atender estes pedidos dispõe de um método *doStep()* que é chamado a cada ciclo de execução do simulador, como já acontecia com algumas classes da versão anterior do simulador. Este método verifica o estado lógico dos elementos Modbus discretos cuja ativação por parte do cliente corresponde a um pedido:

- Saída discreta *"Insert block"*;
- Saída discreta *"Remove block"*;
- Saída discreta *"Malfunction"*;
- Saída discreta *"Fix malfunction"*;
- Saída discreta *"Fix all malfunctions"*;
- Bit de pedido do registo *"Block Info"* (descrito na secção 4.5.6.1).

A ativação de cada um destes elementos Modbus leva à execução dos seguintes métodos (funções), seguindo a ordem apresentada anteriormente:

- *insertBlock(blockType, blockBarcode, equipAlias)*: insere uma peça de determinado tipo e código de barras sobre o equipamento especificado;
  blockType: ID do tipo de peça a inserir, dado pelo registo Modbus *"Block type"*;
  blockBarcode: código de barras da peça, dado pelo registo *"Block barcode"*;
  equipAlias: nomenclatura ou ID do equipamento onde se pretende colocar a peça, dado pelo registo *"Equipment alias"*;
- *removeBlock(equipAlias)*: remove uma peça do equipamento especificado;
  equipAlias: nomenclatura ou ID do equipamento do qual se pretende remover peças, dado pelo registo *"Equipment alias"*;
- *generateMalfunction(equipAlias)*: gera uma avaria no equipamento especificado;
  equipAlias: nomenclatura ou ID do equipamento que se pretende avariar, dado pelo registo *"Equipment alias"*;
- *fixMalfunction(equipAlias)*: repara o equipamento especificado, que deixa de ter avaria;
  equipAlias: nomenclatura ou ID do equipamento que se pretende reparar, dado pelo registo *"Equipment alias"*;
- *fixAllMalfunction()*: repara todos os equipamentos da linha de produção do simulador;
- *setBlockInfo(blockBarcode)*: preenche o registo *"Block info"* com informação sobre a peça com o código de barras especificado e ativa o bit de resposta desse registo;
  blockBarcode: código de barras da peça, dado pelo registo *"Block barcode"*.

A funcionalidade que permite a um cliente Modbus obter informação sobre o estado de equipamentos, através dos registos "Equipment State", não é implementada em nenhuma das classes indicadas. Apesar destes registos serem inicializados na classe "Automated_server", o processo de atualizar a sua informação é feito no método *doStep* dos objetos correspondentes a cada um dos equipamentos do simulador.

O registo "Block info" é singular e é preenchido com informação apenas quando há um pedido por parte do cliente para uma peça específica. Por outro lado, existem múltiplos registos "Equipment State", havendo um para cada equipamento. Estes registos não são dependentes de um pedido por parte do cliente e são atualizados a cada ciclo de execução do simulador. Deste modo, o cliente poderá aceder diretamente ao registo Modbus de qualquer equipamento sem que

haja qualquer comunicação adicional.

A funcionalidade que permite obter informação sobre peças poderia ter sido implementada da mesma forma, havendo um registo por peça que é atualizado a cada ciclo de execução. Contudo, podem haver centenas de peças, que vão sendo adicionadas à linha de produção. Desta forma, o cliente teria de ser informado dinamicamente à cerca das peças adicionadas para guardar numa lista os endereços dos registos Modbus e os códigos de barras das peças respetivas.

Por outro lado, com os equipamentos tudo é mais simples porque estes são definidos no ficheiro de configuração e a sua quantidade permanece inalterada. Quando o cliente é inicializado, pode ler o ficheiro de configuração e guardar em memória uma lista que associa a nomenclatura e o ID dos equipamentos aos endereços Modbus que contêm a sua informação. Essa lista permanece inalterada.

4.5.8 Ficheiro de Configuração

Foi apenas adicionada uma nova entrada ao ficheiro de configuração:

- *facility.alias.maxnumchars = MAXCHARS*
  MAXCHARS: máximo número de caracteres para a nomenclatura de equipamentos.

Esta nova entrada no ficheiro é necessária para informar o simulador de quantos registos Modbus são necessários para a nomenclatura de equipamentos (conjunto de registos "Equipment Alias" descritos anteriormente).

4.5.9 Validação e Testes

Numa primeira fase, as funções implementadas no simulador foram testadas internamente para se verificar se cumpriam os requisitos propostos.

Posteriormente, foi desenvolvido um programa externo, utilizando a linguagem de programação Java, com uma interface Modbus que permite comunicar com o simulador (servidor Modbus). O programa permite inicializar um cliente (escravo) Modbus e conectar-se ao servidor do simulador. Dispõe de um conjunto de funções que foram utilizadas para testar as funcionalidades implementadas no simulador, através do controlo dos elementos Modbus respetivos. Este programa e as funções implementadas podem ser utilizados como base para o desenvolvimento de algoritmos de avaliação automática de programas de controlo da linha de produção do simulador.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Satisfação dos Objetivos

Os objetivos e os requisitos propostos para esta dissertação foram cumpridos na totalidade.

Foi desenvolvido um novo módulo que implementa o armazém automático do simulador. Este módulo inclui o próprio armazém, uma unidade de transporte e dois tapetes de acesso e permite simular o funcionamento do armazém automático da linha de produção flexível. Foi desenvolvido de forma flexível, sendo possível configurar algumas das suas características, como o número de linhas e colunas de armazenamento. Deste modo, é possível definir um armazém com propriedades diferentes do armazém da linha de produção flexível.

Implementou-se também o conceito de código de barras na simulação das peças, o conceito de avaria e reparação de equipamentos e um painel com botões e indicadores luminosos que fornecem uma nova interface de interação com o utilizador do simulador.

Por fim, foram ainda implementadas novas funcionalidades que permitem agir sobre a linha de produção do simulador de forma automática e obter informação sobre os diversos equipamentos e peças do simulador. Foram adicionados novos elementos ao servidor Modbus que permitem que qualquer programa externo ou equipamento que disponha de uma interface Modbus (cliente Modbus) utilize as novas funcionalidades. Desta forma, será possível desenvolver algoritmos que, utilizando estas funções, automatizem o processo de avaliação de programas de controlo da linha de produção.

5.2 Trabalho Futuro

Como trabalho futuro, poderão ser desenvolvidos algoritmos que, dispondo de uma interface Modbus, utilizem as novas funcionalidades implementadas no servidor Modbus do simulador de forma a automatizar o processo de avaliação de programas de controlo da linha de produção. Estes algoritmos irão permitir substituir as ações do utilizar e, ao mesmo tempo, verificar o cumprimento de um conjunto de objetivos predefinidos, de modo a validar o correto funcionamento de programas de controlo do simulador.

Referências

[1] Paulo Portugal. Linha de produção flexível v1.6, 2011. Automação, MIEEC, FEUP.

[2] National Instruments. O protocolo modbus em detalhe, Setembro 2019. Disponível em https://www.ni.com/pt-pt/innovations/white-papers/14/the-modbus-protocol-in-depth.html.