isec
Engenharia

MESTRADO EM ENGENHARIA ELETROTÉCNICA

Criação de projetos chave na mão em robótica industrial

DEFINITIVO

Autor

António Francisco Martins Cardão

Orientador

Doutor Nuno Miguel Fonseca Ferreira

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Coimbra, novembro de 2021

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA

Criação de projetos chave na mão em robótica industrial

Relatório de Estágio de Natureza Profissional para a obtenção do grau de Mestre em Engenharia Eletrotécnica

Especialização em Automação e Comunicações em Sistemas Industriais

Autor

António Francisco Martins Cardão

Orientador

Doutor Nuno Miguel Fonseca Ferreira

Supervisor na empresa

Real Robotic Systems, Lda

Eng. Luís Miguel Nolasco Andrade

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Coimbra, novembro de 2021

Man is still the most extraordinary computer of all.

John F. Kennedy

21 de maio de 1963

AGRADECIMENTOS

Agradeço aos meus familiares, pelo apoio recebido em todas as fases da minha vida, que se revelou crucial para o meu desenvolvimento pessoal e profissional. Agradeço-lhes pela compreensão e habitual motivação constante ao longo de todo o curso.

Ao orientador, Doutor Nuno Ferreira por toda a disponibilidade, preocupação, motivação, incentivo e ajuda no decorrer de todo o estágio curricular e durante o processo de escrita de este relatório de estágio.

Ao supervisor da empresa, e atualmente meu chefe, Engenheiro Luís Nolasco, por toda a dedicação, ensinamento, apoio, motivação, confiança e oportunidade para a realização do estágio curricular, agradeço-lhe a boa disposição, compreensão, paciência, conselhos e a total disponibilidade durante a realização deste trabalho.

À Inês, pela constante motivação, compreensão, ajuda, presença e carinho durante todo o período do estágio curricular e o período da elaboração do presente relatório de estágio.

Ao Hélder, que me acompanhou no decorrer do estágio curricular, por toda a presença, motivação, disponibilidade e companheirismo durante esta etapa.

Aos meus amigos, que continuamente me motivaram e se fizeram sentir presentes durante todo o percurso do mestrado.

E aos meus colegas de curso, por toda a experiência, crescimento e vivência que partilhei com eles ao longo do meu percurso académico.

RESUMO

A indústria está em constante mudança e as revoluções industriais são a prova disso. Desde o surgimento da industrialização que o objetivo da indústria é aumentar a produtividade. A automação industrial e a robótica são fulcrais para esse aumento de produtividade, sendo indispensáveis na produção hoje em dia.

Este relatório surge da realização de um estágio curricular para a obtenção do grau de mestre em Engenharia Electrotécnica, na área de especialização em Automação e Comunicações em Sistemas Industriais, desenvolvido na empresa *Real Robotic Systems, Lda*.

Um dos principais objetivos do estágio foi adquirir conhecimentos gerais das atividades da empresa, desenvolvendo tarefas que permitiram ganhar um maior conhecimento na área da automação industrial e robótica, entre outras tecnologias da indústria 4.0.

Este relatório de estágio está dividido em duas grandes partes, o enquadramento teórico-concetual e a apresentação das atividades desenvolvidas. Dentro das atividades desenvolvidas, são explicadas duas no decorrer do estágio, sendo que uma foi desenvolvida em teletrabalho e a outra no chão de fábrica. A primeira atividade é uma aplicação inovadora que permite realizar a conversão de linguagens de programação, de robôs industriais, entre diferentes modelos de controladores de sistemas robóticos. A segunda atividade foi desenvolvida no chão de fábrica e é um processo que interliga tecnologias de diferentes épocas, como um PLC com mais de 10 anos de idade e um robô colaborativo, e, ainda assim, se apresenta como uma solução atual e flexível como exigido atualmente pela indústria 4.0.

O estágio revelou-se uma mais-valia pelos conhecimentos adquiridos e o contacto com múltiplas tecnologias, levando à aprendizagem de novas formas de programar, o que permitiu quebrar novos paradigmas que me levaram a ganhar experiência profissional.

Palavras-Chave: Indústria 4.0, Automação Industrial, Robótica Industrial, Robótica Colaborativa, *C#*

ABSTRACT

Industry is constantly changing and the industrial revolutions are proof of this. Since the beginning of industrialisation, the aim of industry has been to increase productivity. The industrial automation and robotics are fundamental for this increase of productivity, being nowadays indispensable in the production.

This report arises from the conclusion of a curricular internship for obtaining a master's degree in Electrical Engineering, in the specialization area of Automation and Communications in Industrial Systems, developed in the company *Real Robotic Systems, Lda*.

One of the main objectives of the internship was to acquire general knowledge of the company's activities, developing tasks that allowed to obtain a greater knowledge in the area of industrial automation and robotics, among other Industry 4.0 technologies.

This internship report is divided in two main parts, the theoretical-conceptual framework and the presentation of the developed activities. Within the developed activities, two are explained during the internship, being that one was developed in telecommuting and the other on the shop floor. The first activity is an innovative application that allows the conversion of programming languages, of industrial robots, between different models of controllers of robotic systems. The second activity was developed on the shop floor and is a process that interconnects technologies from different eras, such as a PLC that is more than 10 years old and a collaborative robot, and yet, presents itself as a current and flexible solution, as currently required by Industry 4.0.

The internship proved to be an added value for the knowledge acquired and the contact with multiple technologies, leading to the learning of new ways of programming, which allowed me to break new paradigms that led me to gain professional experience.

Keywords: Industry 4.0, Industrial Automation, Industrial Robotics, Collaborative Robotics, C#

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE FLUXOGRAMAS

ÍNDICE DE TABELAS

ÍNDICE DE BLOCOS DE CÓDIGO

SIMBOLOGIA E ABREVIATURAS

3D	Três Dimensões
AGV	Automated Guided Vehicle
AMF	Atomic Monitoring Function
AP	Autómato Programável
AS/RS	Automated Storage/Retrieval System
ASCII	American Standard Code For Information Interchange
AT	Alta Tensão
BLE	Bluetooth Low Energy
BT	Baixa Tensão
C#	Linguagem de programação C# (pronuncia-se "cê chárp")
C++	Linguagem de programação C++ (pronuncia-se "cê mais mais")
CAD	Computer-aided design
CNC	Computer Numerical Control
Código QR	Quick Response Code ou Código de Resposta Rápida
COM	Comum
CPU	Unidade Central de Processamento
CSS	Cascading Style Sheets ou Folhas de Estilo em Cascata
DGS	Direção-Geral da Saúde
DLL	Dynamic-link library
E/S	Entradas e Saídas
EPI	Equipamentos de Proteção Individual
ERP	Enterprise Resource Planning
F#	Linguagem de programação F# (pronuncia-se "efe chárp")
FILO	Primeiro a entrar, último a sair
FTP	File Transfer Protocol
GPS	Global Positioning System
GUI	Interface Gráfica do Utilizador
HMI	Interface humano/máquina
HTML	Linguagem de Marcação de Hipertexto
IA	Inteligência Artificial
IDE	Integrated Development Environment (Ambiente Integrado de Desenvolvimento)

IEC	International Electrotechnical Commission
IFR	International Federation of Robotics
IIoT	Internet Industrial das Coisas
IL	Instruction List
IoT	Internet das Coisas
IPN	Instituto Pedro Nunes
ISO	International Organization for Standardization (Organização Internacional para Padronização)
IT	Tecnologia de Informação
JS	JavaScript, linguagem de programação
JS/TS	JavaScript/TypeScript
KLR	KUKA Robot Language
LD	Ladder Logic
MES	Manufacturing Execution Systems
mm	Milímetros
NC	Normalmente fechado(a)
NFC	Near Field Communication
Nm	Newton-metro
NO	Normalmente aberto(a)
OPC UA	OPC Unified Architecture
OT	Operational Technology
PLC	Programmable Logic Controller ou Controlador Lógico Programavel
RFID	Radio Frequency Identification ou Identificação por Frequência Rádio
RRS	Real Robotic Systems
RTD	Sensor de temperatura resistivo
RTOS	Real Time Operating System (Sistema operativo de tempo real)
SCADA	Sistema de supervisão e aquisição de dados
SCARA	Selective Compliance Assembly Robot Arm
SO	Sistema Operativo
ST	Strutured Text
TCP/IP	Transmission Control Protocol/Internet Protocolo
TPA	Terminal de Pagamento Automático
TS	TypeScript, superset da linguagem de programação JavaScript
TwinCAT	The Windows Control and Automation Technology

VAC	Volt em Corrente Alternada
VBA	Visual Basic for Applications
VDC	Volt em Corrente Contínua
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

CAPÍTULO 1: INTRODUÇÃO

O presente relatório de estágio, intitulado *Criação de projetos chave na mão em robótica industrial*, surge da realização de um estágio curricular na empresa *Real Robotic Systems, Lda*, realizado no âmbito do Mestrado em Engenharia Eletrotécnica na Área de Especialização em Automação e Comunicações em Sistemas Industriais, no Instituto Superior de Engenharia de Coimbra. Os dados apresentados são relativos ao período em que decorreu o estágio referido, por isso, qualquer modificação posterior não foi contemplada.

1.1 – Motivação

Atualmente, a automação industrial e a robótica deixaram de ser vistas como uma vantagem competitiva e um fator de segurança dos trabalhadores, sendo um importante fator decisivo entre uma produção que gera lucro ou prejuízo. A digitalização do mundo e, por consequência, da indústria trouxe novos desafios em todos os setores. Esta digitalização da sociedade juntamente com as novas tecnologias e a transição atual para a indústria 4.0, abrem novas possibilidades e trazem novas ferramentas e técnicas para automatizar a indústria, sendo os engenheiros de automação industrial responsáveis pela diferenciação de uma unidade industrial relativamente aos seus concorrentes.

A motivação para este estágio curricular surge da ambição de fornecer soluções robóticas e de automação industrial de alto nível, disponível a toda a indústria, aliada a uma enorme vontade de aprender e superar desafios.

Outro fator motivante deste estágio foi a possibilidade de realizar processos de automação com as mais diversas tecnologias. Durante o estágio foi-me permitido ter contacto com projetos de circuitos elétricos ao trabalhar no chão de fábrica até desenvolver e programar aplicações de automação com linguagens de programação de alto nível.

Por fim, o desafio e a oportunidade de trabalhar diariamente com a colaboração de engenheiros séniores em uma empresa nacional de robótica e automação industrial que cada dia ganha mais mercado é outra motivação que me fez optar por este estágio curricular.

1.2 – Objetivos do estágio curricular e do relatório de estágio

O objetivo principal do estágio curricular foi aprender automação industrial e robótica, tendo a aprendizagem passado por diversas tecnologias, *softwares*, *hardwares* e linguagens de programação diferentes. Este objetivo complementa-se pela necessidade de desenvolver novas soluções para os diferentes desafios, o que levou à aprendizagem em vastas áreas dentro da automação industrial e da robótica.

Desde 2020 que vivemos tempos atribulados, a pandemia causada pelo vírus *SARS-CoV-2*, que atingiu Portugal e o mundo, afetou o decorrer do estágio curricular levando a que grande parte do estágio fosse realizado em regime de teletrabalho. Por este motivo, o relatório de estágio aborda duas atividades, sendo uma delas desenvolvida durante o período de teletrabalho, não sendo possível realizar trabalho presencial nem no chão de fábrica, e uma segunda atividade já realizada no regime normal.

A primeira atividade referida foi desenvolvida em teletrabalho, trata-se de um projeto inovador com o objetivo de converter o código-fonte já desenvolvido em um robô *KUKA KR C1* de forma que este pudesse ser executado em um robô *KUKA KR C4*, um modelo mais recente. Este trabalho é particularmente minucioso, tendo em conta a evolução natural dos próprios robôs da *KUKA*, a linguagem de programação utilizada já não partilha da mesma sintaxe. A importância deste projeto centra-se na eficiência, sendo que refazer o código todo seria um trabalho que iria exigir muito tempo e seria muito suscetível a erro.

O segundo projeto refere-se ao período onde foi possível realizar o trabalho no chão de fábrica. Este projeto conta com a modificação de um posto de uma linha de montagem, sendo que é apresentada a solução em termos de automação industrial e com um robô colaborativo. Os robôs colaborativos são a mais recente tecnologia de robôs, sendo estes desenvolvidos para trabalhar em colaboração com o Homem em segurança. Esta tecnologia é uma mais-valia em qualquer empresa, uma vez que a interação entre o robô e o trabalhador é extremamente importante para se poder atingir a maior efetividade e rentabilidade em cada processo (Pentikäinen & Richard, 2016).

1.3 – A empresa Real Robotic Systems, Lda

A empresa escolhida para se realizar o estágio foi a *Real Robotic Systems, Lda* (também conhecida como RRS), sediada no Instituto Pedro Nunes (IPN) em Coimbra, onde inicialmente teve uma incubação virtual e de momento tem uma incubação física.

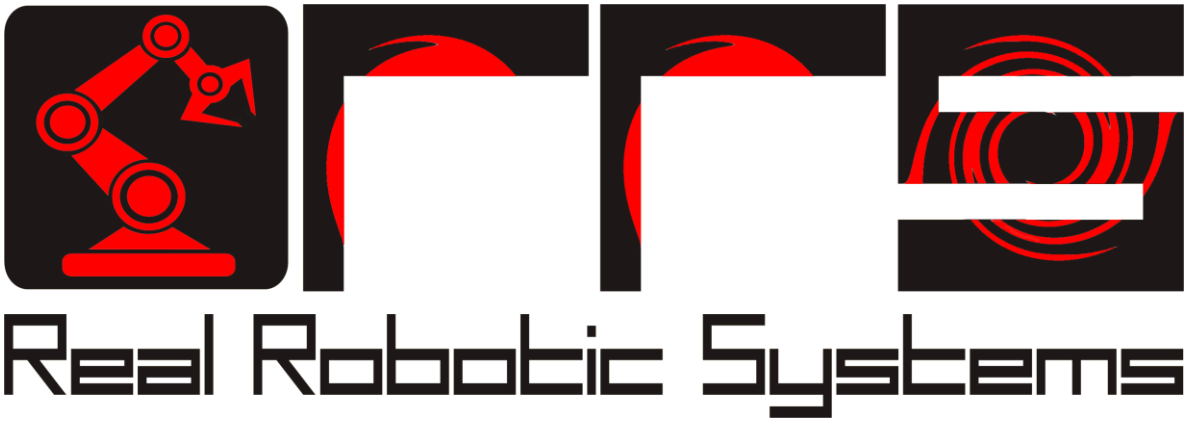


Figura 1 - Logotipo da empresa. Fonte: https://real-robotic-systems.pt

A *Real Robotic Systems* é uma empresa recente no mercado, criada em 2018, que realiza trabalhos abrangidos pelo CAE 71120 (atividades de engenharia e técnicas afins) que tem como missão prestar serviços de alta qualidade aos seus clientes na área de automação industrial e robótica industrial, mais especificamente.

A empresa, atualmente conta com três colaboradores e tem registado uma duplicação de ano para ano no volume de faturação. Os seus principais clientes são *Group Renault*, *Embraer*, *Eiffage Clemessy*, *RTE Delivering Cycling Solutions*, *Allied Motion*, *fluidotronica*, entre outras.

Como já referido anteriormente, a pandemia causada pelo vírus *SARS-CoV-2*, afetou o decorrer do estágio curricular. Por decisão interna da empresa, o estágio decorreu em teletrabalho sempre que as atividades foram compatíveis. Quando as mesmas não foram, ou deixaram de ser, compatíveis, nomeadamente serviços em que fomos subcontratados e houve a necessidade de nos deslocarmos à empresa cliente, esta deslocação era feita com o acompanhamento do responsável da empresa na orientação do estágio, respeitando as normas também impostas pela DGS e a utilizando os devidos EPI.

Mais informações acerca da empresa podem ser consultadas no website da *Real Robotic Systems*: https://real-robotic-systems.pt.

1.4 – Conceitos fundamentais do relatório de estágio

O termo "Indústria 4.0" foi utilizado para se referir aos processos de produção inovadores, parcial ou completamente automatizados através da tecnologia, e aos dispositivos que comunicam de forma autónoma entre si ao longo das atividades da cadeia de valor (European Commission, 2017). Assim, baseia-se basicamente na rede inteligente de máquinas, equipamentos elétricos e novos sistemas de tecnologias da informação (IT) que permitem a otimização dos processos e o aumento da produtividade das cadeias de criação de valor (Bogner, Voelklein, Schroedel, & Franke, 2016).

A transformação digital é o elemento-chave da revolução industrial em curso (Bogner, Voelklein, Schroedel, & Franke, 2016). Aqui, o conceito de digitalização não se refere a uma simples transferência de "análogo" para dados e documentos digitais. Representa antes a ligação em rede entre as interfaces criadas, os processos de negócio, a troca e a gestão de dados. Por conseguinte, a digitalização em rápido crescimento tem vindo a transformar extremamente a dinâmica da maioria das indústrias.

1.5 – Estrutura do relatório de estágio

O presente relatório de estágio irá assentar em 5 capítulos fundamentais, onde se fará uma exposição sobre as atividades desenvolvidas ao longo do estágio curricular.

Neste primeiro capítulo, é apresentada a introdução ao relatório de estágio, as motivações para a realização do mesmo, os objetivos do estágio curricular e do relatório de estágio, a empresa onde o mesmo se realizou e a estrutura deste.

No segundo capítulo é apresentado o enquadramento metodológico, onde é feito um enquadramento teórico-concetual dos temas que se abordam, sendo estes: a evolução da indústria, automação industrial, robótica industrial e colaborativa.

No penúltimo capítulo são apresentadas duas atividades realizadas no decorrer do estágio curricular. A primeira atividade foi realizada no âmbito de teletrabalho. Esta consiste no desenvolvimento de uma aplicação para automatizar um processo de conversão da linguagem de programação *KUKA Robot Language*, que será abordada daqui em diante como *KRL* (devendo esta ser lida como *'Carl'*), usada pelos robôs da *KUKA*. Será apresentada toda a discussão e motivos da utilização da linguagem de programação *C#* para o desenvolvimento da aplicação, assim como o modo de funcionamento e de validação dos códigos *KRL*. A segunda atividade apresentada foi desenvolvida no chão de fábrica, sendo que esta interliga a automação industrial e robótica. Serão abordados os problemas existentes, o porquê da necessidade de se intervir no posto, os cuidados necessários a se ter na intervenção, todos os elementos presentes e usados, os protocolos de comunicação e a descrição das soluções encontradas assim como a implementação das mesmas.

No último capítulo, são apresentadas as principais conclusões bem como as limitações do relatório de estágio. Além das conclusões, é indicada toda a bibliografia referenciada ao longo do documento e os anexos fundamentais.

CAPÍTULO 2: ENQUADRAMENTO TEÓRICO-CONCETUAL

2.1 – Enquadramento e conceitos

A automação encontra-se presente no nosso quotidiano, seja através de processos ou objetos automatizados, esta facilita a nossa vida nos dias atuais, fornecendo-nos mais conforto e qualidade.

A automação industrial deixou de ser exclusivamente uma vantagem e fator de segurança para os trabalhadores e "passou a ser um imperativo sem o qual nenhuma empresa será capaz de sobreviver no mercado" (Pinto, 2021, p. 3). A tendência, hoje, é 'atribuir inteligência' aos processos/máquinas que connosco trabalham, permitindo assim que os mesmos sejam mais autónomos. "Este salto para a indústria digital é conhecido por indústria 4.0 ou 4ª Revolução Industrial" (Pinto, 2021, p. 3).

No entanto, nem sempre foi assim, no passado, o Homem era a ferramenta principal para o trabalho. Existia menos qualidade de vida, os trabalhos eram mais pesados e com baixas remunerações (Jack, 2008, p. 20), sendo que o Homem sempre procurou automatizar. Automatizar significa, genericamente, dispensar o Homem ou outros seres vivos[1] da realização de tarefas, algo que estava na mente da humanidade desde a pré-história, como os relógios de água gregos e romanos e os martelos movidos água dos chineses, há mais de 2 mil anos atrás.

2.2 – Evolução da automação industrial

2.2.1 – Indústria 1.0

Com o surgimento do conceito de industrialização, nos finais do século XVIII, surgiu também a Indústria 1.0 (conhecida como a 1ª revolução industrial). Esta coincide com o surgimento das máquinas a vapor, sendo estas a principal forma de automação da época. Uma das primeiras máquinas alimentadas a vapor de água, foi um tear mecânico, em 1784, apresentado na Figura 2, sendo que a indústria têxtil era a principal indústria de esta época.

[1] Um burro a puxar uma nora não pode ser considerado um sistema automático (Pinto, 2021, p. 4).

Figura 2 - Primeiro tear mecânico desenvolvido em 1784. Fonte: (Nunes, 2015)

Nesta altura as unidades industriais existentes eram escassas e não concorrentes entre si, levando a que as instalações fossem destinadas a fabricar um dado produto durante muitos anos, ao fim do qual exigia uma total reconfiguração do espaço, com custos muito elevados para fabricar um novo produto (Estudar com Você, 2017). Por outro lado, a mão de obra era barata, abundante, sem capacidade de reivindicar por melhores salários, horários ou condições de trabalho, o que fazia com que não houvesse necessidade de automatizar.

Os primeiros sistemas a utilizar ar comprimido como fonte de potência surgiram no final da 1ª revolução industrial. “Nesta fase, só se pode falar em automação num sentido muito estrito, pois significa essencialmente que os músculos humanos podem ser substituídos por outros mecanismos geradores de potência” (Pinto, 2021, pp. 4-5).

2.2.2 – Indústria 2.0

A utilização da energia elétrica, pneumática e hidráulica tornou possível a produção em massa. A principal indústria a contribuir para a produção em massa, foi a indústria automóvel em 1913, ficando a esta associada a indústria 2.0. “Uma das características deste período foi o desenvolvimento de linhas de montagem, que automatizavam os fluxos de produção à medida que matérias-primas e peças eram transformadas em produtos acabados” (Pinto, 2021, p. 5).

A indústria automóvel passou a montar um veículo a cada 30 minutos, em vez das 12 horas anteriormente necessárias, com o surgimento do tapete rolante em 1870. Esta ‘simples’ mudança fez com que cada linha de montagem, produzisse 24 vezes mais por dia.

É durante esta época que surgem as primeiras denúncias das condições do trabalho, exigindo mais segurança, mais tempo de descanso e melhores salários. O aumento da concorrência, com o crescimento do número de unidades industriais, levantou a questão da produtividade, a automatização de alguns componentes do processo de fabrico contribuiu parcialmente para a resolução deste problema, sendo que o objetivo da automação industrial nesta época era essencialmente diminuir os custos, aumentar a produtividade e aumentar a segurança para os trabalhadores (Pinto, 2021, p. 6).

Para conseguir atingir os objetivos anteriormente mencionados foram desenvolvidos os primeiros automatismos, sistemas que permitem a realização automática de operações. Estes foram implementados recorrendo a uma lógica cablada, também conhecida como lógica baseada em relés, como ilustrado na Figura 3. Nesta lógica, "o funcionamento do automatismo é determinado pela forma de ligação dos condutores (cablagem) entre os diferentes constituintes do automatismo (relés, temporizadores, etc.)" (Francisco, 2015, pp. 1-2).

Figura 3 - Exemplo de Lógica Cablada e baseada em relés. Fonte: (Petruzella, 2011, p. 3).

Contudo, a lógica cablada tem alguns problemas relacionados com a modificação do sistema (Petruzella, 2011, p. 2), sendo que Francisco (2015, p. 2) afirma:

> Na lógica cablada, qualquer modificação no funcionamento do automatismo implica modificar fisicamente a cablagem e, normalmente, novos componentes. Também é obrigatório parar o processo de fabrico, uma vez que é necessário realizar trabalho de desmontagem/montagem. O custo final é apreciável.

Nesta altura apenas grandes empresas investiam em sistemas de automatismo, devido à dificuldade de se implementar um sistema que estivesse constantemente atualizado e que não fosse muito dispendioso. Na maioria dos casos a única forma de modificar estes sistemas após a sua instalação era voltar ao ponto de partida, por isto esta automação é designada por automação fixa. Esta ainda hoje se pode encontrar em sistemas mais antigos ou em processos que devem ser criticamente seguros (Pinto, 2021, p. 6).

2.2.3 – Indústria 3.0

Antes do que se considerou o início da terceira revolução industrial surgiram os computadores digitais. Estes foram integrados em algumas máquinas ferramentas, como máquinas de comando numérico (CNC – *Computer Numerical Control*). Esta tecnologia é mais versátil do que a lógica cablada baseada em relés e passou a ser conhecida por automação programável. No entanto, apesar de mais flexível, estas máquinas para serem programadas, tinham de ser desligadas do processo, obrigando a parar a produção, algo que não era desejável.

Após as duas grandes Guerras, o mundo entrou numa época próspera, conhecida como os anos dourados, época em que o termo automação começou a popularizar (Santos G. , 2020). Durante estes 'anos dourados' deu-se a terceira revolução industrial e esta deu origem à designada indústria 3.0. A indústria 3.0 foi estabelecida com a entrada dos microprocessadores na indústria e dos primeiros protocolos de comunicações indústrias (Pinto, 2021, p. 7).

O primeiro grande impacto desta indústria foi o surgimento do autómato programável (AP) ou controlador lógico programável (***P****rogrammable* ***L****ogic* ***C****ontroller* – PLC[2] (em Inglês)), que será abordado no próximo subcapítulo.

A tecnologia industrial neste ponto, dividiu-se em 5 níveis diferentes de gestão e decisão do processo fabril, sendo eles:

- **Nível 0, ou nível de campo:** é constituído pelos sensores e atuadores presentes no chão de fábrica;
- **Nível 1, ou de controlo:** é onde se encontra o PLC. Este recebe informações dos sensores nas suas entradas e emite sinais para os atuadores pelas suas saídas. Para além de ser capaz de enviar e receber dados do nível inferior a ele, também é capaz de comunicar com o nível seguinte, que será abordado no próximo parágrafo. Para isto, foram desenvolvidos os protocolos de comunicação que permitiram recolher os dados referentes aos níveis 0 e 1 e partilhá-los com outro PLC (comunicação dentro do mesmo nível) ou com computadores industriais (comunicação com o nível 2);
- **Nível 2, ou de supervisão:** é constituído essencialmente por computadores industriais integrados em interfaces humano/máquina (HMI[3]), que será uma tecnologia abordada em um subcapítulo futuro. Estes têm um elevado poder de cálculo e correm software de supervisão, o SCADA[4]. Estes conseguem

[2] Esta será a sigla usada daqui em diante para nos referirmos a Controladores Lógicos Programáveis.
[3] Daqui em diante iremos usar esta nomenclatura para nos referirmos a isto.
[4] Sistema de supervisão e aquisição de dados.

comunicar diretamente com os níveis inferiores e funcionam como *gateway* para comunicar com os níveis superiores de IT.

- **Nível 3, ou sistemas de execução da produção (MES[5]):** trata-se de *software* que faz a gestão e monitoriza o que se passa no chão da fábrica. Este, na indústria 3.0, recebia informações do nível imediatamente acima e abaixo. Atualmente, esta hierarquia já não é tão respeitada, sendo que este nível já comunica diretamente com os níveis inferiores de OT.
- **Nível 4, ou planeamento dos recursos da empresa (ERP[6]):** é o *software* que permite o planeamento e logística do negócio de uma ou mais unidades de produção industrial. "Durante a indústria 3.0 este *software* era usado ao nível dos conselhos de administração das empresas e permitia introduzir em toda a cadeia produtiva as decisões mais ou menos políticas de gestão" (Pinto, 2021, p. 9).

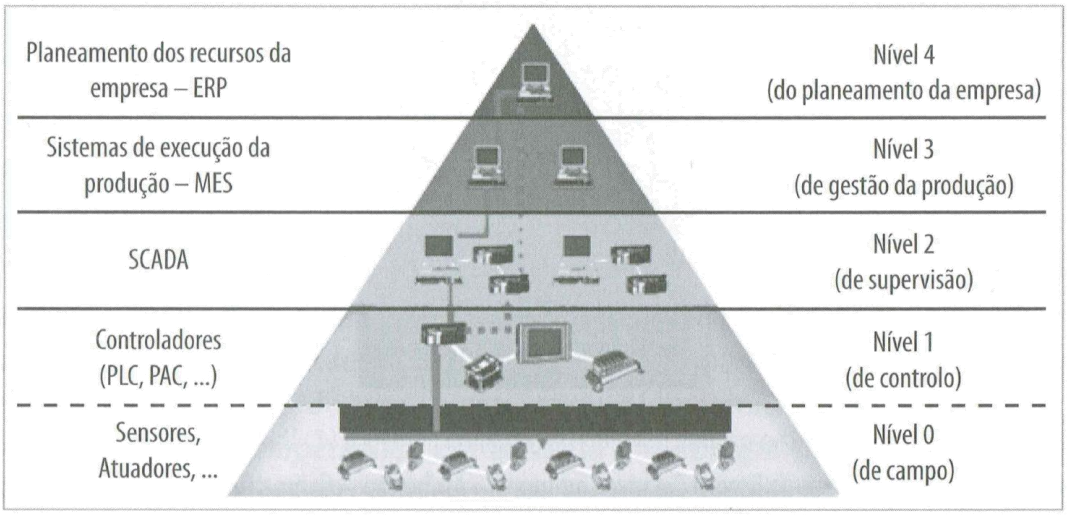


Figura 4 - Pirâmide da automação industrial na indústria 3.0. Fonte: (Pinto, 2021, p. 8)

O tempo de resposta de cada nível está relacionado com a sua hierarquia. Quanto mais elevado o nível, maior é a sua complexidade, e por isso, é necessário mais tempo para que uma decisão tomada em um nível superior chegue a um nível inferior. Quanto maior a diferença entre os níveis, maior será o tempo necessário para que as decisões se reflitam no chão de fábrica.

Por fim, é de referir que na indústria 3.0 toda a nova tecnologia permitiu que muitas alterações ao processo de fabrico fossem concretizadas em tempo real e sem a necessidade de haver uma paragem na linha. Este conceito, conhecido com a

[5] *Manufacturing Execution Systems*.
[6] *Enterprise Resource Planning*.

automação flexível, é possível devido aos próprios controladores e aos robôs capazes de se adaptarem às necessidades.

A distinção entre a automação programável e a automação flexível, é que esta não tem a necessidade de parar o processo de produção ao fim de ser reprogramado ou incluir um novo processo. No entanto, a produção customizada, como alterar as cores pretendidas que se deseja pintar uma peça em uma linha de pintura, apenas é possível na indústria 4.0.

2.2.4 – Indústria 4.0

A 4ª revolução industrial está datada como tendo sido iniciada na segunda década do século XXI, sendo que à data da publicação de este relatório de estágio, esta revolução teve início há aproximadamente 10 anos. A mesma deu origem ao conceito da indústria 4.0 e é amplamente discutida pelos agentes da sociedade.

Surgiu quando se começou a perceber os avanços informáticos, com a alta capacidade de cálculo e a minimização do tamanho dos equipamentos. Esta evolução informática permitiu reformular toda a cadeia produtiva, tendo sido introduzido em 2011 pelo Governo alemão o conceito da Indústria 4.0 (Pinto, 2021, pp. 9-10).

Esta reformulação da indústria tem como objetivo uma ampla automatização de todo o processo a todos os níveis da cadeia. Para isto, a integração da gestão e do chão de fábrica é feita através da troca de informação em tempo real, sendo que desta forma qualquer decisão tomada nos níveis superiores (ERP, MES, SCADA), tem um impacto imediato no processo. Isto apenas é possível pelo facto de os sistemas de gestão e controlo estarem a trabalhar em convergência em uma *cloud*. Alguns autores consideram que o sistema já não é uma pirâmide como representado na Figura 4 e passa a ser mais cilíndrica, como ilustrado na Figura 5.

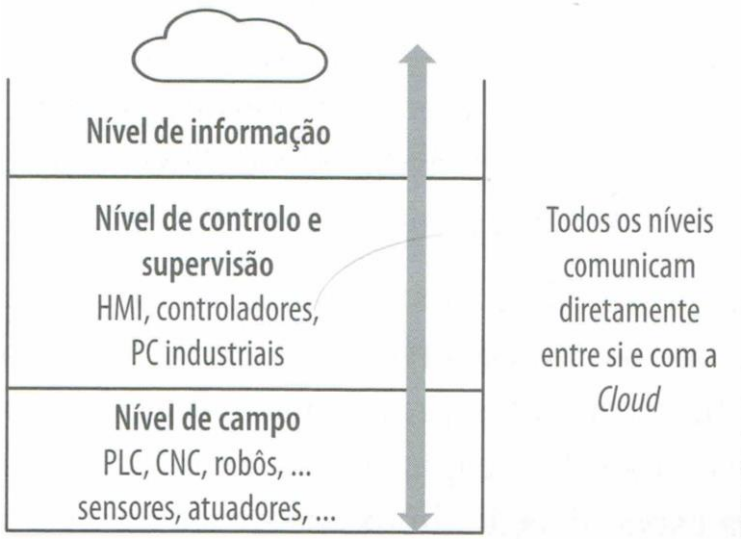


Figura 5 - Estrutura da automação industrial na indústria 4.0. Fonte: (Pinto, 2021, p. 10)

Desta forma, a indústria 4.0 resulta da conjugação de vastas áreas científicas e tecnológicas, com um novo objetivo que nunca antes existiu, a necessidade de diminuir a pegada ecológica da produção. São conhecidos como os principais pilares da indústria 4.0:

- **Redes de comunicação e Internet das Coisas:** redes de comunicação, com ou sem fios de protocolos abertos e normalizados para interligar todo o tipo de equipamento. Também constituído por a Internet das Coisas (IoT) e a Internet Industrial das Coisas (IIoT), sendo uma das bases da digitalização da sociedade;
- **Sistemas ciberfísicos:** sistemas complexos constituídos por um equipamento singular ou múltiplos equipamentos interligados entre si por uma rede que também envolve sensores e atuadores, sejam estes de natureza analógica ou digital, dotados de software que permite a execução de uma ampla gama de tarefas com elevada autonomia;
- ***Cloud*, *fog* e computação de fronteira:** *cloud* surge sobre a globalidade dos processos, em que todos os processos fornecem dados para a *cloud* permitindo assim a sua análise e produzir estatísticas. Estas estatísticas geram informações que permitem fazer manutenções preventivas. *Fog* trata-se de uma rede de computadores de fronteira (*edge computing*), que funcionam como nuvem local. O seu objetivo é tratar informação localmente, diminuindo a submissão e acesso à *cloud* e proporcionando um alívio significativo aos processadores desta. Os servidores *fog* estão no limiar do espaço físico onde decorre o processo automatizado e são melhores para processar informação em tempo real;
- **Inteligência artificial, aprendizagem automática e analítica:** inteligência artificial (IA) é um conceito antigo, com a ambição de ter computadores a pensar e tomar decisões como humanos. Para se atingir este objetivo, foi estudado o cérebro do Homem e chegamos ao sistema de redes neurais usadas na IA. Na automação industrial a IA é usada de duas formas distintas, sendo uma localmente com algoritmos de redes neurais, usados por exemplo, no controlo de variáveis ou padrões e formatos das peças recorrendo à visão digital, e usado a nível global onde se recorre à *cloud* e à *big data*. “Desta forma os algoritmos de IA encontram-se instalados nos mais variados suportes, desde os serviços da *cloud* até aos computadores de fronteira, passando por sistemas ciberfísicos e alguns equipamentos inteligentes, como sensores”. Desta forma, a IA com base nos dados recolhidos, tem como função a descrição, diagnóstico, previsão e prescrição;
- **Cibersegurança:** a preocupação com a cibersegurança é um tema atual, uma vez que antes as redes entre os equipamentos eram proprietárias e fechadas, e não existia grandes dados armazenados. Com a entrada da *big data*, da IoT, da IIoT, protocolos de redes abertas e a necessidade da sua

integração com as IT, a cibersegurança passou a ser uma prioridade. Sendo que a *Cisco*[7], identificou como potenciais ameaças nas comunicações indústrias os HMI infetados, acessos por pessoal não autorizado, ameaças a partir dos serviços de *cloud* e internet e ameaças provenientes de acesso remoto;

- **Fábricas digitais gémeas:** concebidos para substituir os protótipos físicos, mais caros e complexos do chão de fábrica, são desenvolvidas fábricas digitais gémeas. Estas permitem que sejam testadas diferentes abordagens ao problema existente, sem haver a necessidade de se mexer no sistema real do chão de fábrica. Possibilita desenvolver otimizações nos processos e interagir em tempo real, sendo possível detetar defeitos ou desvios no comportamento. Desta forma, fábricas digitais gémeas são também um método de manutenção preventiva;
- **Realidade virtual e aumentada:** passa por introduzir o humano, recorrendo a meios informáticos, em uma realidade criada artificialmente. Estes cenários são fabris e têm como objetivo desenvolver capacidades com uma máquina especifica, o chão de fábrica ou uma sala de reuniões, com o objetivo de permitir que as instalações possam ser 'visitadas e testadas'.

Deste modo, as vantagens e desafios da indústria 4.0 são a de tornar a produção mais flexível e customizada, aumentar a produtividade, aumentar a eficiência na utilização de recursos e de energia, diminuir desperdícios, ligar vários dispositivos de diferentes processos logísticos, otimizar as decisões do processo, identificar e prevenir problemas com a manutenção preventiva, facilitar novas oportunidades de negócio e melhorar as condições laborais.

2.3 – Elementos da automação industrial

Neste subcapítulo vamos abordar o *hardware* que é usado na automação industrial, situando-se quase todo no chão de fábrica. Ficarão de fora os PLC, os robôs e as interfaces humano/máquina uma vez que cada um destes terá o seu próprio subcapítulo. Os elementos que iremos abordar nesta secção estão divididos em:

- Máquinas ferramentas de comando numérico destinadas ao processamento do material;
- Equipamento de manipulação e montagem;
- Elementos de transporte;
- Armazéns automáticos;

[7] Empresa de redes e comunicações de referência a nível mundial.

- Sensores variados, elementos de sinalização e segurança.

Por uma questão de coerência com a própria história, iremos abordar estas tecnologias tentando respeitar a ordem do seu surgimento na indústria, sempre que se justifique.

Na primeira revolução industrial (indústria 1.0), existiram evoluções para facilitar a tarefa humana, no entanto, não existiram grandes avanços tecnológicos que ainda hoje se encontrem presentes na indústria.

A segunda revolução industrial, com o início da integração da eletricidade na indústria, trouxe novas tecnologias que ainda hoje são recorrentemente encontradas em um chão de fábrica. As mais comuns são os tapetes rolantes, circuitos pneumáticos, válvulas pneumáticas, circuitos elétricos, sensores e atuadores.

2.3.1 – Tapetes transportadores

Os tapetes rolantes permitiam interligar múltiplas unidades de produção, acelerando a produção e ajudando na produção em série. Dentro destes existem os seguintes tipos de tapetes rolantes:

- **Tapete transportador de correia:** sendo este resistente, com baixa necessidade de manutenção, suporte para transporte de peças de qualquer dimensão e com boa aplicação quando à a necessidade de mover produtos a granel, uma vez que o produto é sempre apoiado, por todo, devido à correia do transportador;
- **Tapete transportador de rolos:** a vantagem de estes tapetes é o facto de estes permitirem fazer um trajeto que não seja retilíneo sem que haja diminuição da velocidade de transporte. Permite também a acumulação de peças (*buffer*) e têm grande resistências a impactos e capacidade de transporte de peças com peso elevado;
- **Tapete telescópico de correia e rolos:** este tipo de tapete encontra-se presente em docas de carregamento e descarregamento, uma vez que é facilmente movível e reconfigurável em diferentes comprimentos. Este sistema é por norma usado para carregar/descarregar pacotes soltos;
- **Tapete transportador de corrente:** este tapete é usado quando não existe a necessidade da acumulação do produto e quando o que se procura transportar não pode ser corretamente transportado pelo tapete transportador de rolos. Estes são usados principalmente para mover paletes, *containers* industriais ou *racks* com dimensões e pesos elevados. O seu fluxo também é contínuo o que permite com que as peças se movam independentemente uma das outras consoante as necessidades. Este tapete é usado principalmente na indústria agroalimentar e automóvel;
- **Tapete transportador extensível:** são usados em espaços pequenos, uma vez que conseguem reduzir o seu tamanho até três vezes. Ótimo para transportar pacotes soltos e manobrar uma vez que possuem rodas com travões. (JHernando, 2020)

2.3.2 – Sistemas pneumáticos

Relativamente aos sistemas pneumáticos desenvolvidos e usados na indústria 2.0, podem ser vistos como “um conjunto de elementos físicos convenientemente associados que, utilizam um gás como meio de transferência de energia, permite a transmissão e controlo de forças e movimento” (De Negri, Kinceler, & Silveira, 1998, p. 49). Ainda hoje são regularmente usados, sendo que os principais motivos para a utilização do ar comprimido, são os factos de que estes têm uma resposta rápida, existe em abundância, é gratuito e não é necessário haver uma preocupação com o caminho de retorno uma vez que este pode ser injetado diretamente para o ar (ao contrário da hidráulica, por exemplo).

Para se controlar a direção e sentido em que o ar comprimido circula são utilizadas válvulas direcionais, sendo estas também usadas na hidráulica. A representação do funcionamento de esta válvulas está representada na Figura 6.

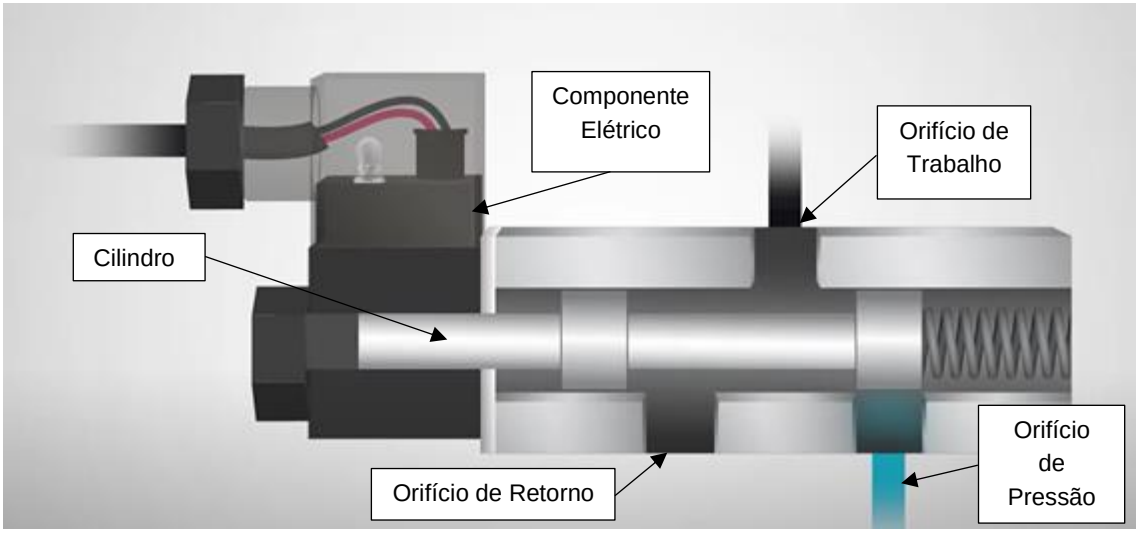


Figura 6 - Representação do interior de uma válvula direcional. Fonte: (RealPars, 2019)

A válvula direcional, também chamada de servo-válvula, é um dispositivo eletromecânico. O controlo desta é feito eletricamente, que quando energizado a corrente flui por uma bobine que gera um campo magnético. O campo magnético gerado exerce força sobre o cilindro que é obrigado a se deslocar para a sua posição de trabalho, como ilustrado na Figura 7. A bobine encontra-se no interior dos componentes elétricos na figura acima. O cilindro amovível está atracado a uma mola, que quando não existe uma força externa a atuar sobre o cilindro (por exemplo, força magnética gerada pela bobine), move-se para a posição de repouso.

Existem, na válvula apresentada acima, três orifícios onde se encontra ligado o tubo de admissão (orifício de pressão) por onde é fornecido o ar comprimido (ou um fluido no caso da hidráulica), o tubo de saída que se liga ao elemento pneumático

(orifício de trabalho) que se pretende controlar e o orifício de retorno por onde sai o ar comprimido (ou um fluido no caso da hidráulica) quando se deseja a descompressão.

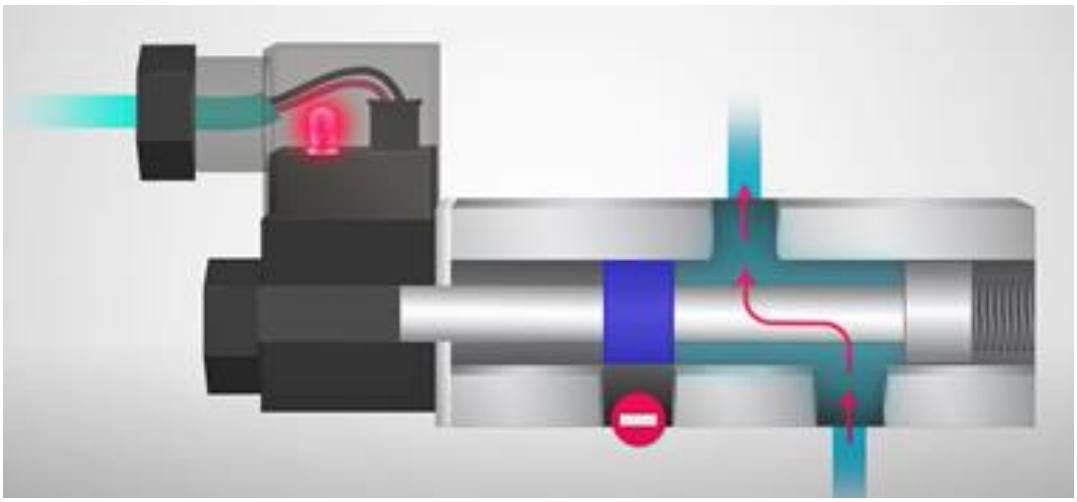


Figura 7 - Representação do funcionamento da electroválvula quando a bobine está energizada. Fonte: (RealPars, 2019)

O tubo de admissão pode trocar de posição com o orifício de retorno, sendo que apenas inverte o modo de funcionamento da válvula. Conforme as ligações, a válvula pode estar a atuar como normalmente aberta (*NO*[8]) (na posição de repouso o elemento pneumático não está sob pressão) ou como normalmente fechada (*NC*[9]) (na posição de repouso o elemento pneumático está sob pressão). A configuração da válvula direcional apresentada na Figura 6 e Figura 7, representa uma ligação normalmente aberta.

Existem tipos diferentes de válvulas direcionais, sendo que as suas diferenças são distinguidas pelo número de orifícios e o número de posições de funcionamento.

Desta forma, a válvula apresentada nas duas figuras acima, é uma válvula do tipo 3/2, porque tem três orifícios e o cilindro tem apenas duas posições de funcionamento. O facto de esta válvula quando não energizada ter apenas uma posição de funcionamento possível (posição de repouso), define-a como uma válvula monoestável. Existem também válvula biestável, estas não possuem mola e contêm uma bobine de cada lado, sendo

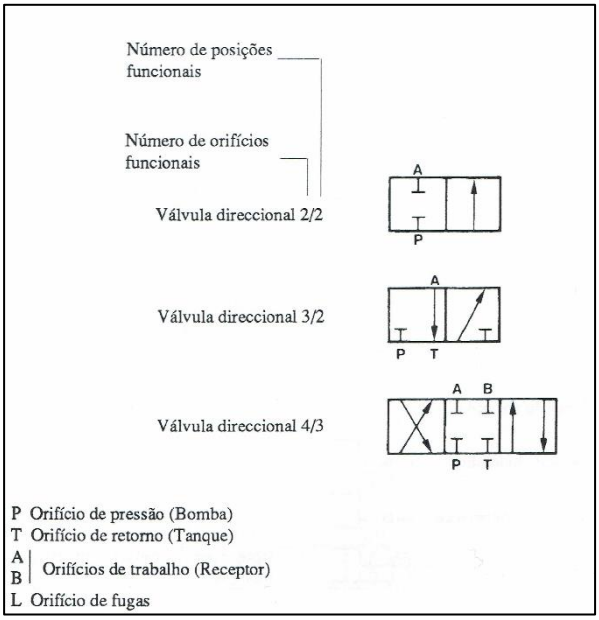


Figura 8 - Representação esquemática das válvulas direcionais. Fonte: (Gotz, 1991)

[8] *Normally open*.
[9] *Normally closed*.

que para o cilindro se mover é necessário energizar uma das bobines, consoante a direção pretendida. (Bongas, 2021; Gotz, 1991).

2.3.3 – Relés

Quanto aos sistemas elétricos da indústria 2.0, estes ainda hoje se encontram em uso na indústria. Um dos principais dispositivos de controlo elétrico que permitiu o surgimento da lógica cablada baseada em relés, é o próprio relé.

O relé é um interruptor de operação elétrica que permite que um circuito de baixa potência controle um circuito de alta potência, de forma independente e eletricamente isolados um do outro. Desta forma um circuito de BT (que pode ser proveniente de um PLC), pode controlar um circuito de AT.

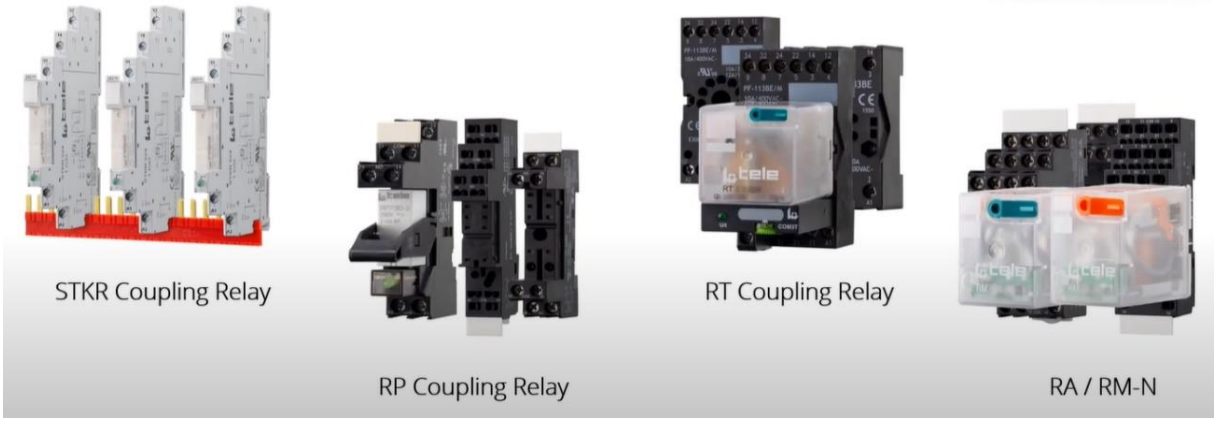


Figura 9 - Diferentes tipos de relés. Fonte: (TELE Controls Inc., 2020)

Os primeiros relés a surgir, conseguiram cumprir esta função recorrendo a um sistema eletromecânico, sendo, ainda hoje, conhecidos como relés eletromecânicos. Estes são constituídos por o lado primário com uma bobine onde o circuito de baixa potência é conectado e pelo lado secundário onde existem pelo menos dois contactos onde o circuito de alta potência é conectado.

Por norma, do lado secundário os relés têm sempre três pinos, sendo estes o comum (*COM*), o normalmente aberto (*NO*) e o normalmente fechado (*NC*), sendo que o contacto *COM* está sempre em contacto com outro contacto, seja ele o *NC* ou o *NO*.

Do lado primário existem apenas dois pinos para que a corrente passe pela bobine, sendo estes os *A1* e *A2*, conforme representado na Figura 10.

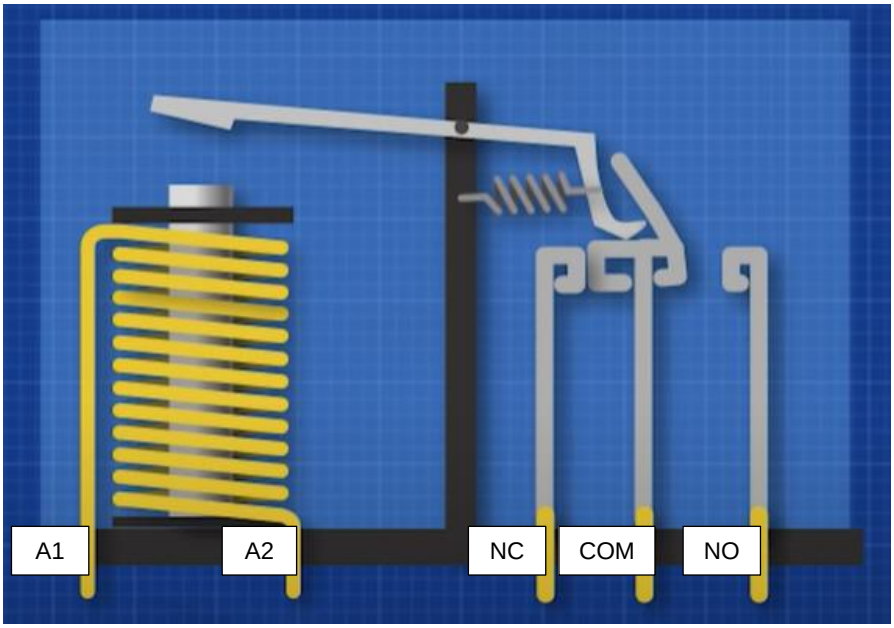


Figura 10 - Representação de um relé eletromecânico. Fonte: (The Engineering Minset, 2020)

Quando o circuito de baixa potência energiza a bobine, esta cria um campo magnético que atrai a alavanca responsável pela mudança do estado dos contactos, passando a corrente a circular entre os pinos *COM* e *NO*. Quando se para de energizar a bobine no lado primário, o campo magnético para de ser gerado e a alavanca é movida para o seu estado inicial pela força exercida pela mola. Nesta posição, no lado secundário, a corrente pode fluir pelos pinos *COM* e *NC*.

Existem também relés de *latching* que possuem uma 'memória' acerca do seu estado, mesmo que não exista nenhum campo magnético a atuar como ilustrado na Figura 11. Estes relés podem possuir uma bobine ou duas bobines, sem mola.

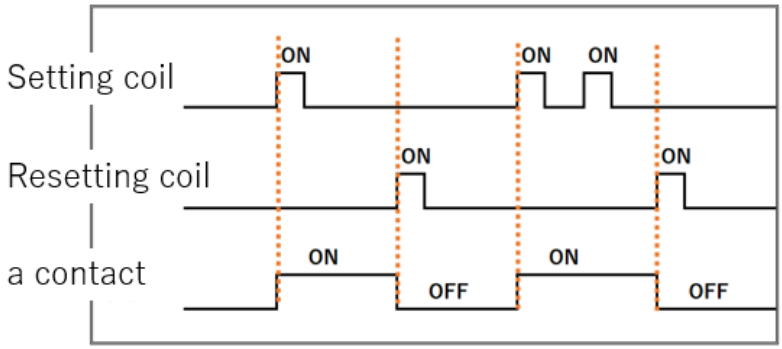


Figura 11 - Representação gráfica dos estados do relé de latching. Fonte: (Panasonic, 2020)

No caso de ter apenas uma bobine e ser um relé de *latching*, esta quando energizada com uma polaridade permite que a corrente flua no circuito secundário e quando energizado com a polaridade inversa, impede que a corrente circule no circuito secundário. Isto porque o sentido da corrente gera o sentido com que o campo magnético é criado. Consoante o sentido do campo magnético, o circuito secundário fica aberto ou fechado.

Quanto aos relés de *latching* com duas bobines, estes possuem uma bobine de cada lado, quando uma é energizada move o interruptor em um determinado sentido fechando o circuito secundário e a outra bobine quando energizada faz mover o interruptor na direção contrária abrindo o cilindro como ilustrado na Figura 12. A bobine que fecha o circuito secundário é chamada de *Setting Coil* e a bobine que abre o circuito secundário é chamada de *Resetting Coil*.

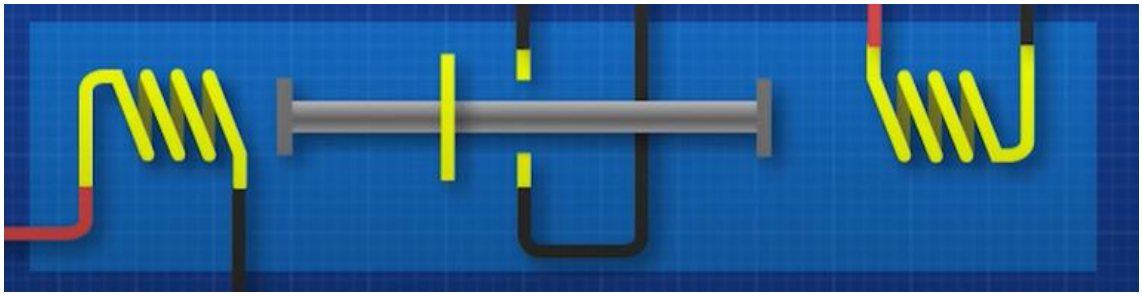


Figura 12 - Representação de um relé de latching com duas bobines. Fonte: (The Engineering Minset, 2020)

Atualmente, com os desenvolvimentos eletrónicos, existem também relés de estado sólido, sendo estes relés puramente eletrónicos e sem componentes mecânicas ou móveis. Desta forma, ao invés de termos uma bobine do lado primário, existe um LED que quando energizado emite luz. Do lado secundário existe um fototransístor onde a sua resistência varia consoante a luz que lhe incide. Desta forma quando o LED está ativo, o fototransístor permite que a corrente circule por ele, quando o LED não emite luz, o fototransístor bloqueia a passagem da corrente do lado secundário. Pode-se ver uma ilustração de um relé de estado sólido na Figura 13.

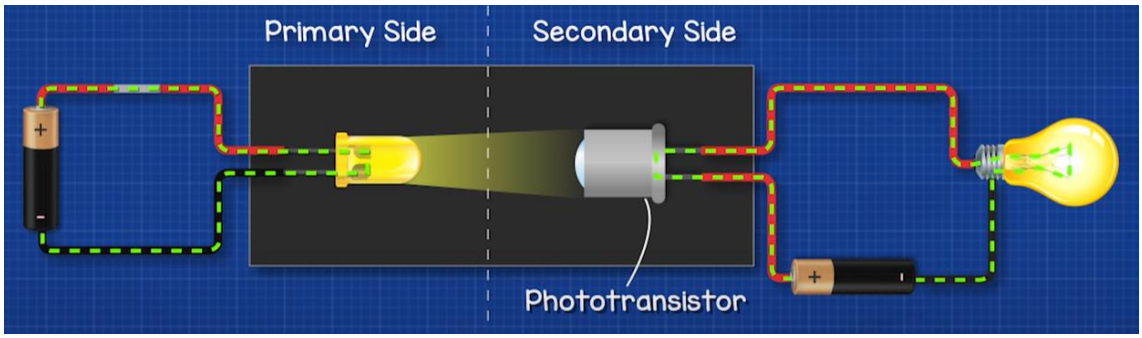


Figura 13 - Representação do funcionamento de relé de estado sólido. Fonte: (The Engineering Minset, 2020)

Dentro dos relés, existem ainda relés especiais, nomeadamente relés temporizados. Estes, por não serem de grande relevância para o entendimento do restante relatório de estágio, ficam apenas apresentados em termo de curiosidade para os que despertarem um maior interesse pelos mesmos no Anexo 1 - Relés temporizados.

Outro tipo de relés especiais, são os relés de segurança, introduzidos durante a indústria 3.0. Estes relés podem ser facilmente reconhecidos em um quadro elétrico, sendo normalmente estes de cores vivas e com vários LED, como o da Figura 14.

Estes são responsáveis pela segurança do sistema de controlo, fazendo o monitoramento ativo de circuitos elétricos. Estes circuitos podem ser botões de parada de emergência, portas de proteção, barreiras de luz, esteiras comutadoras, bimanual, etc.

O motivo pelo qual um relé eletromecânico normal não é considerado seguro ou um relé de segurança, é explicado pela empresa *Pilz Industrieelektronik S.L.* (2021) produtora de relés de segurança entre outros:

> Um relé de comutação normal utiliza uma bobina com fio e o movimento mecânico dos contactos de metal para ligar e desligar a carga. Após ciclos de ligação repetidos, os contactos de metal podem fundir. Se isso ocorrer e o operador acionar o botão de parada de emergência, a máquina continuará a funcionar. Nesse caso, seria uma condição perigosa para o operador. Por essa razão, muitos padrões e normas de segurança europeias, americanas, nacionais e internacionais impedem a utilização de relés ou disjuntores simples em máquinas perigosas.

Por este motivo, para que um relé seja considerado um relé de segurança este tem de cumprir com as exigências da EN 60947-5-1, da EN 60204-1 e da VDE 0113-1. A comutação de estado também tem de funcionar de modo redundante com auto-monitorização, o dispositivo deve continuar funcional mesmo no caso de falha de um componente e em cada ciclo de início e fim de funcionamento da máquina, este tem de verificar automaticamente se o relé do dispositivo de segurança está a abrir e a fechar o circuito corretamente (Pilz, 2021).

Figura 14 - Exemplo de relé de segurança da marca Pilz, modelo PZE X4. Fonte: (Pilz, 2021)

2.3.4 – Sensores discretos

"Os sensores são os olhos e os ouvidos, figurativamente falando, de um sistema de controlo. De uma forma geral, se se pretende controlar algo, é necessário 'senti-lo'" (RealPars, 2018). Para isto, existem inúmeros tipos diferentes de sensores que permitem ao sistema de controlo 'sentir'.

Inicialmente, apenas existiam sensores digitais, ou melhor dizendo relativamente à época da indústria 2.0, interruptores de sinais que permitem abrir ou fechar o circuito e são "utilizados para detetar duas situações exclusivas, como a presença ou ausência de uma peça ou um valor baixo ou alto de um nível. Permitem, desta forma, controlar a evolução do processo de manufatura e acionar alarmes e sistemas de segurança" (Pinto, 2021, p. 90).

Dentro destes, uns dos mais comuns que ainda hoje se encontram na indústria e foram desenvolvidos na indústria 2.0 são os sensores de fim de curso ou sensores de contacto, Figura 15. Estes "destinam-se, como o nome indica, a assinalar quando determinada parte em movimento atingiu certa posição através de um contacto físico direto. Trata-se, de facto, de interruptores atuados, não por operadores humanos, mas por objetos ou máquinas em movimento" (Pinto, 2021, p. 92). Atualmente, os sensores de fim de curso mais modernos apresentam três contactos, sendo um o comum, um normalmente aberto e outro normalmente fechado.

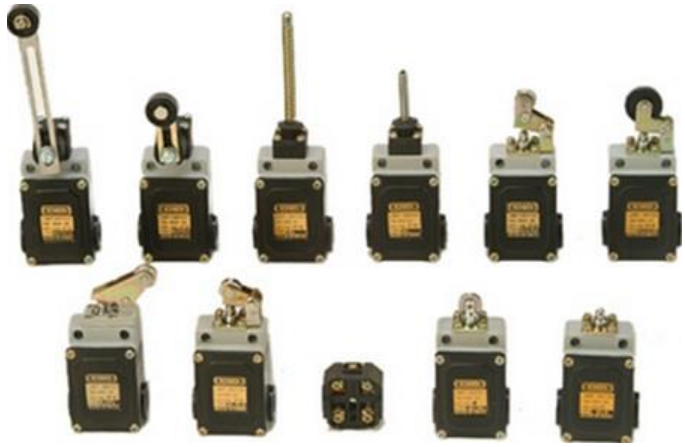


Figura 15 - Exemplos de sensores de fim de curso. Fonte: (Saber Eletrica, 2021)

Outro sensor digital/interruptor ainda hoje usado que vem desde a indústria 2.0 é o botão de pressão, que permite ao operador acionar[10] um circuito enquanto o botão estiver pressionado. Este pode ser encontrado mesmo fora da indústria, como por exemplo em uma campainha.

[10] Entenda-se como mudar o estado do circuito, abrindo ou fechando o mesmo.

Com a chegada da indústria 3.0, a introdução do PLC como instrumento principal de controlo e o desenvolvimento da eletrónica, foram desenvolvidos novos sensores digitais, os mais antigos, como os interruptores abordados acima, passaram a incluir um conversor analógico/digital de um *bit* (principalmente agora para a indústria 4.0 e para o IoT). Estes podem ser sensores magnéticos, sensores de proximidade magnéticos, sensores de proximidade capacitivos, sensores óticos e sensores ultrassónicos.

Os sensores magnéticos são constituídos por duas partes, uma contém o circuito com o interruptor magnético e a outra um íman que abre ou fecha o circuito consoante o circuito é *NC* ou *NO*, respetivamente.

Os sensores de proximidade magnéticos recorrem a um campo magnético para verificar se existe um objeto com propriedades magnéticas na sua proximidade, enquanto os sensores de proximidade capacitivos são usados para verificar se existem algum objeto não magnético na sua proximidade.

Os sensores óticos detetam a presença ou ausência de um feixe de luz e os sensores ultrassónicos detetam a presença ou ausência de ultrassons. No entanto, o primeiro como usa propriedades da luz está suscetível a que sujo bloqueie a passagem da mesma, podendo dar um falso sinal. O ultrassónico apesar de ter a capacidade de resolver problemas únicos, também tem um problema relacionado com ruído que em alguns casos o pode colocar como uma má opção.

2.3.5 – Sensores analógicos

Com a entrada da indústria 3.0 e o desenvolvimento dos semicondutores, surgiu uma nova variedade de sensores, os sensores analógicos. Estes permitem medir grandezas físicas e convertê-las em um sinal elétrico que pode ser interpretado pelo dispositivo de controlo, fazendo com que o sistema de controlo consiga manipular mais profundamente o sistema, uma vez que o ‘sente’ de uma forma mais aproximada às das sensações humanas. Por este motivo, estes novos sensores dão a capacidade ao sistema de controlo de ver mais além do que era possível com os sensores discretos.

O sinal pode ser fornecido como corrente ou tensão, consoante o tipo de sensores, sendo mais comum hoje em dia a transmissão ser feita de 4mA a 20mA, uma vez que esta é imune ao ruído, pode percorrer longas distâncias sem sofrer perdas, diferencia o valor mínimo de uma falha de ligação, entre outras características. Atualmente os sensores tendem a possuir um conversor analógico/digital que transforma a grandeza física medida em sequências de múltiplos *bits.* Esta conversão possibilita a transmissão dessa informação em formato digital, de forma que esta possa ser compreendida por um PLC (Pinto, 2021, p. 101).

Os sensores analógicos podem ser qualificados em vários grandes grupos, sendo que esta divisão é feita baseada na finalidade para qual os sensores foram desenvolvidos. Os sensores podem ser usados para várias finalidades, como: medir caudal, temperatura, nível, pressão, posicionamento, força, gás, luz, som, humidade, entre outras apresentadas na Figura 16. Dentro dos grandes grupos ainda é possível dividir em subgrupos, os sensores ativos e os sensores passivos.

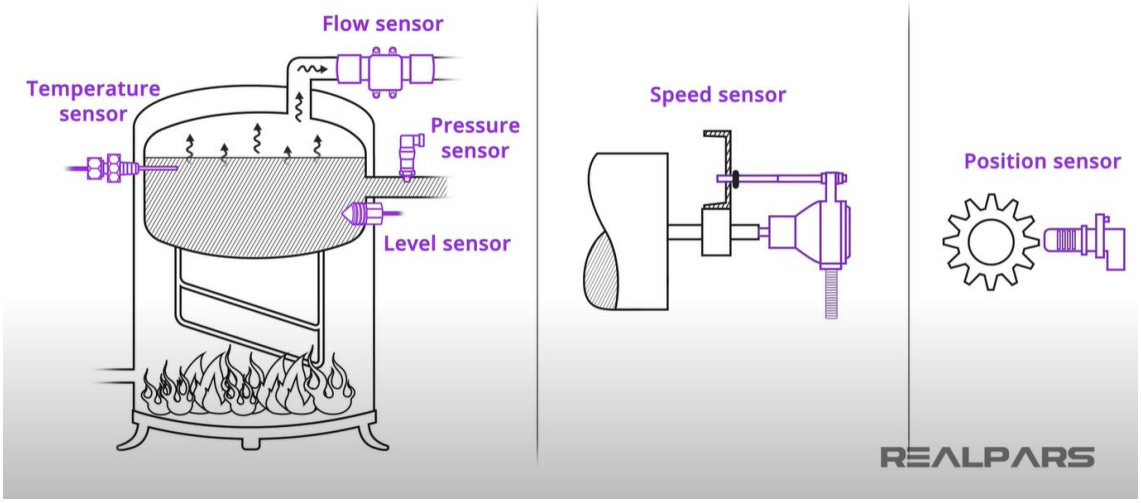


Figura 16 - Diferentes tipos de sensores. Fonte: (RealPars, 2020)

Os sensores ativos não necessitam de qualquer alimentação externa pelo que eles próprios geram o seu próprio sinal, ao contrário dos sensores passivos que necessitam de uma fonte externa para fornecerem sinal. Um exemplo é o termostato, que consoante a temperatura do seu ambiente aumenta, também a voltagem que ele gera aumenta[11], sendo ele um sensor ativo por gerar o seu próprio sinal. Já o RTD (sensor de temperatura resistivo) varia a sua resistência interna consoante a temperatura do seu ambiente varia. Deste modo, é necessária uma fonte externa que forneça tensão, possibilitando avaliar o valor da resistência do RTD para que se consiga extrair o valor da temperatura do ambiente.

2.3.6 – *Encoders*

Os *encoders* são sensores extremamente importantes e vastamente usados nas tecnologias atuais, um exemplo do seu uso são os sistemas robóticos. Estes possibilitam medir a distância, controlar velocidade, medir ângulos, posicionamento, entre outras aplicações.

A sua construção divide-se em dois grupos, os *encoders* lineares e os *encoders* rotativos, como apresentado na Figura 17. A diferença entre estes é que os rotativos são construídos em um disco, permitindo medir a rotação ou velocidade angular de

[11] A tensão de um termostato anda na casa dos mV.

veios, assim como a sua posição. Os *encoders* lineares são construídos ao longo de uma superfície retilínea e destinam-se a medir com elevada precisão, a posição de partes com movimentação linear, sendo que ambos se baseiam nos mesmos princípios.

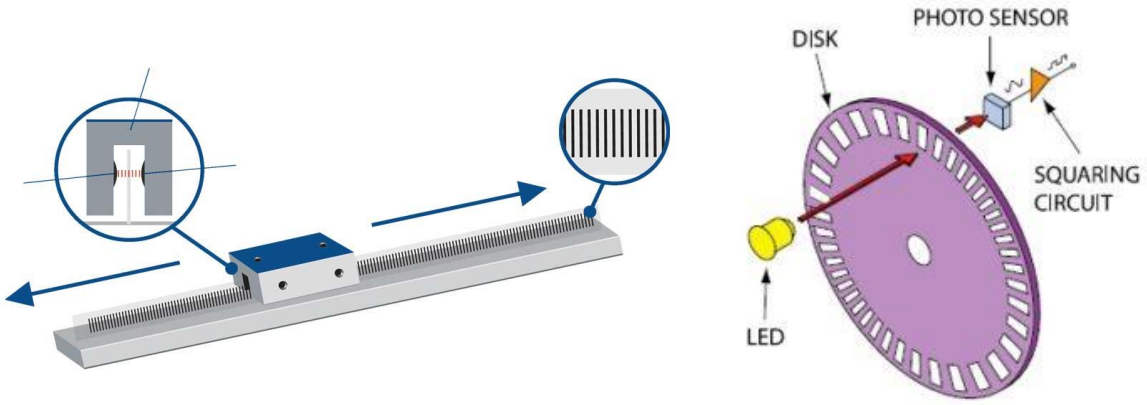


Figura 17 - Exemplo de encoder linear e rotativo, respetivamente. Fontes: (Alcantara, 2020; Schweber, 2018)

Independentemente da construção do *encoder*, estes podem ter dois princípios de funcionamento distintos, sendo eles pertencentes ao grupo dos *encoders* óticos ou *encoders* magnéticos. Desta forma, a diferença é apenas o tipo de sensor usado para captar os sinais dos *encoders*, baseando-se na obstrução ou passagem de luz, no caso dos *encoders* óticos, ou da alteração do seu campo magnético, no caso dos *encoders* magnéticos.

Os *encoders* óticos foram os primeiros a serem desenvolvidos e estes podem oferecer precisões de 0.01º. Contudo, têm a desvantagem de necessitarem de estar em ambientes relativamente limpos.

Os *encoders* magnéticos são mais robustos em termos do seu ambiente, sendo ideais para se usar em ambientes menos limpos, no entanto estes têm a desvantagem de serem menos precisos, sendo que a sua precisão máxima é de aproximadamente 0.1º.

Por fim, os *encoders* podem ser do tipo incrementais ou absolutos. A diferença é que um *encoder* incremental nunca sabe a sua posição absoluta, salvo no caso de este partir de uma referência conhecida e fizer a contagem do seu deslocamento, enquanto o *encoder* absoluto sabe sempre qual o seu posicionamento. Esta diferença está diretamente relacionada com a sua construção, sendo esta apresentada na Figura 18.

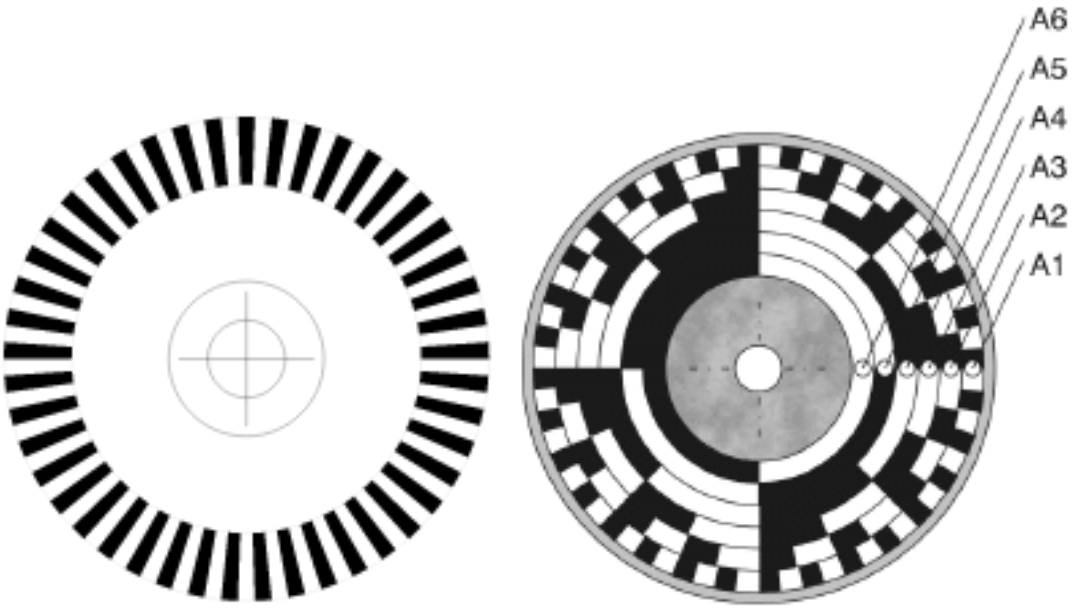


Figura 18 - Exemplo de encoder incremental e encoder absoluto de 6 bits, respetivamente. Fonte: (Oliveira, 2019; Trimble, 2013)

O número de posições conhecidas por um *encoder* absoluto é dado pela seguinte equação:

$$nº\ posições = 2^{nº\ de\ bits}$$

Equação 1 - Cálculo do número de posições conhecidas em um encoder absoluto.

Desta forma, para o caso do *encoder* absoluto de 6 bits, este tem 64 posições conhecidas.

2.3.7 – Sistemas de identificação digital

Atualmente é possível espalhar informação digital pelo chão de fábrica. Esta informação pode ser de dois tipos:

- A informação é estática e destina-se apenas a ser lida. Geralmente, corresponde a informação de identificação e a características do equipamento onde está associado;
- A informação é dinâmica, ou seja, pode ser atualizada ao longo do tempo de forma automática. (Pinto, 2021, p. 106)

Sendo que para suportar esta informação existem várias tecnologias, sendo elas:

- Código de barras;
- Código matricial;
- Etiquetas RFID;
- Etiquetas *Bluetooth*;
- Etiquetas NFC;

Para se ler os dados registados em cada um destes suportes de informação, são necessários ‘sensores’ específicos, sendo estes dotados com algum protocolo de comunicação que os permite ligar a controladores ou outros sistemas de IT.

2.3.7.1 – Código de barras e matriciais

Os códigos de barras e matriciais são largamente usados como elementos passivos, consistem em rótulos colados em produtos ou pacotes que evoluíram de códigos unidimensionais (código de barras) para bidimensionais (código *QR*). A vantagem dos códigos bidimensionais sobre os unidimensionais é a capacidade de armazenamento que este consegue suportar, podendo ser armazenados dados na vertical e na horizontal. Para ler os dados que os códigos possuem é necessário um leitor ótico, sendo que atualmente as câmaras dos *smartphones* já têm essa capacidade.

No entanto, estas tecnologias apresentam desvantagens sobre, por exemplo, etiquetas RFID. Essas desvantagens são:

- Tempo que leva para ler o código, apesar de suficientemente rápido num supermercado, não o é para linhas de montagem;
- Facilidade de serem danificados, impossibilitando a leitura do código;
- Os códigos têm de estar visíveis face ao leitor.

2.3.7.2 – Etiquetas RFID

As etiquetas/tecnologia RFID suportam informação que se destina a ser lida e, eventualmente, atualizada (escrita), através de ondas de frequência rádio. Estas etiquetas são usadas quando se pretende informar máquinas ferramentas da forma como proceder com a peça/objeto, consoante os dados da sua etiqueta RFID associada, constrói um histórico com o registo do que aconteceu a cada componente nas diferentes fases da sua vida.

Para se atingir este objetivo, Pinto (2021) afirma que há duas formas:

- A etiqueta contém um identificador único da mesma. Com base nessa identificação, programa acede a uma base de dados, atualizando o histórico, e/ou lê o procedimento a ser executado naquele instante. Vantagem desta solução: etiquetas muito simples, normalmente

passivas (logo, não alimentada por energia). A desvantagem é exigir servidores com quantidades gigantescas de informação;

- A etiqueta é capaz de enviar e receber informação. Para isso contém uma pequena antena e um microchip. Além de um elemento de identificação, guarda o histórico dos procedimentos a que foi sujeita (como que um diário) e fornece as instruções a cada agente que opera sobre ela. Vantagem desta solução: não são necessários servidores para armazenamento de dados, bastando haver leitores dessas etiquetas. Desvantagens: as etiquetas, que têm de ser ativas, são muito mais caras e difíceis de integrar. (p. 107)

2.3.7.3 – Etiquetas *Bluetooth*

As etiquetas Bluetooth, são do tipo etiquetas RFID ativas, mas que comunicam usando a emissão de sinais BLE[12] pelos seus leitores. Algumas das vantagens de estas etiquetas sobre as etiquetas de RFID são:

- Facilmente lidas por *smartphones*, *tablets* e computadores (com *Bluetooth*), sendo que como estes já têm conectividade à internet facilmente se transmite a informação;
- Os leitores têm maior alcance de leitura;
- Têm menores consumos, mesmo que envie maior quantidade de dados, uma vez que a tecnologia BLE foi desenvolvida para ter uma durabilidade de 3 anos sem recarga.
- Devido à sua produção em massa, o custo das etiquetas é muito baixo e por isso pode ser descartado após a utilização.

2.3.7.4 – Etiquetas NFC

Nos tempos atuais, os leitores de NFC são muito comuns nos *smartphones* e em cartões multibanco, sendo etiquetas muito parecidas com as etiquetas *Bluetooth*.

[12] *Bluetooth Low Energy*.

O maior problema de estas etiquetas é o seu baixo alcance de transmissão, sendo que a distância de leitura é de apenas alguns centímetros. Esta tecnologia foi desenvolvida para ser, essencialmente, de leitura, sendo que é a tecnologia que se usa nos TPA[13] quando queremos fazer o pagamento com o cartão multibanco com a tecnologia '*contactless*'. As vantagens de esta distância tão curta de leitura relativamente às outras etiquetas, é a diminuição dos erros de leitura.

2.3.8 – Outros sensores de segurança

Além de estes sensores, com as preocupações atuais de segurança, foram desenvolvidos também sensores de segurança que "numa unidade industrial são de importância vital, não só para a proteção das pessoas, mas também do próprio equipamento, permitindo ainda aumentar a eficiência da mobilidade dentro das instalações" (Pinto, 2021, p. 99).

No contexto de um dispositivo de segurança, assim como os relés de segurança, também os sensores de segurança têm características de 'seguros a falhas', ou seja, em caso de falha no sensor, o estado em que o sistema fica é seguro.

A maioria destes sensores de segurança são sensores de movimento, sendo que estes, em geral, funcionam como interruptores, parando máquinas em movimento ou acionando alarmes sonoros/luminosos.

Estes sensores usam uma tecnologia idêntica à do funcionamento dos sensores de proximidade para distâncias elevadas, é "baseada no corte de feixes de radiação eletromagnética na gama de frequências do radar, dos infravermelhos ou outras geradas por laser. Alguns podem ser baseados em ultrassons" (Pinto, 2021, p. 100).

Os sensores de segurança podem ser:

- Tiras de comutação, que contêm uma tira de borracha pelo qual atravessa um feixe de luz infravermelho, quando algo deforma a borracha esta interrompe o fecho de luz e aciona um alarme;
- Barreiras de luz, onde feixes de radiação luminosa formam uma barreira cujo corte/bloqueio aciona um alarme;
- Cortinas e grelhas de luz, onde múltiplos emissores e recetores de feixes de luz instalados em barras laterais paralelas formam uma cortina ou grelha de luz que quando cortada/bloqueada aciona um alarme;
- *Scanners laser* que são emissores de radiação *laser* e recetores dos feixes refletidos, que quando obstruídos acionam um alarme.

[13] Terminal de Pagamento Automático

2.3.9 – AGV

Outra grande evolução que chegou aquando da indústria 3.0 foram os veículos guiados automaticamente, sendo estes conhecidos como AGV. “Os AGV são dispositivos destinados ao transporte de matérias-primas, produtos finais ou partes a serem processadas, que se movimentam automaticamente” (Pinto, 2021, p. 16).

Os primeiros AGV guiavam-se por um fio de cobre estendido no chão, este gerava um campo magnético que os AGV captavam, guiando-se por ele. Outra forma de os guiar é colocando balizas ao longo de todo o trajeto.

Nos finais da indústria 3.0, já se permitia a que os AGV se deslocassem mais livremente e em ambientes cada vez menos estruturados. Desta forma, os AGV apenas sabiam o seu ponto de partida (posição atual) e o ponto de chegada e a partir disto, o próprio calculava o melhor percurso e deslocava-se até lá, naturalmente a desviar-se de obstáculos e evitando colisões. Isto foi possível devido à evolução da odometria que permite medir o deslocamento através de *encoders*, colocados em geral nas rodas dos AGV, que trabalham cooperativamente com sensores de visão, sensores de ultrassons e também informação obtida por GPS[14] no caso de AGV exteriores.

2.3.10 – Armazéns automáticos

Por fim, o último elemento de automação da indústria 3.0, excluindo as tecnologias que terão subcapítulos dedicados, são os armazéns automáticos.

> Até à 3ª revolução industrial eram operários que faziam a recolha e o depósito dos produtos, o que significava que tinham de ter acesso facilitado aos armazéns. A sua dimensão tinha de ser grande, atendendo a que, ao não haver grande capacidade de previsão das necessidades de consumo e das encomendas, era preciso armazenar largos stocks. (Pinto, 2021, p. 21)

A automação e a indústria 3.0, permitiram automação dos armazéns devido a dois componentes principais:

- Automação do movimento físico dos produtos;
- Automação dos processos de recuperação e inventariação;

[14] *Global Positioning Systems*.

Estes avanços no armazenamento de armazéns inicialmente eram feitos recorrendo a empilhadores conduzidos por operadores, apoiados por sistemas de guiamento, e já numa fase mais recente, feito por AGV.

Em armazéns mais avançados, pode chegar a haver elevadores, tapetes rolantes e, eventualmente robôs no processo de enchimento e deslocamento de paletes. A este processo de armazenar e recuperar peças é normalmente chamado pela sigla AS/RS (*Automated Storage/Retrieval System*).

Atualmente, está-se a abandonar o termo armazém e a preferir-se usar o termo centro de distribuição, segundo *Supply Chain Minded* no livro de Pinto (2021, pp. 22-23), estas são as diferenças entre eles:

- Um armazém é usado para armazenar produtos, enquanto um centro de distribuição, além de armazenar produtos, oferece serviços como a reunião de produtos diferentes para os fornecer em conjunto (*product mixing*), a receção seguida de expedição imediata dos produtos (*crossdocking*), a embalagem, etc.;
- Um centro de distribuição mantém os produtos, em geral, durante bastante menos tempo do que um armazém. Por isso, a velocidade de circulação dos produtos é muito maior nos centros de distribuição;
- Os centros de distribuição estão centrados no cliente. Enquanto o papel de um armazém é armazenar produtos com eficiência, o papel dos centros de distribuição é atender com eficiência os pedidos dos clientes, sejam eles externos sejam a própria linha de produção;
- As operações num centro de distribuição são muito mais complexas do que num armazém, sendo necessariamente dotados de tecnologia mais avançada.

2.3.11 – Elementos da automação industrial introduzidos pela indústria 4.0

Na indústria 4.0, os equipamentos de automação são os mesmos. “O que essencialmente distingue as versões modernas destes equipamentos, face às anteriores, é que agora constituem sistemas ciberfísicos, ou seja, são sistemas inteligentes, complexos e capazes de se ligar ao mundo das coisas e das pessoas” (Pinto, 2021, p. 24), fazendo parte integrante da IoT.

De esta forma, as grandes novidades da indústria 4.0, são a manufatura aditiva, impressoras 3D e os *drones*. “O conceito de ‘manufatura aditiva’ deriva formalmente do conceito mais antigo de ‘prototipagem rápida’” (Pinto, 2021, p. 26), hoje em dia, esta manufatura, é conhecida como impressoras 3D. Não é que as impressoras 3D sejam propriamente um conceito novo, na verdade estas foram desenvolvidas na década de 80 do século XX, contudo, apenas em 2017 se tornaram financeiramente acessíveis a pequenas e médias empresas. Estas impressoras são conhecidas geralmente por fazerem injeção de plástico e construírem peças baseadas em ficheiros CAD, no entanto hoje já é possível o depósito simultâneo de diferentes tipos de materiais. “Um exemplo recente da sua utilidade, com grande impacto social, foi a sua utilização na construção de ventiladores para combater a pandemia de COVID-19, em 2020” (Pinto, 2021, p. 26).

Já os *drones* são veículos aéreos não tripulados que integram controladores, sistemas de comunicação e uma vasta gama de sensores de posição, movimento e de visão. Os *drones* inicialmente foram desenvolvidos como brinquedos, mas atualmente são usados como ferramentas eficazes na produção industrial, sobretudo na área de inspeção. Um exemplo do seu uso é para inspecionar o estado de linhas de alta tensão, um trabalho que, para o ser humano é complicado, demorado e lento (Pinto, 2021, p. 27). A presença de *drones* em toda a indústria é algo que se prevê para o futuro, no entanto o maior ‘calcanhar de Aquiles’ neste momento em relação a esta tecnologia são as capacidades das baterias.

2.4 – PLC

2.4.1 – História do PLC

Como explicado anteriormente, o PLC surgiu na indústria 3.0, principalmente na indústria automóvel. Estes ainda hoje são usados em larga escala na automação industrial (Francisco, 2015, pp. 1-2).

Figura 19 - PLC da Siemens, modelo SIMATIC S7-1200. Fonte: (Siemens Brasil, 2021)

Inicialmente os PLC foram usados para substituir a lógica cablada (baseada em relés), mas esta mudança também trouxe amplas funcionalidades adicionais sobre a lógica cablada, o que fez com que estes estivessem mais presentes e em aplicações mais complexas (Petruzella, 2011, p. 2).

As razões que levaram à rápida adoção pela indústria a esta nova tecnologia, foram o facto de este ser um 'computador' desenvolvido especificamente para controlar e funcionar num ambiente industrial, tendo a capacidade de múltiplas E/S, trabalhar em temperaturas extremas, ter imunidade ao ruído elétrico, e ser resistente a vibrações e impactos.

Como a estrutura do PLC implementa os mesmos princípios usados em um computador, este para além de ser capaz de realizar todas as tarefas realizadas anteriormente pela lógica cablada (baseada em relés), também é capaz de realizar outras tarefas como contagens, temporizações, cálculos matemáticos, comparações e processar sinais analógicos (Petruzella, 2011, p. 2).

Denota-se uma concordância entre autores no que diz respeito aos benefícios dos PLC sobre a lógica cablada, sendo eles:

- **Maior Fiabilidade**:
    - O sistema contém menor número de peças mecânicas em movimento e menos componentes;
    - Uma vez que o programa foi escrito e testado este pode ser transferido facilmente para outros PLC;
    - Em termos de ligações da cablagem, esta ainda que necessária exige menor quantidade e é fácil de reproduzir para outro sistema industrial idêntico;

  Isto traz maior robustez ao sistema, traduzindo-se em maior fiabilidade, os componentes do sistema operam durante anos antes de falharem (Francisco, 2015, p. 5; Jack, 2008, p. 20; Petruzella, 2011, p. 2; Parr, 1993, pp. 6 - 7);

- **Facilidade de modificação**: é fácil de criar e alterar um programa em um PLC ao invés da lógica cablada, que necessita de modificar as ligações e/ou fazer novas. Dentro de um PLC a relação das E/S é determinada pelo programa ao invés de como estão ligados os cabos. Também novas funcionalidades ou alteração de condições de funcionamento do programa são fáceis de implementar (Francisco, 2015, p. 5; Petruzella, 2011, p. 2);
- **Menor espaço**: os PLC ocupam menos espaço dado às suas dimensões reduzidas (Francisco, 2015, p. 5);
- **Manutenção facilitada:** facilidade de realizar manutenção, descobrir problemas e de não criar problemas, uma vez que os PLC contêm sinalizadores na sua face que indicam a existência de sinais elétricos, ou a falta deles, nas E/S e nas informações sobre o estado de funcionamento do autómato. Também o programa tem ferramentas que facilitam a tarefa de *debug* (Francisco, 2015, p. 5; Jack, 2008, p. 20; Petruzella, 2011, p. 3);
- **Reduzido custo:** a instalação é mais simples, rápida e com menos componentes, o que faz com que tenha um custo de instalação mais reduzido, relativamente à lógica cablada (baseada em relés) (Jack, 2008, p. 20; Petruzella, 2011, p. 3);
- **Capacidade de comunicação:** os PLC têm capacidade de comunicar com outros dispositivos, como PLC, HMI, Robôs, etc. e capacidade de comunicar com uma rede de forma a fazer o *download* e *upload* de programas (Petruzella, 2011, p. 3);
- **Resposta em 'Tempo-Real':** os PLC foram desenvolvidos para ter alta-velocidade e resposta em 'tempo-real'. Isto significa que um evento desencadeado pode ser respondido em um exato momento[15] por uma operação e/ou a modificação do estado de uma ou mais saídas. Uma vez que a maquinaria industrial pode chegar a processar milhares de peças por segundo e essas peças podem estar no campo de atuação de um sensor apenas por uma fração de segundo, os PLC e os seus componentes têm de ser igualmente ou ainda mais rápidos no tempo de resposta. Ainda que esta não seja uma vantagem sobre a lógica cablada, é uma caraterística bastante importante do PLC (Petruzella, 2011, p. 3; Parr, 1993, pp. 6 - 7);

[15] A afirmação "respondido em um exato momento" neste contexto é que a resposta ao evento acontece em uma velocidade superior à de a perceção humana se aperceber que determinado evento aconteceu. Por norma os tempos de resposta andam na volta de 0.005 a 20 milissegundos, dependendo do tamanho do programa, número de operações e do processador do PLC. Em casos extremos, o ciclo do programa pode chegar e ultrapassar os 50 milissegundos, mas a principal causa de isto acontecer é má programação e programação mal otimizada.

2.4.2 – Arquitetura do Autómato

Quando falamos de um PLC, normalmente referirmo-nos a múltiplos componentes que se interligam entre si e que formam um PLC funcional. Entre esses componentes, existem cinco que se encontram presentes em todos os PLC, sendo eles:

- **Unidade Central de Processamento (CPU):** também conhecida como "microprocessador, é o 'cérebro' do autómato, realiza operações aritméticas e lógicas e funções de controlo. (…) Tem ainda a seu cargo a gestão dos periféricos e o diagnóstico dos defeitos que possam ocorrer internamente" (Francisco, 2015, p. 3). Este também é o responsável por ler os valores lógicos das entradas presentes na memória, executar as instruções do programa e atualizar os valores das saídas na memória (Francisco, 2015, p. 3; Petruzella, 2011, p. 5);
- **Memória do Autómato:** são um conjunto de memórias que asseguram o funcionamento do PLC. Entre elas existem:
  1. *Memória de sistema*: onde se encontra o sistema operativo do autómato;
  2. *Memória de programa*: que contém o registo de toda a programação feita pelo utilizador e o que é pretendido ao autómato realizar;
  3. *Memória de dados*: onde são guardados os valores de dados das E/S, os resultados das operações realizadas pelo CPU e os dados necessários à execução do programa (Francisco, 2015, pp. 3-4);
- **Entradas/Saídas (E/S):** "As E/S asseguram a integração direta do autómato no seu ambiente de trabalho" (Francisco, 2015, p. 4). As entradas trazem as informações do mundo exterior até ao processador e as saídas que levam a informação processada pelo PLC até ao mundo exterior (Parr, 1993, pp. 7-9);
- **Fonte de Alimentação:** a fonte de alimentação é responsável por manter o autómato ligado, com as tensões necessárias para o funcionamento do autómato, proporcionadas a partir da tensão da rede elétrica. Existem dois tipos de autómatos. Os autómatos alimentados a 24 VDC com fonte de alimentação externa, e a 230 VAC com fonte de alimentação interna (Francisco, 2015, p. 4);
- **Periféricos:** estes são dispositivos que se conectam ao autómato através das suas portas de comunicação. O periférico mais comum de se utilizar, hoje em dia, é o computador, dado que isto é o meio mais comum para a programação do autómato (Francisco, 2015, p. 4).

Desta forma, o modo de funcionamento do autómato é relativamente simples e pode ser explicado pela Figura 20.

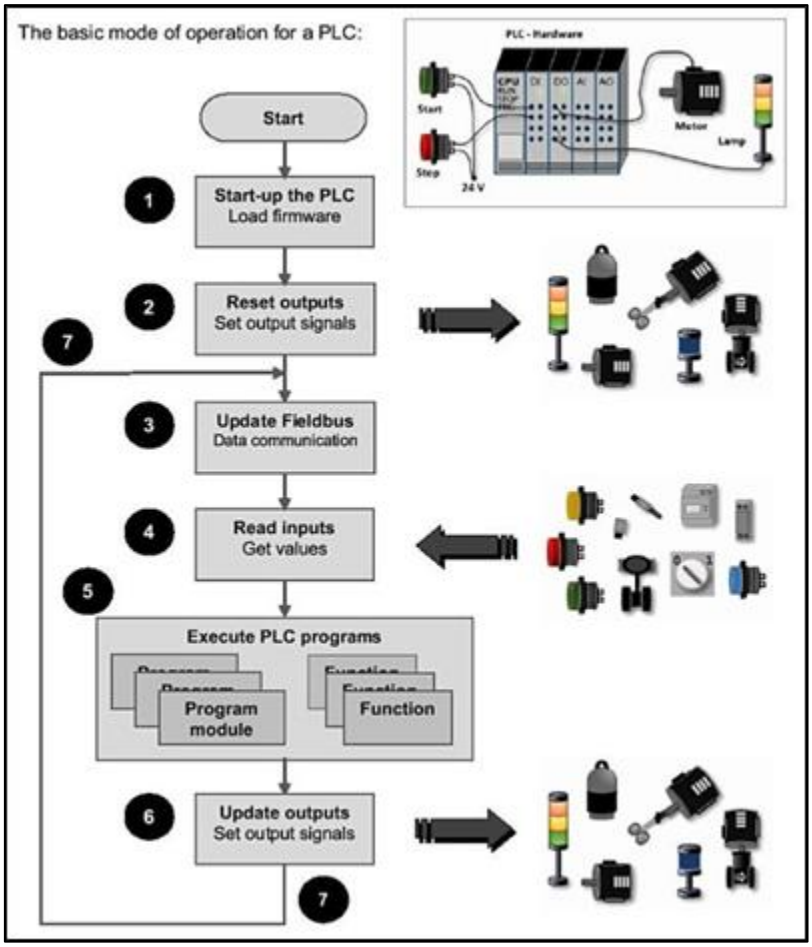


Figura 20 - Arquitetura do modo de funcionamento do PLC. Fonte: (Antonsen, 2020, p. 10)

Explicando o que acontece em cada passo da imagem anterior, obtemos a seguinte sequência:

1. Representa a inicialização do PLC a frio[16], onde o PLC apenas está a carregar o seu *firmware* para arrancar com o seu SO[17];
2. Neste passo o PLC coloca todas as suas saídas e as saídas dos seus periféricos sem tensão. Contudo, a maioria dos PLC é possível configurar se esta instrução acontece, se as saídas são inicializadas no seu valor padrão pré-programado ou se os valores das saídas permanecem iguais ao momento antes do PLC ser colocado no modo *STOP*[18];
3. Este é o primeiro passo dentro *loop* do funcionamento do PLC, sendo que é quando se sucede a atualização dos dados que vêm através das redes de campo (*Fieldbus*). Em alguns casos é possível também inicializar o PLC a partir deste ponto, sendo este designado por arranque a quente[19];

[16] *Cold Start.*
[17] Sistema Operativo.
[18] Modo em que o PLC não está a executar o seu programa.
[19] *Warm Start.*

4. Neste passo o PLC lê os valores das suas entradas e entradas dos periféricos, sendo que em nenhum dos passos seguintes esta informação é atualizada novamente, até que se conclua o ciclo;
5. Aqui os programas e funções desenvolvidas pelo programador são finalmente executadas, sendo que a sua ordem de execução é sequencial e nos PLC mais recentes, a ordem da chamada é definida pelo programador, sendo que podem existir programas ou funções presentes no PLC que não sejam chamadas;
6. Ao fim de correr os programas, o PLC atualiza os valores das suas saídas e das saídas dos seus periféricos consoante os resultados obtidos dos programas;
7. Por fim o ciclo repete-se a partir do passo 4.

Existem também tarefas[20] que podem ser incluídas no funcionamento do PLC que são executadas quando determinado evento acontece, sendo que estas realizam uma interrupção no PLC e são executadas prioritariamente. Dentro do IDE *TIA-Portal V15.1*, a figura seguinte representa as tarefas[20] (chamados de *Organization Blocks* no *TIA-Portal*) possível de adicionar ao funcionamento do PLC.

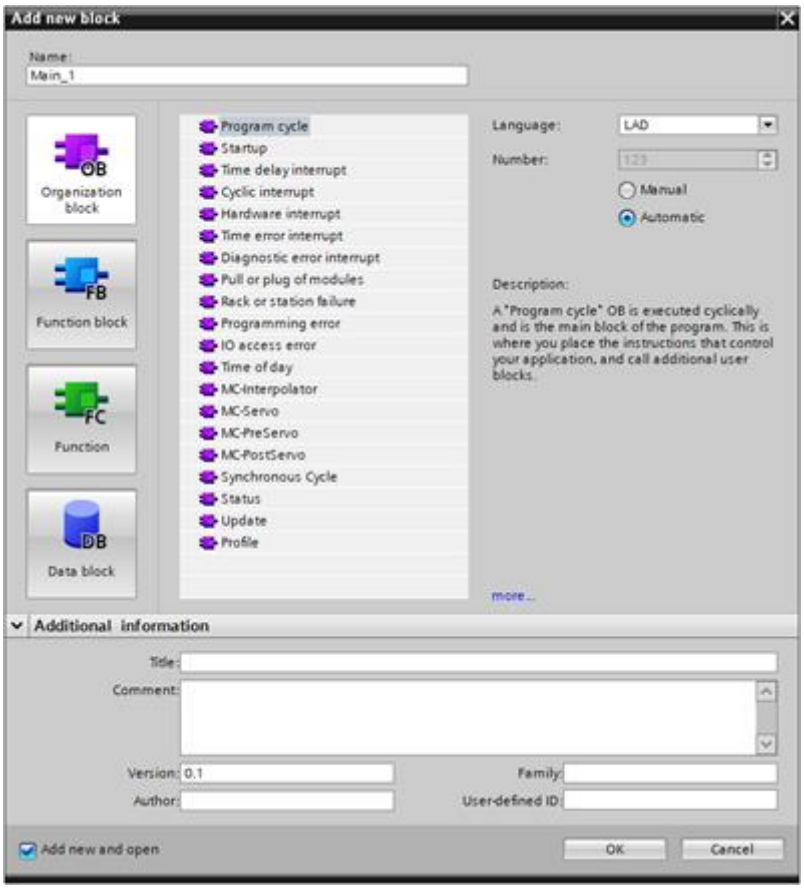


Figura 21 – Tarefas possíveis de criar no TIA-Portal V15.1, para o PLC - CPU 1518-4 PN/DP. Fonte: TIA Portal V15-1

[20] *Task.*

2.4.3 – Programação de Autómatos e IEC 61131-3

Como já referido, os PLC foram desenvolvidos para oferecer uma prática, flexível e programável opção à indústria, sendo que estes evoluíram com o tempo de uma forma em que muitos PLC modernos, e de alto desempenho, são na verdade computadores a correr os mais recentes sistemas operativos juntamente com *software* de resposta em tempo-real (por exemplo, os 'PLC', ou melhor dizendo, os computadores, da *Beckhoff* - *TwinCat 3* que correm em cima de sistemas operativos como *Windows 10*, *Windows 7* e *Windows CE* da *Microsoft* (Beckhoff, 2021)) (Tisserant, Bessard, & Sousa, 2007, p. 183).

No entanto, no início do surgimento dos PLC, existia uma grande barreira, sendo esta a falta de padronização dos sistemas. Com a diversidade de linguagens de programação no mercado e os múltiplos fabricantes a desenvolver PLC sem que houvesse uma padronização na linguagem de programação a ser usada, passou a existir uma grande curva de aprendizagem dentro das tecnologias de cada fabricante, uma vez que os conhecimentos desenvolvidos para o PLC de um fabricante, podiam não ser de forma alguma compatíveis com o PLC de outro fabricante.

Outro problema que surgiu foi o facto de que quem fazia a automação industrial na época, ou melhor dizendo, quem fazia os circuitos de automação em lógica cablada (baseada em relés), eram os eletricistas, que não tinham conhecimento em programação. Para pôr fim a isto, o IEC (*International Electrotechnical Commission*) desenvolveu uma coleção de padronizações com a intenção de criar uma experiência comum a todos os programadores de automação industrial (Tisserant, Bessard, & Sousa, 2007, p. 183).

Esta padronização das linguagens de programação dos PLC, atualmente conhecida como IEC 61131-3, foi gradualmente criada com a união de várias padronizações de diferentes países ao longo de muitos anos, como ilustrado na Figura 22. A mesma permite que todas as arquiteturas e modos de funcionamento de todos os PLC sejam semelhantes. No entanto, nem todos os fabricantes produtores de PLC implementam todas as padronizações definidas pela norma.

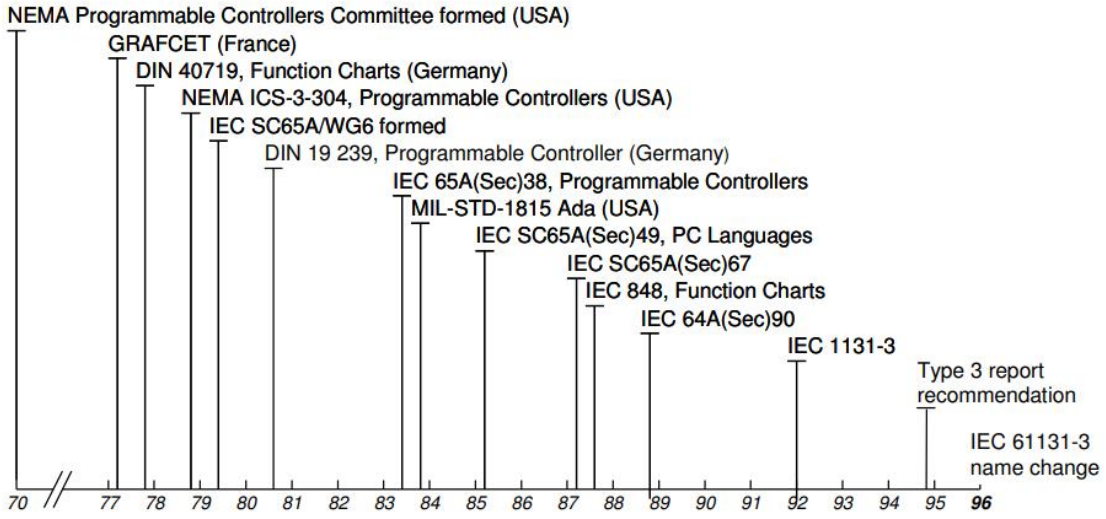


Figura 22 - Gráfico temporal do desenvolvimento das normas relativamente às linguagens de programação dos PLC até à IEC 61131-3. Fonte: (Universidade de São Paulo)

Atualmente, as empresas fabricantes de PLC, também produzem um IDE[21] específico para trabalhar com a sua tecnologia, sendo que o PLC é vendido à unidade e o IDE é vendido por licença. O IDE consiste em uma interface gráfica do utilizador (GUI) e um compilador. O GUI permite programar em qualquer uma das linguagens de programação definidas pelo PLC, enquanto o compilador é responsável por compilar o código escrito em cada programa em código máquina que pode ser executado pelo PLC (Tisserant, Bessard, & Sousa, 2007, p. 183).

O IEC 61131-3 define cinco linguagens de programação padronizadas nos PLC, sendo duas linguagens de programação textuais, duas linguagens de programação de blocos e uma linguagem sequencial de blocos. Diferentes linguagens permitem atingir diferentes métodos de programação, sendo que nem sempre é possível fazer uma tradução direta de uma linguagem para outra, devido a funcionalidade diferenciadas entre as linguagens (Crispin, 1997, p. 53).

De seguida serão apresentadas as cinco linguagens de programação padronizadas pela IEC 61131-3, no entanto, é necessário olhar aos elementos comuns a todas as cinco linguagens primeiro.

2.4.3.1 – Elementos comuns entre as linguagens de programação definidas pelo IEC 61131-3

Existem três elementos que são comuns às cinco linguagens de programação, sendo elas identificadores, tipos de dados e as variáveis.

Os identificadores são os nomes dados pelo programador para se referir a variáveis, blocos de funções, programas, etc. A norma do IEC especifica que estes

[21] Ambiente Integrado de Desenvolvimento (*Integrated Development Environment*).

identificadores devem ter um tamanho de pelo menos seis caracteres, sendo o seu conjunto único. Estes podem ser letras ou caracteres numéricos (que estejam incluídas nas primeiras 127 posições da tabela de ASCII), ou o *underscore*, existindo apenas uma restrição de que o primeiro caracter do identificador não pode ser numérico.

Os tipos de dados, são os tipos de dados que a norma obriga a implementar, sendo que diferentes tipos de dados alocam diferentes comprimentos de memória. Os dados que a norma IEC 61131-3 obriga a implementar são os seguintes apresentados na Figura 23:

IEC standard data type	*Description*
INT	Integer or whole number (–32768 to 32767)
SINT	Short integer (–128 to +127)
DINT	Double integer (-2^{31} to $+2^{31}-1$)
LINT	Long integer (-2^{63} to $+2^{63}-1$)
USINT	Unsigned short integer (0 to 255)
UINT	Unsigned integer (0 to $2^{16}-1$)
UDINT	Unsigned double integer (0 to $2^{32}-1$)
ULINT	Unsigned long integer (0 to $2^{64}-1$)
REAL	Real or floating point number (-10^{-38} to $+10^{38}$)
LREAL	Long real number (-10^{-308} to $+10^{308}$)
TIME	Time duration in days(d), hours(h), minutes(m), seconds(s) and milliseconds(ms)
DATE	Calendar date
TIME_OF_DAY (or TOD)	Time of day as obtained by a system real-time clock
DATE_AND_TIME (or DT)	Date and time of day
STRING	Character string to store textual information
BOOL	Bit string of one bit; e.g. can have one of two states: TRUE (1) or FALSE(0)
BYTE	8-bit binary string
WORD	16-bit binary string
DWORD	32-bit binary string
LWORD	64-bit binary string
F_EDGE	Falling edge
R_EDGE	Rising edge
ANY	Overloaded variables in functions or function blocks

The IEC standard allows PLC memory locations to be referenced directly and the % character is used for this purpose. As discussed in Chapter 2, PLC memory is organized into three main regions, namely input image memory (I), output image memory (Q) and internal memory (M). A memory location is identified by selecting options from the following list:

% (I or Q or M) (X or B or W or D or L) (one or more numeric fields)

Figura 23 - Tipos de dados padronizados pela norma IEC 61131-3 e as três principais regiões de memória. Fonte: (Crispin, 1997, p. 55)

Por último, temos as variáveis que permitem identificar dados de objetos, mesmo que o seu conteúdo mude. Estas contêm um indentificador e um tipo de dados, pelo que este pode ser *real*, *int*, *bool*, etc. (Crispin, 1997, pp. 54-55). As variáveis são sempre acessíveis independentemente da linguagem de programação que esteja a ser usada, exceto se estas pertencerem a um escopo inalcançável.

Desta forma, entendendo os elementos comuns entre as cinco linguagens, podemos começar a analise das linguagens.

2.4.3.2 – *Ladder Logic* (LD)

Ladder Logic, é uma linguagem de blocos e foi a resposta ao problema, acima mencionado, de que a automação industrial, na época em que o PLC surgiu, era feita por eletricistas. Esta linguagem de programação foi totalmente desenvolvida para combater esse problema, sendo que esta é fácil de sintaxe, intuitiva e o seu funcionamento é baseado no funcionamento dos relés. Desta forma, facilmente os eletricistas, já experientes na área, com uma baixa curva de aprendizagem, conseguiam evoluir da lógica cablada para a programação de PLC com mais facilidade.

De facto, esta linguagem de programação é tão simples que, ainda hoje, é a mais usada dentro da automação industrial para programar PLC. Para se programar nesta linguagem é necessário colocar blocos seguidos uns dos outros conectados entre si por uma linha, que é representativa de um cabo que transmite energia. Desta forma, alguns blocos comportam-se como interruptores do circuito, consoante a sua variável, impedindo que a energia chegue aos blocos mais à frente, enquanto outros blocos escrevem nas variáveis. É apresentado um exemplo na Figura 24.

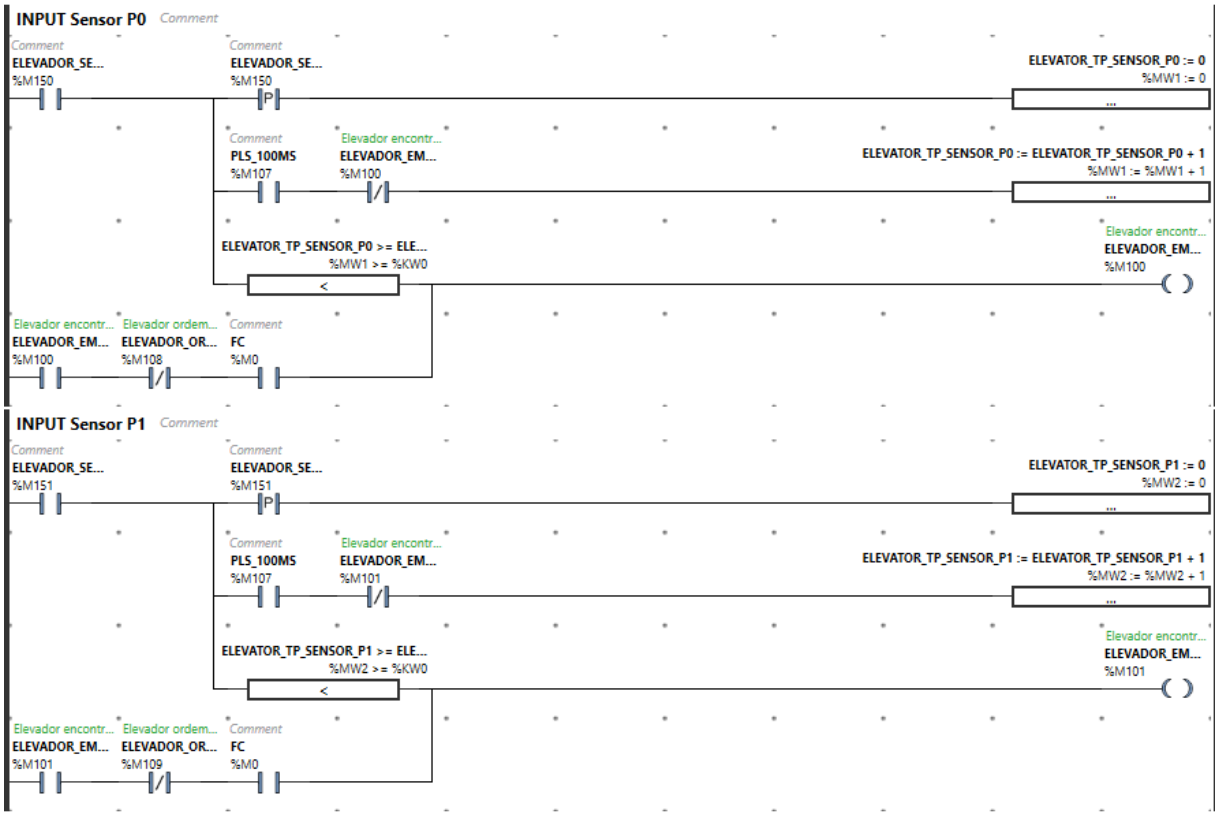


Figura 24 - Exemplo de um programa em Ladder Logic. Fonte: Própria.

Ainda que existam muitos blocos de código diferentes, com diferentes funcionalidades, existem cinco blocos que representam entre 50 a 90% dos blocos. Estes cinco elementos mais usados são:

1. Contacto normalmente fechado;
2. Contacto normalmente aberto;
3. *Coil*;
4. *Set Coil*;
5. *Reset Coil*;

Para os que queiram ir mais a fundo neste tema, dado que é a linguagem de programação mais usada nos PLC, no

Anexo 2 - Por dentro de Ladder Logic é abordado o modo de funcionamento dos blocos acima referidos, assim como a sua recriação em uma linguagem de programação de alto nível, mostrando exemplos da implementação dos seus códigos e são apresentados os seus resultados.

2.4.3.3 – *Strutured Text (ST)*

Strutured Text (será abordada como ST daqui para a frente) é uma linguagem de programação textual, a de mais alto-nível padronizada pela IEC 61131-3 e de sintaxe parecida com *Pascal*[22]. Esta usa palavras-chaves como *IF*, *THEN*, *CASE*, *FOR*, *WHILE* e *REPEAT*. Estas palavras-chaves permitem construir códigos avançados e complexos.

Apesar da linguagem estar padronizada, nem sempre é fácil trabalhar com esta, dado que não existe a padronização das livrarias que a linguagem possui. Por este motivo, existem livrarias desenvolvidas pelos fabricantes que são pertencentes ao seu IDE. Isto cria uma barreira na utilização desta linguagem, pelo que muitas vezes não existem funcionalidades equivalentes entre as IDE de diferentes fabricantes. Um exemplo do referido é a função *SIZE* usada no Bloco de Código 32, apresentada no *Anexo 3 - Implementações em Strutured Text comparadas com C#*, na página 165, que não pode ser encontrada em outro IDE.

Em ST, a estrutura do programa, em geral, é da forma apresentada no Bloco de Código 1:

[22] Linguagem de programação, orientada a objetos, criada em 1970.

```
PROGRAM NomeProg
    VAR_INPUT
        (*Lista com as variáveis de entrada*)
    END_VAR
    VAR_OUTPUT
        (*Lista com as variáveis de saída*)
    END_VAR
    VAR
        (*Lista com as variáveis locais do programa*)
    END_VAR

    (*Corpo do programa*)
END_PROGRAM
```

Bloco de Código 1 - Exemplo da estrutura de um programa em strutured text pela norma IEC 61131-3.

As funções (ou métodos) são iniciadas com a palavra-chave *FUNCTION_BLOCK* e terminam com o *END_FUNCTION_BLOCK*. Da mesma forma, praticamente todas as palavras-chaves têm uma palavra-chave que representa a terminação do seu bloco/escopo com a seguinte estrutura:

END_<palavra-chave>

Alguns exemplos disto são apresentados no Bloco de Código 2, Bloco de Código 3, Bloco de Código 4, Bloco de Código 5 e Bloco de Código 6. A terminação de expressões é delimitada com o uso do ponto-e-vírgula (;).

Serão de seguida apresentadas as estruturas das palavras-chaves anteriormente apresentadas, sendo apresentadas as estruturas condicionais primeiros e as estruturas de repetição de seguida.

```
IF <expressão booleana verdadeira> THEN
    <expressão>
ELSIF <expressão booleana verdadeira> THEN
    <expressão>
ELSE
    <expressão>
END_IF
```

Bloco de Código 2 - Apresentação da estrutura condicional IF-ELSE na linguagem de programação Strutured Text.

```
CASE <expressão numérica> OF
    <expressão>
<valor> : <expressão>
<valor> : <expressão>
<valor> : <expressão>
ELSE
    <expressão>
END_CASE
```

Bloco de Código 3 - Apresentação da estrutura condicional SWITCH-CASE na linguagem de programação Strutured Text.

```
FOR <index> := <inicio> TO <final> BY <incremento> DO
    <expressão>
END_FOR
```

Bloco de Código 4 - Apresentação da estrutura de repetição FOR na linguagem de programação Strutured Text.

```
WHILE <expressão booleana verdadeira> DO
    <expressão>
END_WHILE
```

Bloco de Código 5 - Apresentação da estrutura de repetição WHILE na linguagem de programação Strutured Text.

```
REPEAT
    <expressão>
UNTIL <expressão booleana falsa>
END_REPEAT
```

Bloco de Código 6 - Apresentação da estrutura de repetição REPEAT na linguagem de programação Strutured Text.

Estas estruturas permitem muita liberdade e controlo no código (Crispin, 1997, pp. 58-60).

Se existir um maior interesse por esta linguagem de programação, o *Anexo 3 - Implementações em Strutured Text comparadas com C#*, na página 164, apresenta algumas comparações entre ST e *C#*, sendo demonstradas alguns problemas existentes com a linguagem ST, principalmente quando se pretende trabalhar com códigos flexíveis.

2.4.3.4 – *Instruction List (IL)*

A linguagem de programação *Instruction List*, padronizada pelo IEC 61131-3, é de baixo nível e muito similar à linguagem de programação *Assembly*, sendo esta uma

linguagem textual. Esta linguagem é complexa de se escrever, sendo normalmente usada quando existe uma necessidade de que o código tenha um alto desempenho.

O IEC 61131-3 define as seguintes operações para esta linguagem:

Instruction Operator	*Modifiers*	*Operand type*	*Description*
LD	N	Any	Load operand into accumulator (e.g. LD %IX1)
ST	N	Any	Store accumulator into operand (e.g. ST %QX2)
S		BOOL	Set operand to 1 (TRUE)
R		BOOL	Set operand to 0 (FALSE)
AND	N and ( )	BOOL	Logical AND
OR	N and ( )	BOOL	Logical OR
XOR	N and ( )	BOOL	Exclusive OR
ADD	( )	Any	Addition
SUB	( )	Any	Subtraction
MUL	( )	Any	Multiplication
DIV	( )	Any	Division
GT	( )	Any	Greater than comparison operator
GE	( )	Any	Greater than or equal to comparison operator
EQ	( )	Any	Equal to comparison operator
NE	( )	Any	Not equal to comparison operator
LE	( )	Any	Less than or equal to comparison operator
LT	( )	Any	Less than comparison operator
JMP	C,N	Label	Jump to label
CAL	C,N	Name	Call function block by name
RET	C,N		Return from function or function block

Modifiers are: N = NOT, ( ) = deferred execution and C = conditional execution. Instructions that can be used with any data type are said to be overloaded. The accumulator is a CPU data register.

Figura 25 - Operadores do Instruction List definidos pela norma IEC 61131-3. Fonte: (Crispin, 1997, p. 61)

Em muitos casos, os IDE permitem que as instruções sejam traduzir diretamente código de *Ladder Logic* em *Instruction List* e vice-versa, assim como no acontece no *EcoStruxure Machine Expert-Basic* abaixo apresentados na Figura 26 e Figura 27 (Crispin, 1997, pp. 60-61). No entanto, em alguns casos não é possível converter blocos mais complexos de *Ladder Logic* diretamente em *Instruction List*.

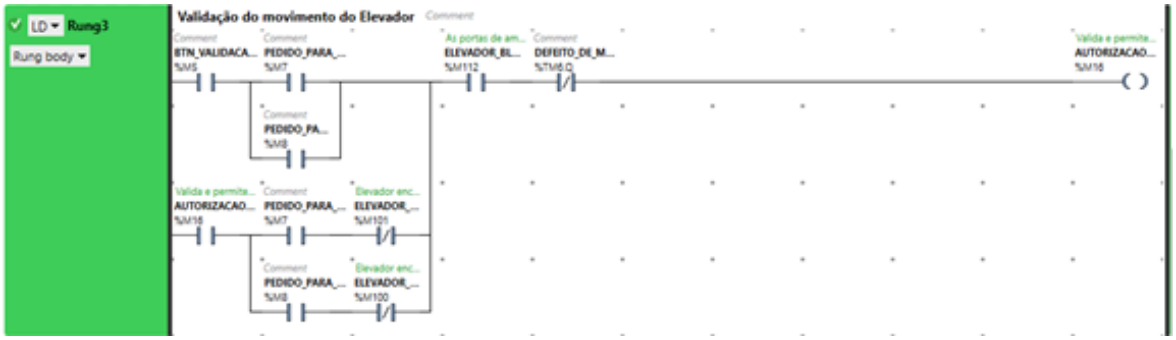


Figura 26 - Exemplo de código em Ladder Logic. Fonte: Próprio

Figura 27 - Exemplo do código da Figura 26 traduzido para Instruction List. Fonte: Próprio

Outra questão relativamente a esta linguagem, também presente nas restantes quatro com a exceção da *Ladder Logic*, é que mesmo havendo um padrão definido os fabricantes nem sempre os seguem, como apresentado na Tabela 1.

Tabela 1 - Diferenças entre operações em IL de cada fabricante. Fonte: (Wisdom Jobs India, 2011)

IEC 61131-3	Schneider	OMRON	Siemens	SAIA	Operação
LD	LD	LD	A	STH	Ler uma entrada lógica
LDN	LDN	LDNOT	NA	STL	Ler uma entrada lógica e negar
AND	AND	AND	A	ANH	Operador lógico "AND"
ANDN	ANDN	ANDNOT	NA	-	Operador lógico "NAND"
OR	OR	OR	O	ORH	Operador lógico "OR"
ORN	ORN	ORNOT	ON	-	Operador lógico "NOR"
ST	ST	OUT	=	OUT	Colocar valor lógico numa saída

2.4.3.5 – *Function Block Diagram (FBD)*

O *Function Block Diagram* é uma linguagem de programação de blocos onde os programas são expressos com a interligação de *function blocks*. Desta forma, uma analogia pode ser realizada com um diagrama de circuitos onde as conexões representam o fluxo do programa, assim como foi realizada com a *Ladder Logic*. Cada bloco é representado por um retângulo com as entradas do lado esquerdo e as saídas do lado direito, desta forma, as entradas podem apenas se ligar às saídas.

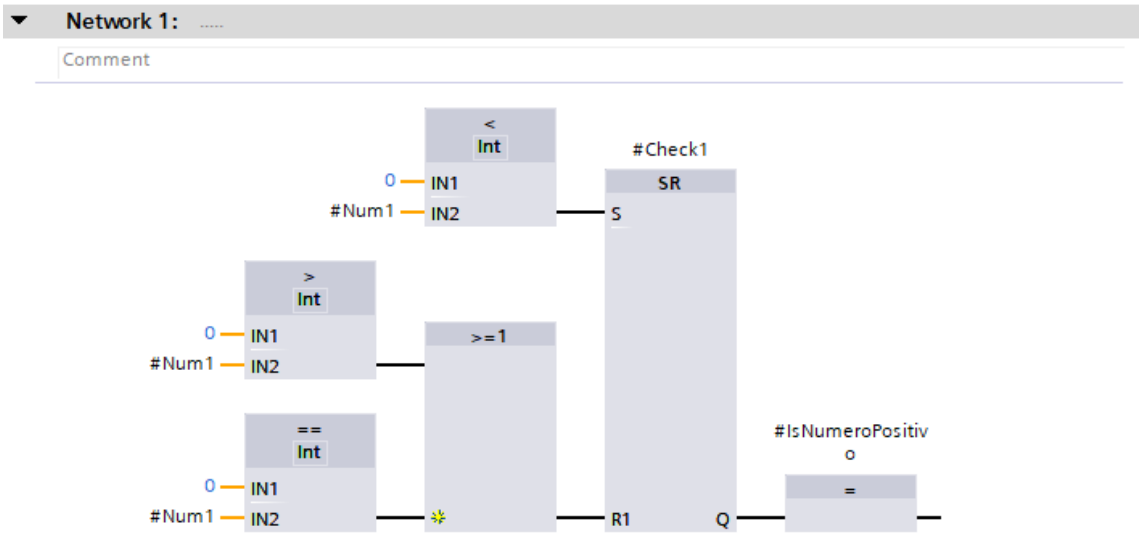


Figura 28 - Exemplo do um programa de Function Block Diagram. Fonte: Próprio

O IEC 61131-3 define poucos blocos para a livraria padrão desta linguagem, pelo que esta não é muito usada dada a incerteza das funcionalidades implementadas em cada IDE, ainda que tenha a capacidade de resolver problemas únicos. Esta não pode ser traduzida diretamente para ST uma vez que estruturas de IF/THEN/ELSE são difíceis de representar (Crispin, 1997, pp. 62-67).

2.4.3.6 – *Sequential function chart (SFC)*

A norma IEC 61131-3 padroniza uma linguagem sequencial de blocos, sendo ela referida como a *sequential function chart*. Esta linguagem é baseada no *Grafcet*[23], que foi introduzida pela norma francesa em 1977, como apresentado na Figura 22 da página 37.

A sequencia é definida como uma serie de passos que têm de ocorrer em uma determinada ordem. Cada passo é representado por um bloco retangular, que

[23] Uma linguagem gráfica para desenvolvimento de programas de controlo sequencial.

apresenta o número do passo, ligado a outro bloco retangular com a ação a ser realizada daquele passo.

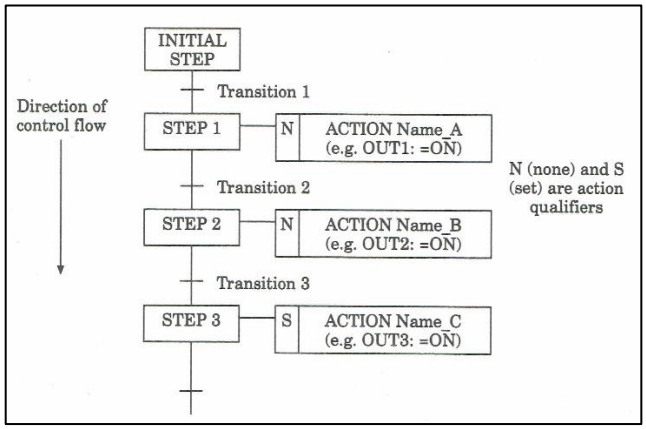


Figura 29 - Exemplo de um programa de Sequential function chart. Fonte: (Crispin, 1997, p. 65)

A união de múltiplos passos cria uma sequência que representa o programa.

2.4.4 – Marcas, modelos e versões de PLC

Em forma de conclusão deste subcapítulo, serão abordados algumas das principais Empresas a desenvolver Autómatos Programáveis e que foram usadas durante o estágio curricular. Estas são *Siemens AG*, *Allen-Bradley* e *Schneider Electric*. A Figura 30 representa outras marcas de PLC que não serão abordadas, mas que também merecem ser mencionadas pelo seu espaço no mercado.

Figura 30 - Fabricantes produtores de PLC relevantes. Fonte: (ladderlogicworld.com, 2017)

Dados obtidos do *website* Ladder Logic World (2017), apresentam *Siemens AG* como o fabricante de PLC com maior cota de mercado, seguindo-se *Allen-Bradley* em segundo lugar da lista e *Schneider Electric* em quarto lugar.

Recorrendo à ferramenta *Google Trends*, fizemos uma pesquisa relativamente ao destaque de cada empresa nos últimos 5 anos, sendo que os resultados são apresentados no gráfico seguinte.

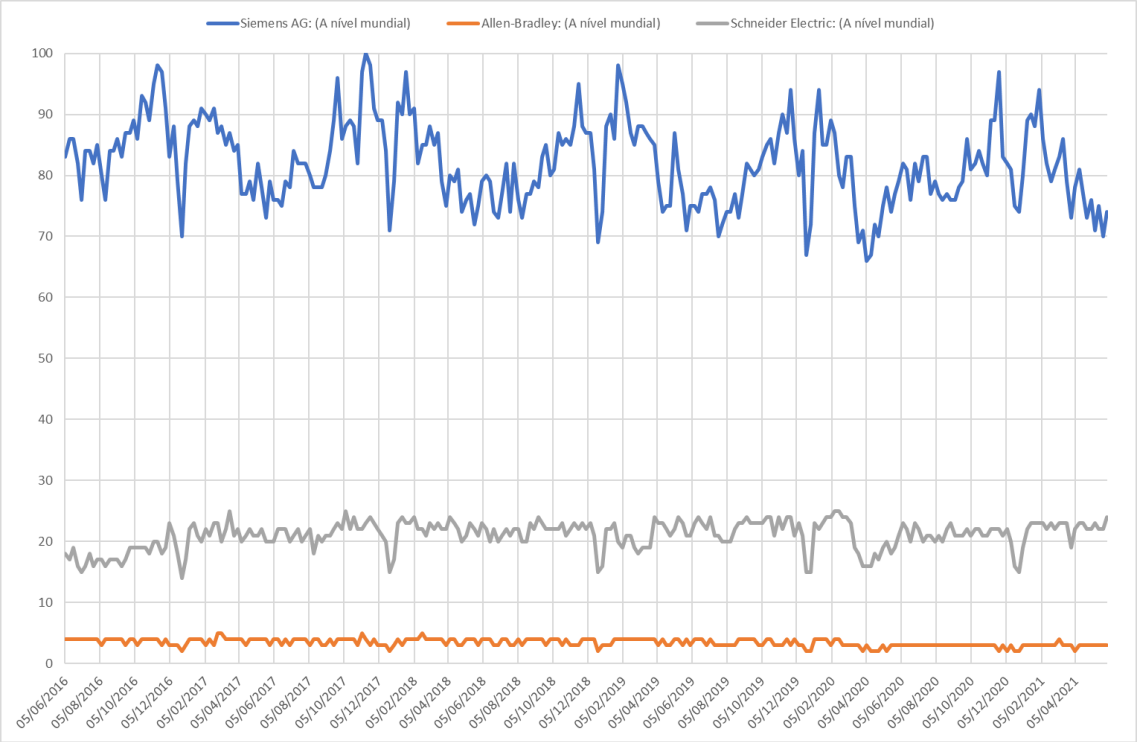


Figura 31 - Gráfico obtido a partir dos dados Google Trends relativamente à relevância de pesquisas de cada fabricante de PLC. Fonte: (Google Trends, 2021)

Analisando o gráfico, rapidamente entendemos que a *Siemens AG* é a empresa mais relevante, sendo esta mais pesquisada que as restantes empresas. Esta informação corrobora-se quando se compara com os dados do *website* Ladder Logic World (2017) apresentados anteriormente.

No entanto, este gráfico mostra uma contrariedade, uma vez que a *Schneider Electric* se apresenta mais relevante do que a *Allen-Bradley*. Esta contrariedade pode dever-se ao facto de este gráfico não representar realmente qual a marca de PLC mais usada, uma vez que estas empresas não são exclusivamente produtoras de PLC.

Pelo motivo de os fabricantes anteriormente mencionados não serem exclusivamente produtores de PLC, realizou-se uma pesquisa pelos *softwares* IDE de cada fabricante anteriormente referidos, os quais foram usados durante o decorrer do estágio curricular, de forma a tentar obter-se os dados mais corretos relativamente a qual é, ou foi, o *software*, e respetivamente o fabricante, que teve uma maior relevância nos últimos anos em termos de fabrico de PLC.

Recorremos novamente ao uso da ferramenta *Google Trends*, para pesquisar pelos IDE de cada fabricante, sendo eles:

- *Siemens AG*:
  - TIA-Portal (data de lançamento: 23 de novembro de 2010 (Siemens, 2010));
  - STEP 7-Micro/WIN V4.0 (data de lançamento: 8 de junho de 2004 (versões anteriores foram lançadas ainda mais cedo) (Siemens, 2004));
- *Allen-Bradley*:
  - RSLogix 5000 V20.x (data de lançamento: julho de 2015 (versões anteriores foram lançadas ainda mais cedo) (Rockwell Automation, 2015));
  - Studio 5000 (data de lançamento: 5 de março de 2018 (Rockwell Automation, 2018));
- *Schneider Electric*:
  - PL7 PRO V4.5 (data de lançamento: 30 de junho de 2005 (Schneider Electric, 2021));
  - EcoStruxure Machine Expert (data de lançamento: 8 de maio de 2021 (Schneider Electric, 2021));

Sendo que desta forma, obtemos o seguinte gráfico com dados desde 2004 a 05 de setembro de 2021, a nível mundial nas pesquisas do *Google*.

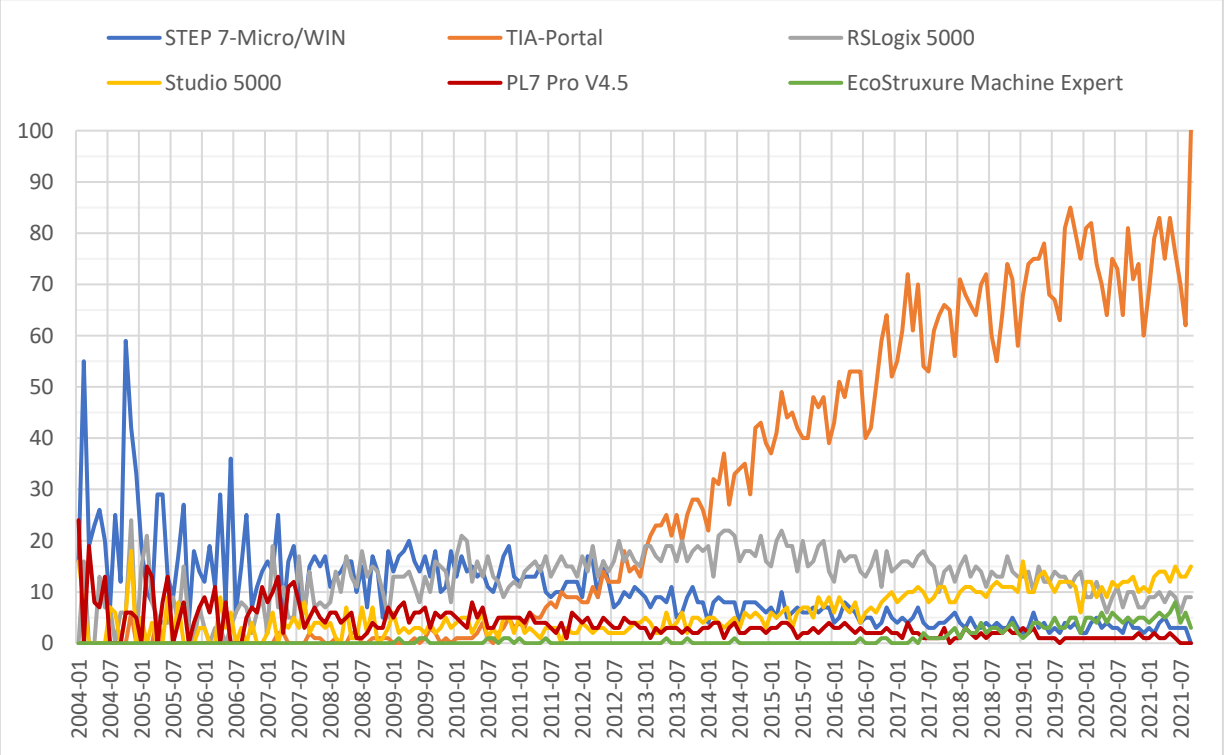


Figura 32 - Gráfico obtido através dados da ferramenta Google Trends relativamente aos IDE mais pesquisado pelo motor de busca. Fonte: (Google Trends, 2021)

De forma a facilitar a leitura da Figura 32, realizou-se um novo gráfico baseado neste, onde a cada 10 dados é obtida a sua média e usado esse valor para desenhar a nova linha gráfica. Esta simplificação dos dados, tem como objetivo a simplificação

da leitura dos dados obtidos, uma vez que os dados variam de forma mais suave. O novo gráfico obtido é apresentado na Figura 33.

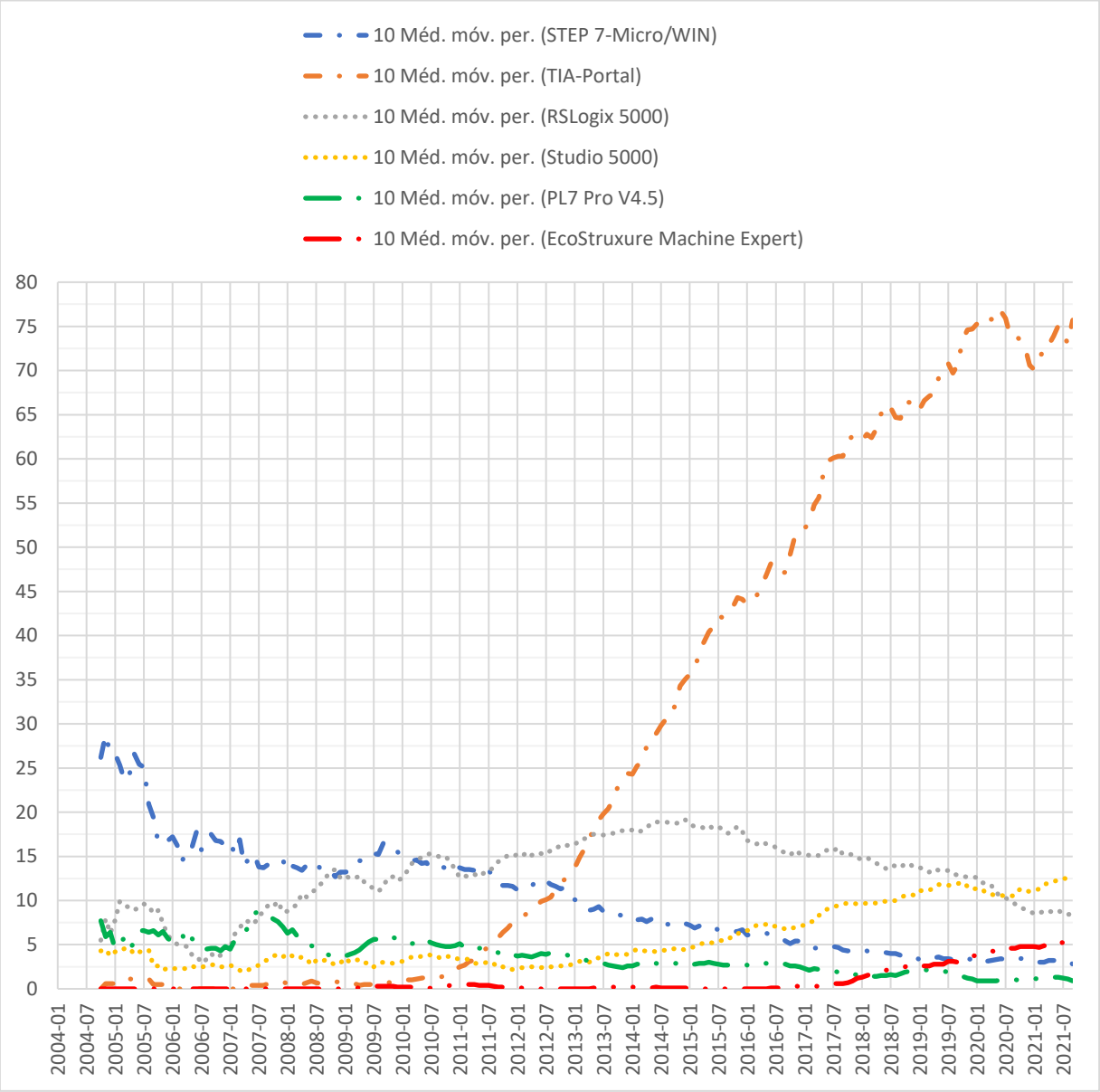


Figura 33 - Gráfico realizado com a média de 10 pontos do gráfico apresentado na Figura 32. Fonte: Próprio

Facilmente entendemos que a tecnologia da *Siemens AG* sempre teve elevada relevância no mercado, sendo que atualmente o IDE *TIA-Portal* domina o mercado.

Também é possível entender que os IDE da *Allen-Bradley* sempre foram mais pesquisados do que os da *Schneider Electric*, sendo estes dados concordantes com os apresentados anteriormente.

2.5 – Interface humano/máquina e sistemas de supervisão

Os sistemas de supervisão, conhecidos por SCADA, existem quase desde o início da automação e consistem em computadores equipados com unidades de aquisição de dados a partir dos equipamentos que se pretende monitorizar.

Durante a última década “ocorreu uma tremenda evolução dos terminais portáteis, vulgo tablets, e nas comunicações com e sem fios. Deste modo, a distinção inicial de sistemas SCADA e HMI está a diluir-se” (Pinto, 2021, p. 136). Esta diluição também se fez notar no decorrer do estágio curricular, tendo sido criada uma aplicação com o objetivo de desempenhar o papel de uma interface humano/máquina em um computador. No entanto, as atividades apresentadas neste relatório de estágio apenas usam a tecnologia HMI de determinados fabricantes, sendo que a apresentação das restantes tecnologias já sai fora do âmbito do mesmo.

2.5.1 – A evolução do HMI

O conceito HMI (interface humano/máquina) é por si só muito genérico. Qualquer interação realizada entre o humano e a máquina, ou vice-versa, ainda que seja através de um simples botão ou um *smartphone*, encaixa-se no conceito de HMI. Neste relatório de estágio serão apenas abordados as HMI construídos especificamente para comunicar com PLC, robôs e máquinas industriais.

Na época de pré-computadores as HMI (para fins industriais) eram sistemas de botões, para iniciar, parar ou pausar as máquinas, sendo esta a comunicação possível no sentido humano-máquina. No sentido máquina-humano, a comunicação possível, era realizada através de sirenes e balizas luminosas, que indicavam o estado de funcionamento da máquina ao operador (Pinto, 2021, pp. 136-137).

A partir do aparecimento do PLC e dos terminais de vídeo, as HMI passaram a ter um papel mais notório fornecendo a visualização e seguimento do processo, através da comunicação com o PLC. Além da visão e do seguimento também é possível tomar ações sobre o sistema através de botões virtuais, ou físicos no caso da HMI os possuir.

No decorrer do estágio, existiu contacto com diversas tecnologias de HMI de diferentes épocas e com diferentes capacidades. Iremos abordar dois HMI que representam duas épocas distintas e, por isso, as suas tecnologias diferem completamente, sendo possível notar claramente a evolução. Iremos abordar as tecnologias de HMI da mais antiga para a mais recente no mercado.

2.5.1.1 – *Telemecanique* XBT-E015010

A tecnologia de HMI mais antiga que existiu contacto no decorrer do estágio curricular, foi o modelo XBT-E015010 da *Telemecanique*, apresentado na Figura 34. Esta HMI é relativamente simples, sendo que o seu ecrã é apenas capaz de reproduzir caracteres, não existindo qualquer suporte para imagens ou qualquer tipo de coloração. Os caracteres têm de ser introduzidos em uma das 4 linhas disponíveis no ecrã e têm limite.

O seu modo de funcionamento é numa arquitetura de mestre/escravo, sendo ele sempre o escravo. O mestre é normalmente um PLC, com o qual este comunica por um cabo RS485 através de um protocolo de comunicação fechado.

Figura 34 - Telemecanique XBT-E015010. Fonte: (Google Imagens, 2021)

Com a utilização do seu IDE, XBT-L1000, ilustrado na Figura 35, múltiplas páginas podem ser criadas e é possível, dentro de cada página, ter valores animados que variem consoante o valor de uma *word* no PLC. Para que isto aconteça, existe uma lista de correspondências previamente criada. Quando a *word* associada à lista de correspondências obtém um novo valor e este corresponde a uma das animações pré-programadas, o valor pré-programado, ou a página, é apresentada no ecrã.

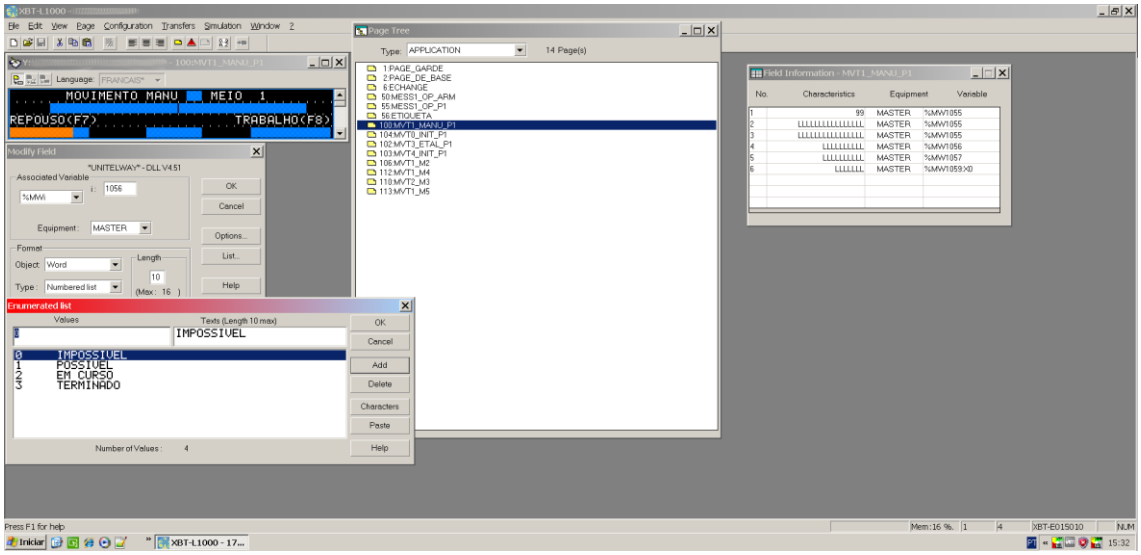


Figura 35 - Exemplo do IDE XBT-L1000 para programar o XBT-E015010. Também é possível visualizar a lista de correspondências relativamente à sua word. Fonte: Própria

Esta HMI também contem um espaço dedicado a alarmes do sistema, Figura 36, sendo que o número de alarmes definidos é limitado. Quando um bit ou múltiplos bits de uma *word* de alarme são ativos, a HMI apresenta uma mensagem pré-programada do erro.

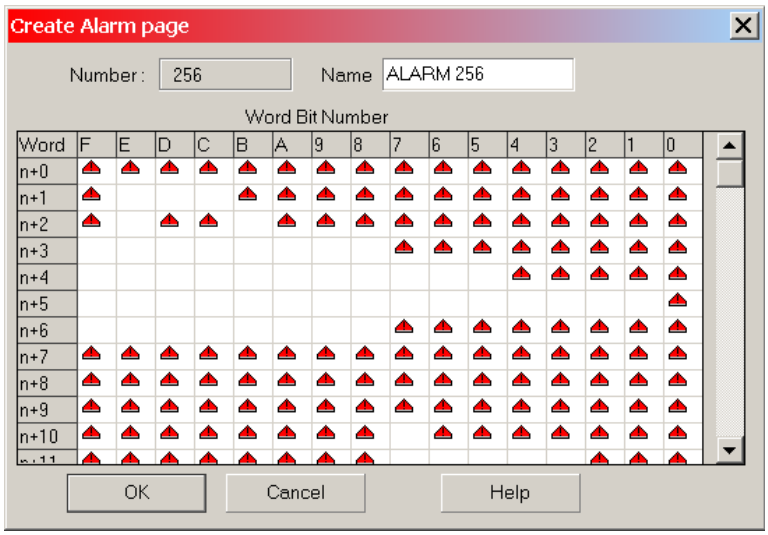


Figura 36 - Janela de alarmes, os espaços em branco são espaços livres, sendo os outros espaços ocupados por alarmes já definidos. Fonte: Própria

Relativamente à comunicação humano/máquina, esta HMI oferece múltiplos botões, sendo que tem vinte e quatro botões de funções (*F1* a *F24*) e mais um pequeno teclado. Contudo, estes botões enviam sempre a mesma informação independentemente de qual página está a ser exibida, sendo que cabe ao mestre, por norma um PLC, gerir a informação da função de cada botão relativamente a uma função global ou apenas destinado a uma determinada página. Para isto, o PLC cruza a informação de qual página está aberta, informação enviada pelo HMI e registada em uma *word* do PLC, com o botão que está a ser pressionado e verifica se existe alguma função associada a essa combinação.

2.5.1.2 – *Beijer* X2 Pro 12

Esta HMI é bem mais avançada, sendo no fundo um *tablet* com o SO *Windows CE* a correr um projeto de WPF[24]. Desta forma, o dispositivo fabricado pela *Beijer* tem a capacidade de representar milhares de cores e pixéis, posicionar múltiplos elementos em qualquer posição no ecrã sendo sempre acessíveis pelo seu ecrã tátil.

A utilização de este dispositivo permite fazer programas extremamente avançados, sendo que ao contrário da HMI anterior, este permite fazer processamento de sinais internamente e despoletar ações internas ou em variáveis externas. Este também é capaz de realizar operações complexas assim como o PLC, no entanto, este (ou HMI desta geração ou mais avançadas) não substitui um PLC, uma vez que não são desenvolvidos para ter uma resposta em tempo-real.

Figura 37 - Printscreen de um projeto tirado do iX Developer (o IDE para desenvolver projetos para as HMI da Beijer). Fonte: Própria

O que faz esta HMI tão poderosa é o facto de usar a tecnologia *WPF*, sendo que desta forma, permite desenvolver código dinâmica em cada página em *C#* e alterar componentes gráficos recorrendo ao *XAML*[25] de cada página.

Para além de possibilitar esta programação direta, ainda permite ao programador adicionar DLL[26] externos que tenha desenvolvido, possibilitando assim o utilizador usar DLL com código desenvolvido em *C++*, *C#*, *F#* ou *Visual Basic*.

---

[24] ***Windows Presentation Foundation***: é uma *framework* de UI (interface de utilizador) para criar aplicações de *desktop*;
[25] "Do inglês *eXtensible Application Markup Language*; é uma linguagem de programação baseada na XML e com conceitos inerentes ao desenvolvimento para a *framework* .NET" (Loureiro, 2017, p. 287).
[26] *Dynamic Link Library*;

No entanto, mesmo sem escrever uma única linha de código, o IDE da HMI já oferece uma zona dedicada a múltiplas funções como os representados na Figura 38.

Figura 38 - Funcionalidades disponíveis a adicionar a um projeto iX Developer. Fonte: Próprio

Além destas capacidades, esta HMI ainda tem a capacidade de fornecer serviços como de Internet, FTP, servidor OPC UA, servidor *Web* e acesso remoto.

Outra das suas grandes capacidades é a alta compatibilidade com PLC dos mais variados fabricantes, sendo que desta forma, consegue comunicar com diversos protocolos de comunicação e PLC.

Esta alta capacidade de comunicação, permite comunicar com as camadas de OT e de IT, diretamente. Esta HMI pode ser considerada um SCADA/HMI uma vez que, segundo o autor Pinto (2021), têm todas as capacidades dos sistemas SCADA, sendo eles:

- Definição de variáveis e comandos, e monitorização dos processos controlados;
- Comunicação com outros sistemas em níveis hierárquicos diferentes;
- Monitorização online de parâmetros como temperatura, pressão, etc.;
- Troca de dados bidirecional entre os sistemas de controlo e supervisão do setor de fabrico;
- Definição de parâmetros de regulação;
- Gestão de alarmes;
- Representação gráfica de variáveis do processo produtivo;
- Armazenamento de dados;
- Gestão do histórico de dados em bases de dados relacionais. (pp. 138-139)

2.6 – Robótica Industrial.

2.6.1 – Origem do termo *robot* e a sua definição

A palavra portuguesa robô, ou *robot* em inglês, provem do termo eslavo *Robota* que "significa trabalhos forçados ou escravos, e teve a sua divulgação numa peça de 1921 de Karel Čapek" (Santos V. M., 2003, pp. 1-2). Com o decorrer do tempo, o sonho passou a ser realidade e o termo robô, o escravo do homem, passou a acompanhar o Homem. Um exemplo de como a humanidade sonhava com o surgimento do robô é visível, por volta de 1950 quando Isaac Asimov definiu as Leis da Robótica:

- 1ª Lei: Um robô não pode maltratar um ser humano, ou pela sua passividade deixar que um ser humano seja maltratado;
- 2ª Lei: Um robô deve obedecer às ordens dadas por um ser humano, exceto se entrarem em conflito com a 1ª lei;
- 3ª Lei: Um robô deve proteger a sua própria existência desde que essa proteção não entre em conflito com a 1ª ou 2ª lei. (Santos V. M., 2003, pp. 1-2)

A ISO[27] 8373:2012 define que para se ser considerado um robô industrial, este tem de ter controlo automático, ser reprogramável, manipulador de multifunções e programável em três ou mais eixos, que podem ser fixo no local ou móvel para utilização em aplicações de automação industrial (International Organization for Standardization, 2012). Com esta definição, é possível traçar uma linha que divide o que é um robô industrial e do que não o é.

2.6.2 – Crescimento da robótica industrial

A robótica industrial começou no século XX, sendo que dados de 1998 incluídos em acetatos do autor Abreu (2001, p. 4), revelam que os Estados Unidos da América continham um total de 90 000 robôs instalados e a Alemanha ≈ 85 000. O mesmo autor refere que, estes números representam em termos de valores de mercado 2.2 mil

[27] *International Organization for Standardization* (Organização Internacional para Padronização).

milhões de dólares[28], mil milhões de dólares[28] e 600 milhões de dólares[28], no Japão, Estados Unidos da América e Alemanha, respetivamente. Dados mais atuais, afirmam que, mesmo com a pandemia que decorre aquando da data de publicação deste relatório, de 2019 para 2020 a robótica industrial aumentou em valor no mercado 45 mil milhões de dólares[28] nos Estados Unidos da América e prevê-se que para 2030 haja um aumento de 115 mil milhões de dólares[28] (Inteligencia Artificial, 2021).

Atualmente existem múltiplos fabricantes de robôs, sendo eles apresentados na Figura 39 consoante o número de robôs instalados.

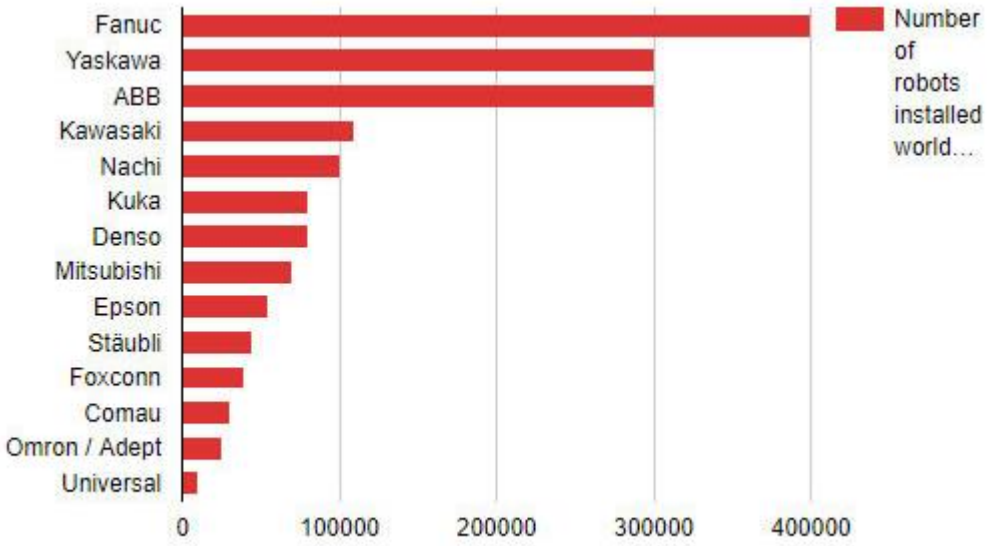


Figura 39 - Número de robôs instalados de cada fabricante. Fonte: (Dias, 2017)

Pela análise da Figura 39 compreende-se que desde os dados de 1998 muitos robôs foram instalados. Este crescimento na área da robótica, deve-se ao facto de que estes têm uma ampla e diversificada área de aplicação no mundo industrial, desde manipuladores dedicados à soldadura por arco, à pintura, à paletização, etc. (Abreu, 2001). Na verdade, o número de robôs fabricados entre o ano de 2010 e 2020 mais do que dobrou e ficou muito próximo de ter triplicado o seu valor como demonstrado na Figura 40 (International Federation of Robotics, 2021, p. 8).

[28] Este valor é relativo a dólares dos Estados Unidos da América e referente ao valor desta moeda na época a que o autor apresentou os dados.

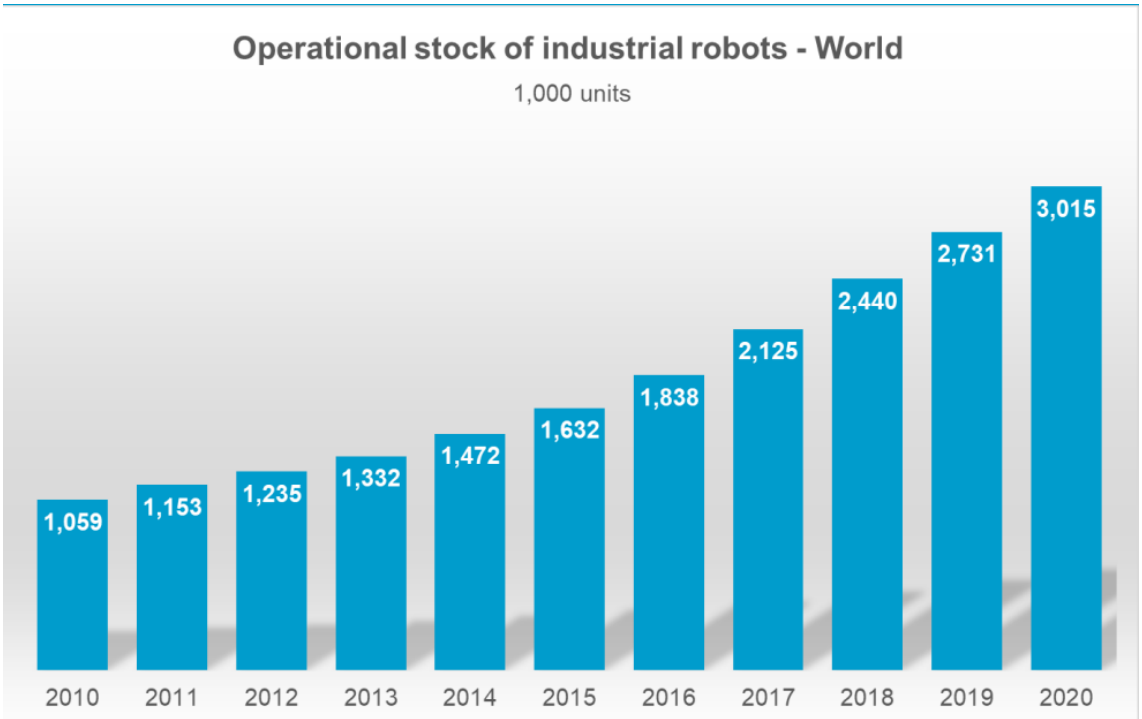


Figura 40 - Stock operacional de robôs industriais em todo o mundo. Fonte: (International Federation of Robotics, 2021, p. 8)

Outro aspeto importante é o crescimento da robótica industrial por regiões, como apresentado na Figura 41. A IFR (2021, p. 13) afirma que há um crescimento na Ásia, enquanto no continente americano e europeu estão em declínio.

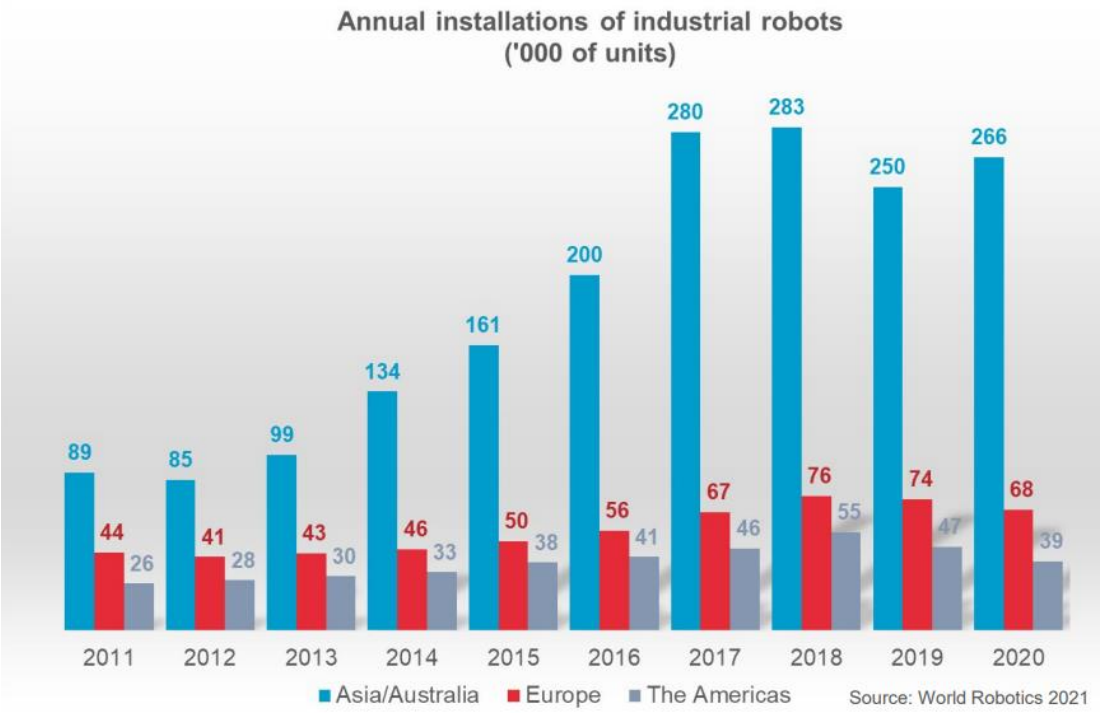


Figura 41 - Instalação anual de robôs industriais por regiões. Fonte: (International Federation of Robotics, 2021, p. 13)

Quanto ao tipo de indústria que mais tem instalado robôs, a IFR (2021, p. 10) afirma que a indústria eletrónica é atualmente a maior consumidora de robôs

industriais, ultrapassando assim a indústria automóvel, que se encontra em declínio como se pode visualizar na Figura 42.

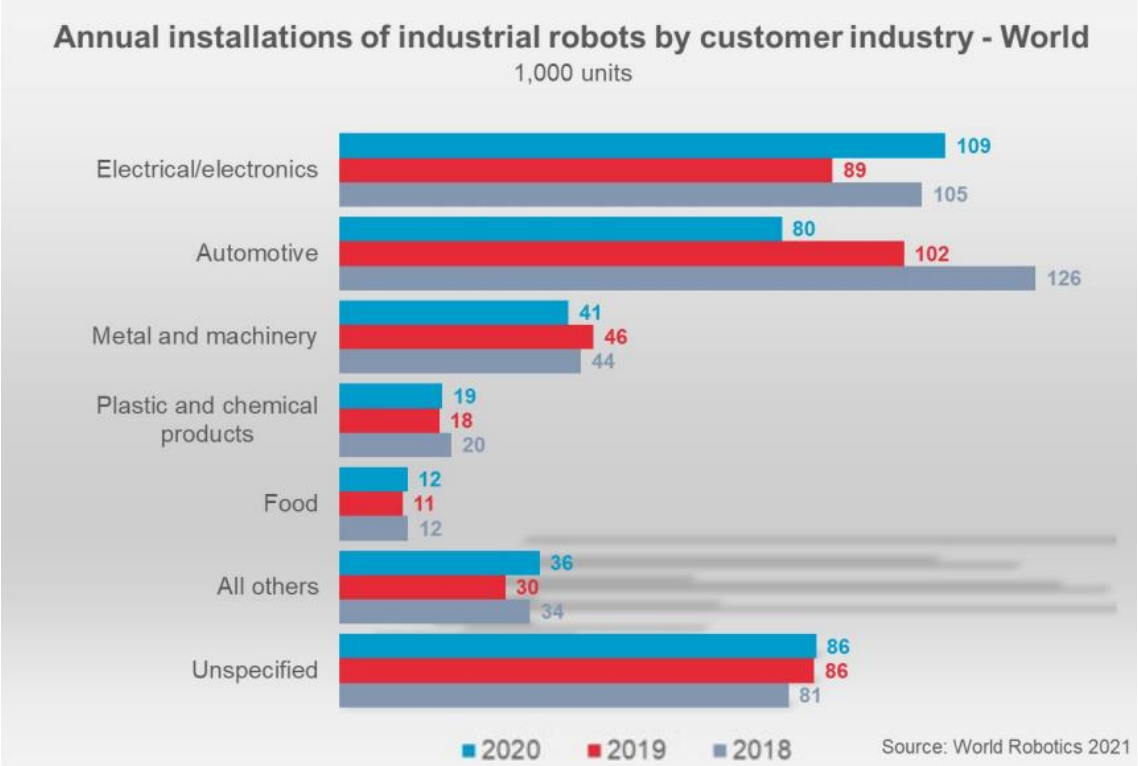


Figura 42 - Gráfico da instalação anual de robôs pelo tipo de indústria. Fonte: (International Federation of Robotics, 2021, p. 10)

Por fim, o IFR (2021, p. 12) afirma, que a robótica colaborativa tem apresentado um crescimento de 6% relativamente à robótica industrial, como se pode verificar na Figura 43.

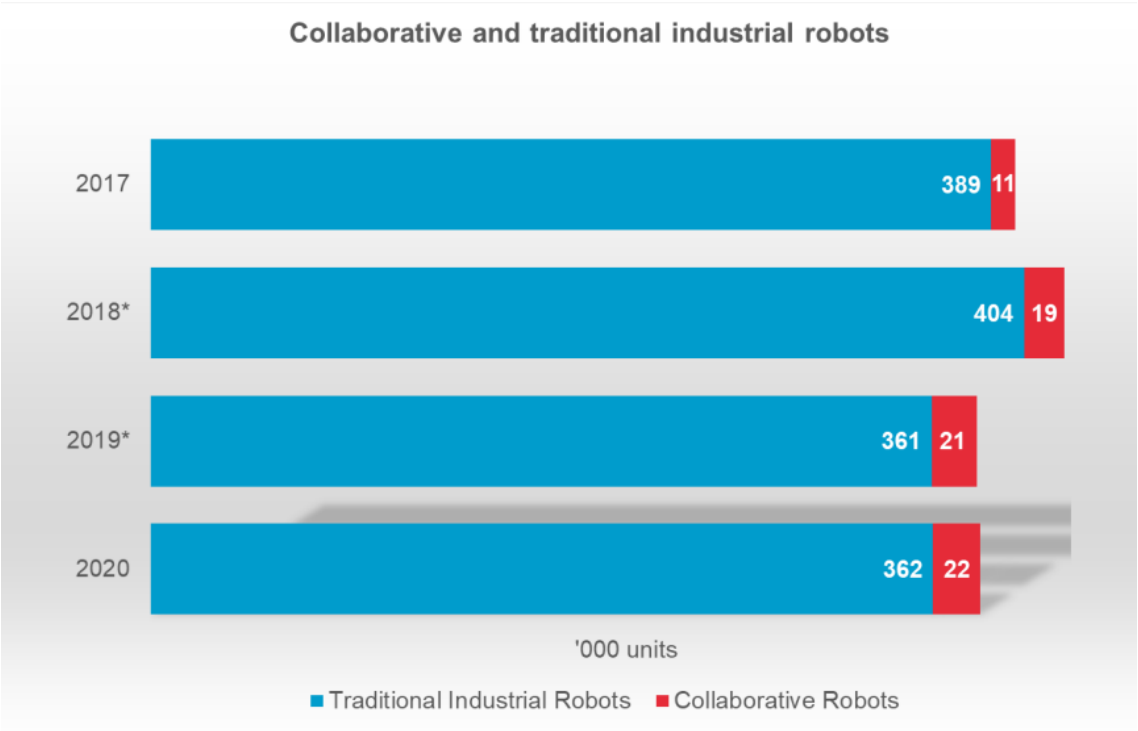


Figura 43 - Comparação entre o número de robôs industriais e colaborativa. Fonte: (International Federation of Robotics, 2021, p. 12)

Mas porque a robótica industrial e colaborativa está tão emergente e cada vez mais presente na indústria?

A mesma frase usada pelo autor Pinto (2021, p. 3) aplicada à automação industrial, pode ser também aplicada à robótica industrial com uma ligeira mudança, sendo que, a robótica industrial "passou a ser um imperativo sem o qual nenhuma *«grande»* empresa será capaz de sobreviver no mercado" atual.[29]

Na verdade, os robôs são uma necessidade das grandes empresas (e pequenas e médias) para que estas se mantenham competitivas no mercado. "Aplica-se a regra '*you'll live well if you manufacture well*', ou seja, 'viverá bem se produzir bem'" (Pires, 2018, p. 1).

2.6.3 – Elementos constituintes de um robô industrial

Os robôs industriais são constituídos essencialmente por quatro elementos distintos. Estes são os brações e punhos, os sensores, os atuadores e o controlador. Com estes pode-se fazer uma analogia com o corpo humano, sendo estes o esqueleto, os nervos, os músculos e o cérebro, respetivamente. De facto, os robôs

[29] A palavra que se encontra dentro de « », não pertence à citação original, tendo sido adicionada para dar mais sentido à afirmação neste contexto.

mais comuns, podem ser comparados diretamente ao braço humano como na Figura 44.

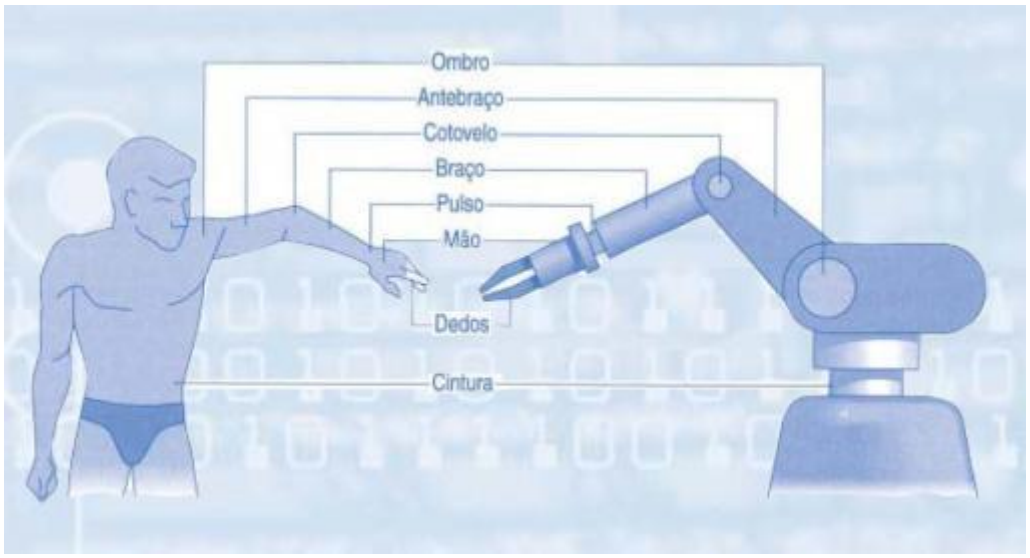


Figura 44 - Analogia entre o braço humano e o robô articulado. Fonte: (Classificação dos robôs, 2011)

2.6.3.1 – Punhos e braços

Os braços e punhos, partes mecânicas rígidas constituídas por elos e juntas, em que o conjunto definem a posição cartesianas (x, y, z, a, b, c) do robô. “O braço é a parte do manipulador que está normalmente associada ao posicionamento (x, y, z) no espaço físico cartesiano, o operacional. O punho afeta essencialmente a orientação (a, b, c) da garra, pinça ou outros *end-effector*” (Santos V. M., 2003, pp. 1-4). No entanto, é normal que haja efeitos cruzados, sendo que o conjunto do braço – punho se afetarem mutuamente na orientação e a posição cartesiana, respetivamente.

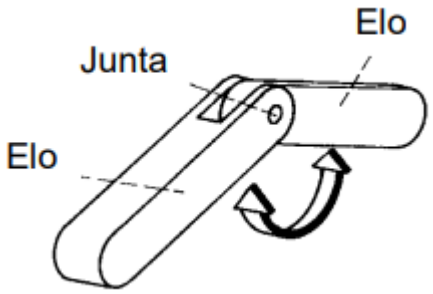


Figura 45 - Exemplo esquemático de elos e junta de um manipulador. Fonte: (Santos V. M., 2003, pp. 1-4)

2.6.3.2 – Sensores

Os sensores são os responsáveis de obter informações do mundo externo e convertê-las de forma que seja possível ao controlador as compreender. Por este motivo, os robôs vêm equipados com diferentes sensores de série, variando consoante o tipo de robô, o modelo, a aplicação e o fabricante.

No entanto, existem sensores que são comuns a todos os robôs, como os *encoders* absolutos (Figura 18, página 24), em cada um dos seus punhos (eixos). Como abordado anteriormente, os *encoders* absolutos permitem ao controlador do robô medir qual o seu ângulo, velocidade e posição, estes são elementos-chave para o cálculo da cinemática do robô.

Existem ainda entradas disponíveis para se conectar sensores externos ao controlador do robô, permitindo que estes informem do estado de algo externo ao sistema robótico, como por exemplo, o estado da sua garra (aberta ou fechada).

2.6.3.3 – Atuadores

Os atuadores também estão presentes nos punhos dos robôs e como referido anteriormente, estes representam os músculos dos robôs. Ao contrário dos sensores, os atuadores transformam a informação digital do controlador em uma atuação no mundo real. Tal como os sensores, os atuadores presentes em cada robô variam consoante o tipo de robô, o modelo, a aplicação e o fabricante, sendo possível usar atuadores elétricos, pneumáticos e hidráulicos. Na seguinte tabela, Santos (2003, pp. 2-3) apresenta as características de cada tipo de atuador.

Tabela 2 - Diferenças entre diferentes fontes de alimentação dos atuadores de um robô industrial. Fonte: (Santos V. M., 2003, pp. 2-3)

Característica	**Tipos de actuadores**		
	Eléctricos	***Hidráulicos***	***Pneumáticos***
Controlo	Fácil. Possibilidade de ser elaborado.	Hoje em dia mais facilitado com as electro servo-válvulas	Muito difícil devido a questões de compressibilidade do ar
Velocidades	Grande	Média/Grande	Muito grande
Binário a baixa velocidade (acelerações)	Pequenos/Médios	Grande	Pequenos
Precisão	Boa. Limitada pelo uso de transmissão	Boa	Má, excepto em operações a posições fixas.
Funcionamento em situação estática	Mau. Requer travões.	Excelente. Trata-se de funcionamento normal.	Bom. Não há risco de danificação do sistema.
Questões ambientais	A presença de arcos eléctricos pode ser indesejável.	Perigo de fugas de óleo.	Sistemas limpos. Risco de poluição sonora de componentes, compressores e das fugas.
Custos	Relativamente baixos	Altos	Relativamente baixos

2.6.3.4 – Controlador

O controlador é a unidade capaz de interpretar informação de sensores, assim como capaz de gerar informação para ativar os atuadores do robô. Em alguns casos, não existe uma separação entre a unidade de potência do robô, sendo que esta fica acoplada no controlador, fornecendo assim energia aos restantes componentes do robô.

Figura 46 - Exemplos de diferentes controladores de robôs KUKA. Fonte: (KUKA AG, 2017)

Atualmente muitos destes controladores são na verdade apenas uma 'máscara', sendo que apenas consistem em computadores com um software especial para o robô. Um exemplo desta 'máscara' é o *KUKA LBR iiwa* sendo que na camada mais baixa de controlo, corre o sistema operativo *Windows* que hospeda um *RTOS*[30] que gere as ações de baixo nível do robô, contudo o utilizador normal apenas tem acesso a uma camada de mais alto nível que é fornecido por um software da *KUKA* de forma que abstraia o utilizador do sistema *Windows* e este consiga manipular o robô com maior facilidade. Na verdade, a abstração do sistema operativo *Windows* é tão elevada que o utilizar em nenhum momento interage com este diretamente.

2.6.4 – Diferentes tipos de robôs

Em 2004 a IFR decidiu que os robôs deviam ser classificados quando à sua construção mecânica (International Federation of Robotics, 2021). Desta forma, os robôs podem ser qualificados como:

- Robôs articulados;
- Robôs cartesianos;
- Robôs cilíndricos;
- Robôs SCARA;
- Robôs delta.

Os robôs articulados são o mais comum de ser usado (Figura 47). Este tem um elevado número de graus de liberdade, sendo o seu desenvolvimento baseado no

[30] Sistema operativo de tempo real (*Real Time Operating System*).

braço humano. Esta simulação do braço humano traz a estes robôs a capacidade de alinhar com múltiplos planos, flexibilidade e velocidade na execução. As desvantagens são a cinemática complicada, assim como a programação e a necessidade de um controlador dedicado a este (Technavio, 2018).

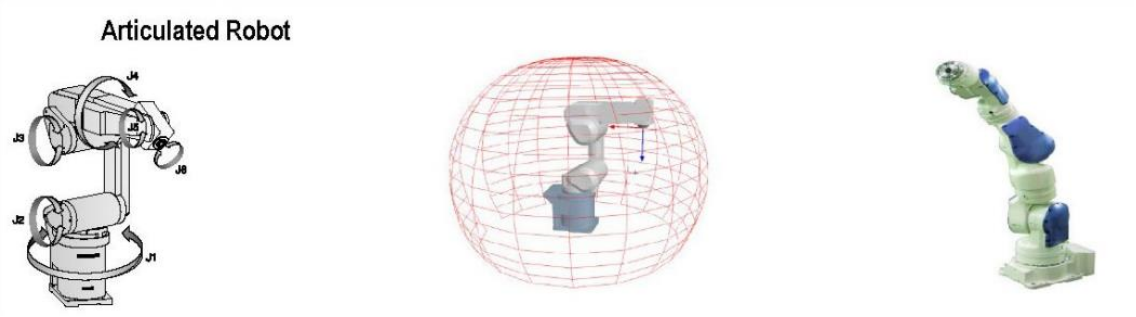


Figura 47 - Robô articulado. Fonte: (International Federation of Robotics, 2021, p. 17)

Os robôs cartesianos, apresentado na Figura 48, são robôs que se distinguem pelo seu movimento, uma vez que este não permite mexer múltiplos eixos em simultâneo, por este motivo os seus movimentos são sempre retilíneos. São geralmente usados para tarefas de *pick and place*, devido à sua alta precisão, facilidade de programação, ser capaz de se deslocar ao longo de grandes superfícies (sendo comum que um dos seus eixos tenha um alcance de 10 ou mais metros) e com capacidade de carga muito elevada (Technavio, 2018).

Figura 48 - Robô cartesiano. Fonte: (International Federation of Robotics, 2021, p. 17)

Utilizado em funções simples, como forjamento, fundição, carga/descarga, etc., o robô cilíndrico, apresentado na Figura 49, é constituído por uma articulação rotativa na sua base, uma articulação prismática e um braço extensível, sendo capaz de se mover na horizontal e na vertical. Este consegue fazer movimentos lineares, na horizontal e na vertical, e rotativos com a sua base. São de simples instalação, simples programação e necessitam de pouco espaço como os robôs articulados (Technavio, 2018).

Figura 49 - Robô cilíndrico. Fonte: (International Federation of Robotics, 2021, p. 17)

Os robôs SCARA (*Selective Compliance Assembly Robot Arm*) são caracterizados pela sua repetibilidade, velocidade e capacidade de abranger grandes áreas de trabalho. Os eixos destes robôs são na vertical com o seu braço na horizontal. Esta arquitetura mecânica permite ao robô ter um alcance no formato de um *donut*, como ilustrado na Figura 50. Desta forma, estes permitem uma grande liberdade de movimento sobre uma superfície plana (Fanuc, 2021).

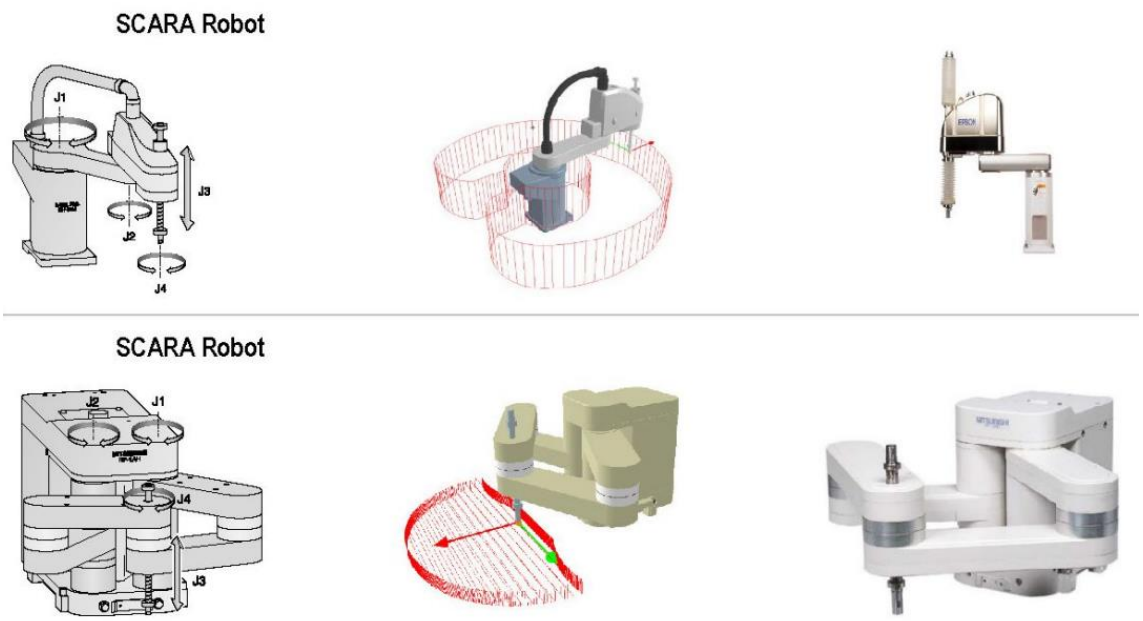


Figura 50 - Robô SCARA. Fonte: (International Federation of Robotics, 2021, p. 17)

Os robôs delta, também chamados de robôs paralelos, têm uma morfologia tipicamente aracnoide, tendo vários elementos de ligação agregados a uma placa principal (Figura 51). Estes são geralmente usados em processos de *pick and place* que necessitam de uma velocidade muito elevada e alta precisão. Porém, este tipo de robô possui uma operação complexa, requerendo um controlador dedicado a este tipo de programação (Technavio, 2018).

Figura 51 - Robô Delta. Fonte: (International Federation of Robotics, 2021, p. 17)

2.6.5 – Robótica industrial *versus* robótica colaborativa

Com os sistemas robóticos cada vez mais presentes na indústria, foi necessário definir as regras de segurança funcional para a utilização dos mesmos. Por este motivo, Platbrood & Görnemann (2018, p. 3) afirmam que "para proteger as pessoas dos perigos resultantes da velocidade, da mobilidade e da força de robôs, estes executam normalmente os seus trabalhos por trás de uma vedação".

Estas vedações são obrigatórias a todos os robôs industriais de forma a garantir a segurança de quem estiver por perto, como descrito pela ISO 10218: Robôs e dispositivos robóticos – requisitos de segurança para robôs industriais – Parte 1: Robôs/ Parte 2: Sistemas de robôs e integração. No entanto,

> Quando, no setor industrial, as capacidades do homem são combinadas com as dos robôs, obtêm-se soluções de produção que se destacam, entre outras coisas, pela mais elevada qualidade, baixos custos, melhor ergonomia e ciclos de trabalho mais rápidos (palavra-chave Indústria 4.0). (…) No entanto, quando se pretende uma estreita interação entre homem e máquina, este método padronizado e eficaz de separar fisicamente a fonte de perigo da pessoa ameaçada não pode ser aplicado. Por isso,
>
> devem ser utilizadas medidas alternativas para reduzir o risco. (Platbrood & Görnemann, 2018, p. 3)

As medidas alternativas que surgiram para diminuir este risco são os designados robôs colaborativos. Estes foram especialmente desenvolvidos para trabalhar em colaboração com o Homem e garantir a segurança do mesmo.

Os robôs colaborativos também têm as suas normas de segurança, ISO/TS 15066, com o objetivo de se desenvolver sistemas robóticos colaborativos mais seguros (International Organization for Standardization, 2016). No entanto, é necessário compreender o que define o perigo junto de um robô.

A ISO/TS 15066 define que os perigos de um robô para um ser humano classificam-se consoante o espaço e o tempo. Se o robô e os seres humanos não compartilharem o mesmo espaço em nenhum momento (tempo), representa uma situação qualificada como 'não interativa' e se existirem tarefas em simultâneo, mas não existir um espaço partilhado é uma interação coexistente. Se existir um espaço de trabalho em comum, mas que não ocorre em simultâneo, esta é qualificada como cooperativa. Se existir um espaço comum e um trabalho realizado em simultâneo esta é definida como uma 'interação colaborativa', como pode ser visualizado na Tabela 3, enquanto a Figura 52 apresenta uma interação coexistente ou uma 'não interação', a Figura 53 um interação cooperativa e a Figura 54 uma interação colaborativa.

Tabela 3 - Qualificação do tipo de interação entre o robô e os humanos. Fonte: (Platbrood & Görnemann, 2018, p. 3)

Aplicação	Espaço de trabalho diferente	Espaço de trabalho comum
Processamento sequencial	(sem interação)	Cooperação
Processamento simultâneo	Coexistência	Colaboração

Figura 52 - Exemplo de uma interação coexistente ou de não interação. Fonte: (Platbrood & Görnemann, 2018, p. 4)

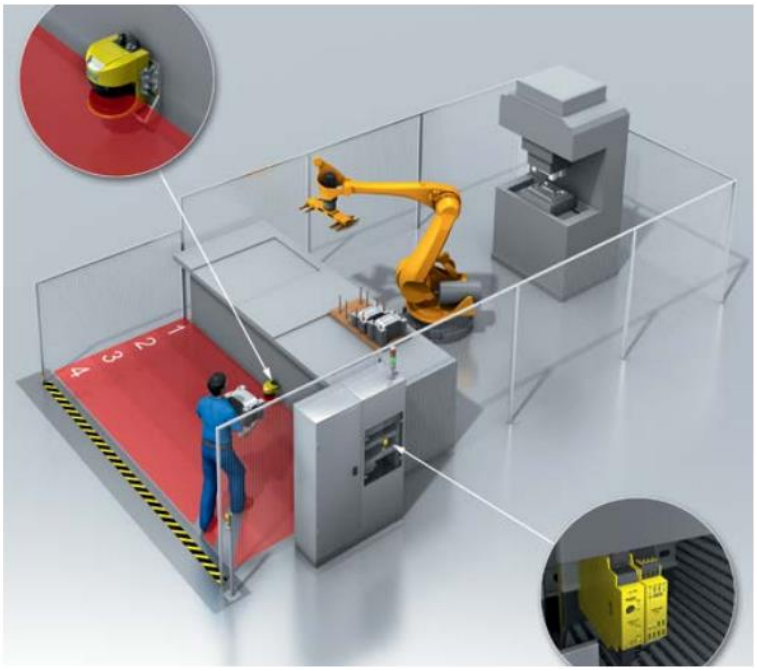


Figura 53 - Exemplo de uma interação cooperativa. Fonte: (Platbrood & Görnemann, 2018, p. 4)

Figura 54 - Exemplo de uma interação colaborativa. Fonte: (Platbrood & Görnemann, 2018, p. 5)

De seguida, Platbrood & Görnemann (2018, p. 5) citam os requisitos básicos para uma aplicação colaborativa:

- O espaço de colaboração deve satisfazer o seguinte:

1. Deve ser concebido de modo que o operador possa executar sem quaisquer problemas e em segurança as suas tarefas,

sem incorrer em perigo devido a equipamentos adicionais ou outras máquinas na zona de trabalho.

2. Não pode haver risco de ferimentos por corte, esmagamento ou perfuração, nem outros riscos por superfícies quentes, peças sob tensão, que não podem ser minimizados pela redução da velocidade, da força ou da potência do sistema robótico. Isto também é válido naturalmente para os respetivos dispositivos de retenção (ferramentas) e peças de trabalho.

- O espaço de trabalho do robô deve prever uma distância mínima para as áreas acessíveis adjacentes, onde pode correr o risco de ser esmagado ou apertado. Se isso não for possível, devem ser utilizados dispositivos de proteção adicionais.
- Sempre que possível, deve ser prevista uma delimitação segura do eixo, de modo a limitar o número de movimentos livres do robô no espaço e para reduzir o risco de ferimento para as pessoas.

2.6.6 – Modos de funcionamento colaborativos

Segundo as ISO 10218-2 e ISO/TS 15066 são especificados quatro modos de funcionamento colaborativo de um robô, dependendo das exigências das aplicações e dos designs dos sistemas robóticos, podendo ser usados individualmente ou combinados. Estes modos de funcionamento são:

- **Paragem de segurança vigiada:** este modo permite parar o robô no espaço colaborativo durante a interação com o operador;
- **Guiamento manual:** o robô pode ser guiado de uma forma segura pelo homem;
- **Limitação da força e da potência – o caminho em direção à colaboração:** “O contato físico entre o sistema robótico (inclusive da peça de trabalho) e uma pessoa (operador) pode ocorrer intencionalmente ou inadvertidamente” (Platbrood & Görnemann, 2018, p. 6). Por este motivo é necessário monitorizar a velocidade e a força do robô, para que este não seja perigoso para os que partilham o espaço colaborativo com o robô. No pior caso, o de uma colisão, o sistema robótico deve parar de imediato;

- **Monitoramento da distância e da velocidade – o futuro:** consoante os elementos presentes no espaço colaborativo, o robô deve evitar estes de forma autónoma e ao mesmo tempo desempenhar a sua função, adaptando a sua velocidade e posição relativamente à posição do operador.

CAPÍTULO 3: ATIVIDADES DESENVOLVIDAS

3.1 – Conversor de código *Kuka*

3.1.1 – Traços gerais

A pandemia provocada pelo vírus *SARS-CoV-2* obrigou ao confinamento da população em geral, o que levou ao encerramento temporário dos clientes da entidade acolhedora do estágio. Como referido anteriormente, numa fase inicial, não existiu oportunidade de realizar trabalho presencial. Esta restrição fez com que surgisse um desafio: não parar a produção, que só foi alcançável através do teletrabalho. Porém, este desafio era ainda maior sabendo que os clientes iriam necessitar de ter a produção com o máximo de rendimento e retorno no momento em que ocorresse o desconfinamento.

Por motivos de confidencialidade e proteção de dados, o cliente desta atividade não pode ser revelado. Iremos abordar o projeto deste cliente de forma geral, sendo este a reprogramação de robôs novos que iriam substituir robôs de modelos mais antigos. Os novos robôs destinam-se a realizar as tarefas que os anteriores realizavam, sendo estes idênticos em dimensão e capacidade de carga, mas com *software* e *hardware* mais recente.

Figura 55 - Consola do KUKA KR C1 e controlador e consola KUKA KR C4, da esquerda para a direita respetivamente. Fonte: (KUKA KCP KR C1 69-000-422 colgante de enseñanza KRC1 usado, s.d.; RobotWorx, s.d.)

Ainda que pareça simples, o problema surge na falta de compatibilidade das versões dos controladores dos robôs. Os robôs mais antigos, têm o controlador *KUKA KR C1*, enquanto os robôs mais recentes já trazem o controlador *KUKA KR C4*. O primeiro controlador será o ‘bisavô’ do *KR C4*, sendo que o seu ‘bisneto’ já conta com a ‘inteligência’ e evolução das três gerações anteriores. Desta forma, a sintaxe do *KR C4* não é mais compatível com a do *KR C1*, ainda que ambos partilhem a mesma

linguagem de programação, neste caso *KRL*[31]. Esta é a linguagem de programação proprietária e usada pela *KUKA* nos seus robôs industriais, existindo outras linguagens de programação proprietárias usadas por outros fabricantes, como, por exemplo, a *RAPID* usada pelos robôs industriais da *ABB*.

Foi necessário refazer o código-fonte ou a parte possível à distância, de forma que este fosse compatível com o controlador *KR C4*, admitindo que este trabalho era muito suscetível a um erro humano, uma vez que a linguagem *KRL* mudou significativamente.

Para se evidenciar as diferenças das linguagens de programação dos dois controladores, são apresentados dois códigos fontes, com algumas instruções equivalentes, mas um para o controlador *KR C1* e o outro para o *KR C4*, sendo estes apresentados no Bloco de Código 7 e no Bloco de Código 8, respetivamente. Para facilitar a leitura dos mesmos, foram adicionados comentários no início de cada expressão na forma 'INSTRUÇÃO <número da instrução>' para que fosse mais fácil identificar as expressões equivalentes.

[31] Nome da linguagem de programação da maioria dos robôs *KUKA*, incluído os controladores *KR C1*, *KR C2*, *KR C3*, *KR C4* e *KR C5*.

```
;INSTRUÇÃO 1
;FOLD ORDA.MVT 1, 0, 0, 0,   0, 0, 0, 0,   0, 1, 1, 1,   0, 0, 0, 0,
Delai=0ms;%{PE}%MKUKATPUSER
TRIGGER WHEN DISTANCE=0 DELAY=0 DO ATT_ORD ('B111000000001') PRIO=-1
;ENDFOLD

;INSTRUÇÃO 2
;FOLD EVTA.VAL 0, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0,
0;%{PE}%MKUKATPUSER
$ADVANCE=0
G_EVENT1=0
G_EVENT2=0
G_EVENT3=0
G_EVENT4=0
G_EVENT5=0
G_EVENT6=0
G_EVENT7=0
G_EVENT8=0
G_EVENT9=0
G_EVENT10=0
G_EVENT11=0
G_EVENT12=0
G_EVENT13=0
G_EVENT14=0
G_EVENT15=0
G_EVENT16=0
ATT_EVT ( )
$ADVANCE=3
;ENDFOLD
```

Bloco de Código 7 - Instruções de código em linguagem KRL para o controlador KR C1.

```
;INSTRUÇÃO 1
;FOLD Orders.PlcCont 1 TRUE, 2 ., 3 ., 4 ., 5 ., 6 ., 7 ., 8 ., 9 .,
10 TRUE, 11 TRUE, 12 TRUE, 13 ., 14 ., 15 ., 16 ., 17 ., 18 ., 19 .,
20 ., 21 ., 22 ., 23 ., 24 ., 25 ., 26 ., 27 ., 28 ., 29 ., 30 ., 31
., 32 ., DELAY=0ms%{PE}%MKUKATPUSER
RI_IlfInit()
TRIGGER WHEN DISTANCE=0 DELAY=0 DO
RI_OPTrigger('B0000000000000000','B0000111000000001') PRIO=-1
;ENDFOLD

;INSTRUÇÃO 2
;FOLD Events.PlcStop 1 ., 2 ., 3 ., 4 ., 5 ., 6 ., 7 ., 8 ., 9 ., 10
., 11 ., 12 ., 13 ., 14 ., 15 ., 16 ., 17 ., 18 ., 19 ., 20 ., 21 .,
22 ., 23 ., 24 ., 25 ., 26 ., 27 ., 28 ., 29 ., 30 ., 31 ., 32
.;%{PE}%MKUKATPUSER
RI_IlfInit()
RI_WaitForEPValid('B0000000000000000','B0000000000000000')
;ENDFOLD
```

Bloco de Código 8 - Instruções de código em linguagem KRL para o controlador KR C4.

Nos blocos de código acima apresentados, pode-se compreender que a sintaxe de cada versão é muito diferente e necessita uma 'conversão' do *KR C1* para que o mesmo código possa ser interpretado pelo controlador *KR C4*. Também é possível entender que as instruções de código destes robôs são de baixo nível, trabalhando bastante diretamente com *bits* e *bytes*.

Em modo de exemplo, o código-fonte original de um dos controladores *KR C1* necessário de converter, continha um total de 92 ficheiros de código-fonte, o que perfazia um total de 7410 linhas de código. Feitas as contas dá uma média de 113 linhas de código por ficheiro, que quando multiplicado pelos dez robôs que necessitavam de ser reprogramados, perfez um total superior a 70 mil linhas de código, mais ≈30 mil linhas de comentários que precisavam de ser revistas (não contabilizando os ficheiros do tipo **.dat*). A dificuldade é acrescida quando olhamos às instruções de baixo nível que trabalham com cadeias de *bits*. Quando calculados os números de zeros e uns, obtemos uma média de zeros e uns de 1.261 por linha de código. Estes dados foram obtidos com um código desenvolvido, o código fonte e a sua execução podem ser visualizados no

Anexo 4 - Código desenvolvido para contabilizar número de conteúdo a ser alterado.

3.1.2 – A Solução

Fazer a conversão dos códigos-fontes à mão seria muito trabalhoso e suscetível a erro humano, não sendo uma solução viável. Assim, desenvolveu-se uma aplicação capaz de fazer a conversão. A grande vantagem de se desenvolver uma

aplicação, apresentada na Figura 56, para realizar esta conversão, é que uma vez desenvolvida, esta não é suscetível ao erro humano e a sua velocidade de conversão por ficheiro é de segundos, senão de milésimos de segundos.

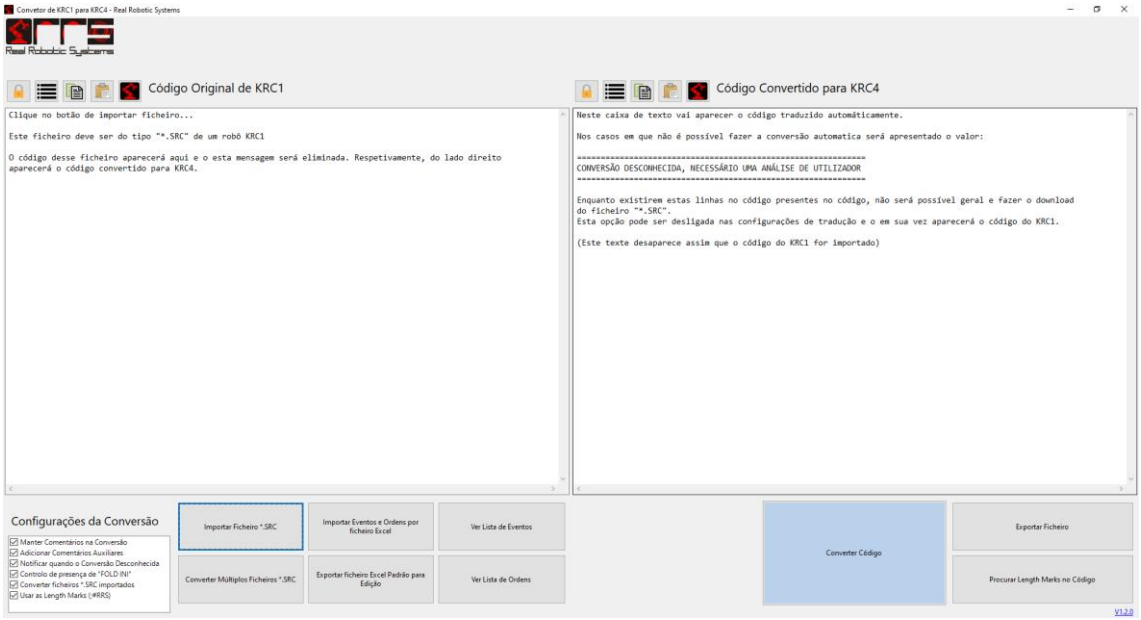


Figura 56 - Aplicação desenvolvida para a conversão dos códigos de KR C1 para KR C4. Fonte: Própria

A aplicação foi desenvolvida para ser rápida, versátil, adaptável às situações que lhe fossem apresentadas e autónoma, sendo que 95% das ações são realizadas com sucesso e sem necessidade da intervenção ou análise humana.

3.1.3 – A linguagem de programação da aplicação

Para o seu desenvolvimento, foram abordadas algumas linguagens de programação e programas. Serão aqui brevemente explicadas as opções colocadas em cima da mesa.

Inicialmente, foi colocada a hipótese de se usar o *Excel*, juntamente com *VBA*, resumidamente um ficheiro *Excel* com macros. Essas macros permitiriam ao utilizador colocar expressões em sintaxe *KRL* compatíveis com o controlador *KR C1* e devolveria um resultado equivalente em linguagem *KRL* compatíveis com o controlador *KR C4*. No entanto esta hipótese rapidamente foi abandonada uma vez que não respeita o que foi estipulado no desenvolvimento da aplicação.

Realizar a conversão, recorrendo ao ficheiro *Excel* com macros, apenas ia ajudar a diminuir o erro humano, mas não se ia ganhar uma velocidade significativa, ainda que esta já permitisse diminuir bastante o tempo, no entanto seria ainda, seguramente, necessário mais do que um dia de trabalho para realizar a conversão de todos os ficheiros. Esta solução também quebra com a ideia acima mencionada de que a aplicação seria ‘rápida, versátil, adaptável às situações que lhe fossem apresentadas e autónoma, sendo que 95% das ações são realizadas com sucesso e

sem necessidade da intervenção ou análise humana'. Usando um ficheiro *Excel* com macros, a aplicação não seria rápida olhando a todo o trabalho que seria necessário fazer e não seria 95% autónoma, uma vez que seria necessário um 'operador' que copiasse expressões entre o ficheiro *Excel* e o novo ficheiro criado para o *KR C4*.

Abandonada a ideia de usar o ficheiro *Excel*, a solução passou por ser fazer um *software* de raiz, sendo que neste ponto seria necessário escolher uma linguagem de programação para desenvolver a aplicação. As opções que foram colocadas em cima da mesa foram as linguagens de programação *JavaScript/TypeScript*, *C/C++* e *C#*.

A utilização de *JS/TS* implicaria a adição de um *runtime* externo, como o *nodeJS* (algo que não era desejado), ou desenvolver uma página *web* onde o programa ia correr. Desta forma toda a parte gráfica (que daqui em diante será abordada como GUI), seria desenvolvida em *HTML* juntamente com *CSS* e o *codebehind* seria *JS/TS*. Esta ideia era executável, mas tinha alguns contras, sendo eles a falta de conhecimento em *HTML* e *CSS* para desenvolver a *GUI* que iria levar algum tempo, o facto de que *JavaScript* não ser uma linguagem fortemente tipada o que, juntamente com a falta de conhecimento da linguagem, traria demora e transtorno e, por fim, o facto de que não era uma página *web* que se pretendia desenvolver.

Abandonada a ideia do *JS/TS*, a seguinte foi *C/C++*. A linguagem *C* foi estudada durante a licenciatura e mestrado, no entanto *C++* não o foi. Contudo, sendo *C++* apenas um *superset* de *C* esta também foi uma proposta posta em cima da mesa. Uma das vantagens da linguagem *C/C++* é o facto de se ter conhecimento sobre as mesmas. Outra das vantagens é que esta não iria correr em um ambiente de *browser*, mas sim como um aplicativo do sistema operativo. A sua forte tipagem também se apresenta como uma das grandes vantagens desta linguagem.

Contudo, existem alguns problemas que a utilização de *C/C++* traz ao processo. O primeiro deles é o facto de que *C/C++* é uma linguagem compilada, o que significa que consoante o sistema operativo com que se ia trabalhar e a arquitetura do processador, seria necessário fazer algumas modificações no código e novas compilações.

A segunda desvantagem é que é uma linguagem de baixo nível quando comparada com *JS/TS* e *C#*. Este menor nível de abstração da linguagem *C/C++*, traz alguns transtornos, como a manipulação de dados do tipo *string*, sendo possível, mas que requer algum trabalho. Da mesma forma, ler e escrever ficheiro não é feito da forma mais simples em *C/C++* o que complica e atrasa o processo, assim como também construir uma GUI com a mesma é complicado. Por fim, a linguagem *C/C++* tem a necessidade de alocamento manual de memória, o que com algum descuido não intencional, podia levar a um vazamento de memória. Por estes motivos a linguagem *C/C++* não foi utilizada.

Em suma, a linguagem utilizada foi *C#*. Esta linguagem oferece as soluções a todos os problemas que as outras apresentam, como o facto de ser uma linguagem

híbrida, o que permite compilar apenas uma vez o código e este vai correr em todos os sistemas operativos e arquiteturas dos processadores, fortemente tipada, uma boa biblioteca para manipular *strings*, assim como também ler/escrever ficheiros e o fácil desenvolvimento de uma GUI recorrendo ao *Windows Forms*.

Ainda que o *C#* não tenha sido uma linguagem de programação estudada no decorrer do percurso académico, esta é uma linguagem que vem da família *C*, sendo que partilha praticamente a mesma sintaxe. A mesma foi desenvolvida para ser um pseudocódigo da linguagem *C* sendo que o sinal # vem da ideia de esta ser 'um nível acima' da linguagem *C++*, sendo então a linguagem *C++++*, que sobrepondo todos os sinais de soma, estes formam um #.

C# inicialmente foi uma linguagem orientada a objetos, mas atualmente é multiparadigma. No entanto, ambas as linguagens anteriormente referidas também têm o paradigma da orientação a objetos, pelo que não é algo que a diferencie das outras.

Apenas para finalizar este subcapítulo, apresentamos, nas duas páginas seguintes, como ficariam as classes em *JavaScript* e *C++* do Bloco de Código 19 e Bloco de Código 24, apresentados no

Anexo 2 - Por dentro de Ladder Logic, presentes nas páginas 141 e 152, respetivamente, e implementados em *C#*.

```javascript
//JavaScript
//Criar a classe BlocoLadder do bloco de código 1
/**
 @constructor
 @abstract
 */
var BlocoLadder = function() {
    if(this.constructor === BlocoLadder)
        throw new Error("Classe abstrata não pode ser inicializada!"
                                                                    );
};

BlocoLadder.prototype.EN = false;
BlocoLadder.prototype.ENO = false;
BlocoLadder.prototype.MostraValores_Entrada = function(Variavel){
    console.log("Valor da entrada do bloco antes de ser executado
                                              bloco: " + this.EN);
    console.log("Valor da Variável associada a este bloco antes de
                              ser executado o bloco: " + Variavel);
    console.log("Valor da saida do bloco antes de ser executado o
                                             bloco: " + this.ENO);
};

BlocoLadder.prototype.MostraValores_Saida = function(Variavel){
    console.log("Valor da entrada do bloco depois de ser executado
                                            o bloco: " + this.EN);
    console.log("Valor da Variável associada a este bloco depois de
                              ser executado o bloco: " + Variavel);
    console.log("Valor da saida do bloco depois de ser executado o
                                             bloco: " + this.ENO);
};

//Criar a classe Coil do bloco de código 6
class Coil extends BlocoLadder{
    ExecutarBloco(Variavel){
        this.MostraValores_Entrada(Variavel);
        if(this.EN)
            this.ENO = Variavel = true;
        else
            this.ENO = Variavel = false;
        this.MostraValores_Saida(Variavel);
        return Variavel; //tipos básicos não é possível
                                          passá-los por referência
    };
};
```

Bloco de Código 9 - Código equivalente ao bloco de código 1 e 6, mas na linguagem de JavaScript.

```
//C++
//Criar a classe BlocoLadder do bloco de código 1
class BlocoLadder{
    protected:
        void MostraValores_Entrada(const bool& Variavel){
            std::cout << "Valor da entrada do bloco antes de ser
                              executado bloco: " << EN << std::endl;
            std::cout << "Valor da Variável associada a este bloco
                                           antes de ser executado o
                                  bloco: " << Variavel << std::endl;
            std::cout << "Valor da saida do bloco antes de ser
                               executado o bloco: " << ENO << std::endl;
        }
        void MostraValores_Saida(const bool& Variavel){
            std::cout << "Valor da entrada do bloco depois de ser
                                executado o bloco: " << EN << std::endl;
            std::cout << "Valor da Variável associada a este bloco
                                          depois de ser executado o
                                  bloco: " << Variavel << std::endl;
            std::cout << "Valor da saida do bloco depois de ser
                               executado o bloco: " << ENO << std::endl;
        }
    public:
        bool EN;
        bool ENO;
};

//Criar a classe Coil do bloco de código 6
class Coil : public BlocoLadder{
    public:
        void ExecutarBloco(bool& Variavel)
        {
            MostraValores_Entrada(Variavel);
            ENO = Variavel = EN ? true : false;
            MostraValores_Saida(Variavel);
        }
};
```

Bloco de Código 10 - Código equivalente ao bloco de código 1 e 6, mas em linguagem de C++.

Com estes blocos de código conseguimos entender o que referíamos anteriormente, principalmente o afastamento e diferença na sintaxe da linguagem de *JavaScript* e o facto de ser muito semelhante o código de *C++* e *C#*.

3.1.4 – Modo de funcionamento da conversão

3.1.4.1 – A 'pilha' de um programa

Qualquer linguagem de programação é composta por expressões, sendo que cada expressão faz ocorrer um determinado acontecimento. Estes acontecimentos podem ser de alocar memória, escrever na memória, ler da memória, mudar o estado de uma saída, etc.

Todos os programas, são constituídos por múltiplas expressões, que executadas de determinada ordem tornam um programa funcional. Para que as expressões fiquem mais organizadas, foram criadas as funções ou métodos, sendo que estes juntam umas determinadas expressões para realizarem uma determinada tarefa.

Dentro das funções podem ainda ser chamadas outras funções, haver estruturas condicionais (*IF-ELSE*) e estruturas repetitivas (*WHILE/FOR*). Em todos estes casos, existe a necessidade de se definir onde é o início e o fim, como por exemplo, é necessário definir-se onde é o início e o fim de uma função, o início de um *IF-ELSE* e o fim do mesmo, etc. Para a linguagem *KRL* da *Kuka*, esta definição é feita da seguinte forma.

```
DEF  T_PM3_16 ( )
 WHILE (E_POUT1==E_POUT2)
  IF E_POUT1 THEN
   AFFICHE (1,1,"M" )
   AFFICHE (3,1,"M" )
  ELSE
   AFFICHE (2,1,"M" )
   AFFICHE (4,1,"M" )
  ENDIF
 ENDWHILE
END
```

Bloco de Código 11 - Código em KRL com uma condição WHILE e IF-ELSE.

Na primeira linha de código do Bloco de Código 11 temos a definição de uma função. A sintaxe para definir uma função é da seguinte forma:

```
DEF <nome da função> (<nome do 1º parâmetro>: <tipo do 1º parâmetro>, …)
```

A palavra *DEF* é uma palavra-chave de *KRL* que elucida que uma função vai ser definida, de seguida indica-se o nome que se quer atribuir à função e por último, se se pretender, define-se os parâmetros da função, sendo que se começa por definir o nome do parâmetro e separado por dois pontos, o tipo do parâmetro, se se pretender, pode ser adicionado um segundo, terceiro ou mais parâmetros separando-os uns dos outros por vírgulas. A quarta linha de código, do mesmo bloco, representa

a chamada de uma outra função. Para uma função ser chamada apenas é necessário colocar o nome da função e por fim os argumentos, se existirem, a serem passados.

Na última linha de código, do bloco, podemos encontrar a palavra *END*, esta também é uma palavra-chave da linguagem *KRL* e representa que a definição daquela função acaba naquela linha. A palavra *END* + <*instrução*> também é usada para definir onde acabam o condicionamento de determinada instrução. Por exemplo, a condição de repetição da linha dois, do Bloco de Código 11, que começa com a palavra-chave *WHILE*, termina na penúltima linha com a palavra-chave *ENDWHILE*. O mesmo acontece com a instrução de estrutura condicional que começa na terceira linha com a palavra-chave *IF* e termina com a palavra-chave *ENDIF*.

Os conceitos apresentados no paragrafo anterior, são importantes, porque são uma das partes mais fundamentais para que se possa verificar se a estrutura do código está bem feita ou se existem erros. Isto porque a estrutura de uma função pode ser armazenada em uma fila do tipo *FILO*[32], a esta estrutura vamos-lhe chamar de pilha. Esta estrutura permite-nos saber o que esperar do fluxo do programa.

Olhando ao bloco de código anterior apresentado, a sua estrutura pode ser representada da seguinte forma:

1) Na primeira linha de código, entramos dentro da função T_PM3_16, sendo que adicionamos a mesma à pilha;

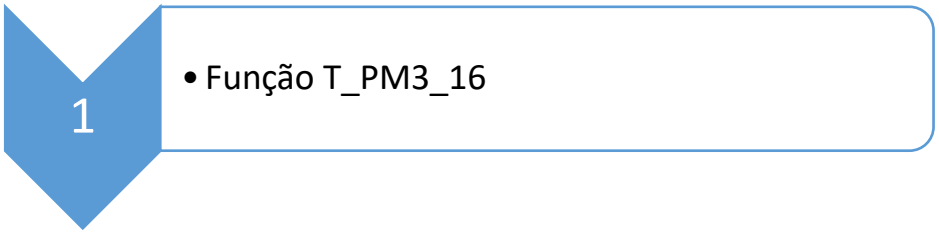


Ilustração 1 - Representação do estado da pilha.

2) Na segunda linha de código, entramos entro de um ciclo *WHILE*, este também adicionado à pilha, mas como a pilha é uma fila do tipo *FILO*, esta passa a ser o membro mais superior e a ter de se sair, se quisermos sair da função, por exemplo;

[32] *First In, Last Out.*

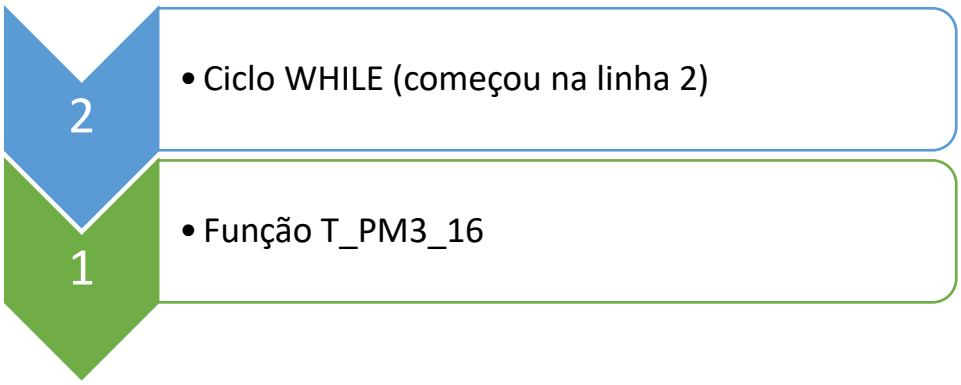


Ilustração 2 - Representação do estado da pilha.

3) Na terceira linha de código, entramos dentro de uma estrutura condicional IF, sendo que esta também é adicionada à pilha, todas as regras já referidas anteriormente continuam a ser respeitadas;

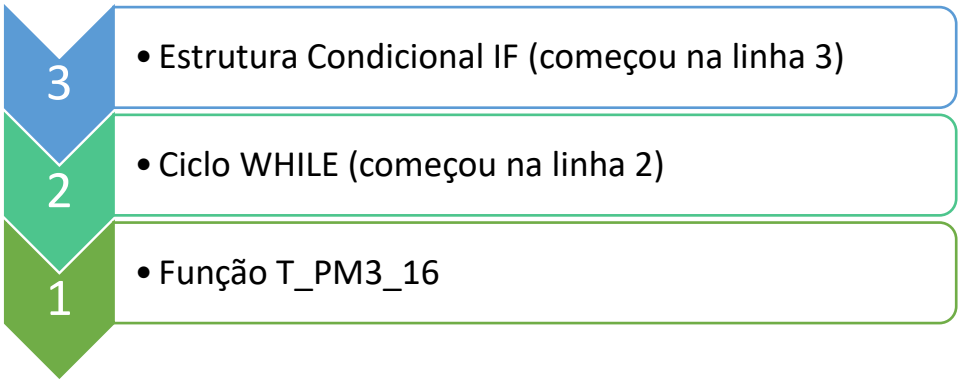


Ilustração 3 - Representação da pilha.

Neste momento, a Ilustração 3 representa o estado da pilha atual, a forma mais simples de entendermos o funcionamento da pilha, é pensar nela como uma coluna do jogo dos 4 em linha. Quando olhamos a qualquer coluna da Figura 57, facilmente entendemos que temos obrigatoriamente de retirar primeiro a peça mais superior antes de conseguirmos retirar uma peça mais inferior, esta é a perfeita representação de uma fila *FILO* e do que acontece na nossa pilha.

Figura 57 - Jogo dos 4 em linha. Fonte: (Jogo Quatro em linha tridimensional, 2021)

Tendo o último parágrafo em mente, conseguimos entender que no ponto da Ilustração 3, a instrução que esperamos para retirar um elemento da pilha (sendo o elemento superior a estrutura condicional *IF*) é um *ENDIF*. Caso encontrássemos a palavra-chave *ENDWHILE* ou *END* (para terminar o bloco de repetição ou a função, respetivamente), estas representariam um erro no código, visto que não está a respeitar as regras de funcionamento da pilha.

Neste exemplo, para esvaziar a pilha, existem três formas, duas possíveis de usar na sintaxe *KRL* e uma terceira que é comum em praticamente todas as linguagens de programação, mas não na linguagem de programação *KRL*. Estes três métodos são os seguintes:

1. Execução normal do programa, sendo que primeiro encontramos a palavra-chave *ENDIF* que nos retira o elemento mais superior da pilha, de seguida a palavra-chave *ENDWHILE* que retira o elemento seguinte da pilha e por fim a palavra-chave *END* que retira o último elemento da pilha;
2. Funcionamento normal do programa, mas usando a palavra-chave *RETURN*. Esta palavra-chave retira todos os elementos da pilha de uma função. Olhando a uma pilha local, sendo que esta pilha é criada e iniciada sempre que é chamada uma nova função, esta pilha é limpa por completo;
3. A última forma de limpar dados da pilha, relembrando que esta forma não existe na linguagem *KRL*, é com a palavra-chave *THROW* (esta palavra-chave pode ter outros nomes em algumas linguagens de programação, mas em geral é esta a sua designação). Esta palavra-chave pode eliminar por completo uma pilha e terminar com um programa, como pode apenas retirar um elemento da pilha, isto depende de como o programa está escrito. A palavra-chave *THROW* lança um erro ‘programado’, este erro lançado se não for tratado pode ser fatal para o funcionamento do programa, esvaziando a pilha por completo e terminando o programa. O tratamento de estes erros é feito pelo bloco *TRY-CATCH*, sendo que estes também são

uma estrutura mais avançada que é adicionada à pilha, este bloco pode tratar do erro impedido que ele retire mais elementos da pilha ou não, dependendo do tipo de erro que o bloco *TRY-CATCH* está programado para 'lidar'. Um exemplo da utilização da palavra-chave THROW é na 9ª linha do Bloco de Código 9.

3.1.4.2 – Declarações, inicializações e ficheiros **.DAT*

Curiosamente, a estrutura de ficheiros de um programa *KRL* pode ser comparada à de um programa da linguagem *C/C++*. Em *C*, quando se cria um novo ficheiro do tipo **.C*, também é criado, por boa prática, um ficheiro **.h*[33]. Da mesma forma, na linguagem de programação *KRL* também quando são criados ficheiros **.SRC*, o ficheiro equivalente ao **.C* na linguagem *C*, também é criado conjuntamente o ficheiro **.dat*, o equivalente ao ficheiro **.h* em *C/C++*. Este ficheiro **.dat* é onde são feitas as declarações de variáveis e as suas respetivas inicializações. A seguir apresentamos um exemplo destas declarações e inicializações, presentes em um ficheiro **.dat*, no Bloco de Código 12.

```
DEFDAT  T_PM3_16
DECL INT SUCCESS
DECL E6POS XP_ANT2={X 289.8739,Y -270.2709,Z 286.9847,A -0.993033,B
0.3619163,C -179.0779,S 6,T 18,E1 0.0,E2 0.0,E3 0.0,E4 0.0,E5 0.0,E6
0.0}
DECL FDAT FP_ANT2={TOOL_NO 4,BASE_NO 2,IPO_FRAME #BASE}
DECL LDAT LCPDAT11={VEL 2.0,ACC 100.0,APO_DIST 100.0,APO_FAC 50.0}
DECL PDAT PPDAT10={VEL 100.0,ACC 100.0,APO_DIST 100.0}
DECL GRP_TYP GGDAT4={G_APO #NO,TIME 0.2,CTRL #OFF,DLY 0,DST 1}
ENDDAT
```

Bloco de Código 12 - Exemplo de inicializações feitas no ficheiro *.dat.

Este é apenas um curto exemplo meramente ilustrativo, sendo que utilizando o mesmo programa usado anteriormente, disponível no

Anexo 4 - Código desenvolvido para contabilizar número de conteúdo a ser alterado, calculamos o número médio de linhas por ficheiro **.dat* e obtemos uma média de 63 linhas de código por ficheiro.

Analisando o Bloco de Código 12, facilmente se compreende que para declarar uma nova variável, apenas é necessário colocar a palavra-chave *DECL <Tipo de dado> <Nome da Variável> [= <Valor inicial da variável>]*. Também é possível compreender que a maioria das declarações feitas, estão relacionados com as posições do robô. Desta forma, visto que todos estes pontos tiveram de ser refeitos,

[33] Sabe-se que não é necessário criar um *header file* (*.h) sempre que se cria um ficheiro C/C++, no entanto, para este exemplo vamos assumir que sim;

pelo facto de que os novos robôs não partilharem as mesmas escalas que os anteriores, nem equivalência nas suas medições, os ficheiros **.dat* apenas foram criados. Os ficheiros **.dat* criados, mantinham-se em um estado básico e sem grandes declarações, para além do início da declaração dos dados e a sua sucessiva finalização. Outro motivo que levou a que esta parte da conversão ficasse mais básica, foi o facto de não haver mais tempo para se dedicar a este projeto.

3.1.4.3 – Desenvolvimento da parte GUI

Para o desenvolvimento da parte gráfica do software foi usado os *Windows Forms*, como já anteriormente tinha sido referido. Esta tecnologia ajudou pelo facto de ser simples de usar e interativa. As alternativas existentes eram construir uma aplicação com *XAML*, recorrendo ao *WPF* ou *Xamarin Forms*, que são desenvolvedores gráficos compatíveis com *C#* e o *Visual Studio 2019*[34], no entanto estas tecnologia não são tão interativas, sendo necessário um conhecimento mais profundo acerca das mesmas para as puder utilizar[35].

A GUI da aplicação consiste em dois campos de texto, onde à esquerda é onde se coloca o código em *KR C1* e à direita aparece o mesmo código convertido para *KR C4*, sendo que todos os restantes elementos são de controlo da aplicação. Os botões de controlo têm diversas funcionalidades sendo de seguida apresentadas as mais importantes.

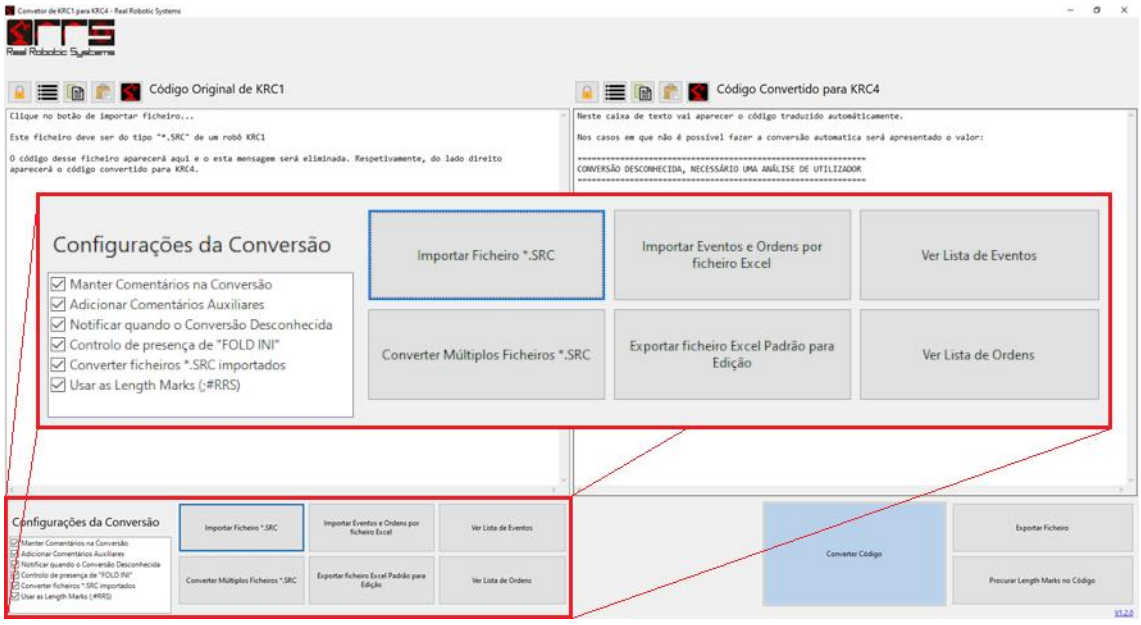


Figura 58 - Apresentação das definições possíveis de se realizar na aplicação. Fonte: Própria

As primeiras configurações possíveis da aplicação que iremos abordar aqui, encontram-se no canto inferior esquerdo, visível na Figura 58. Estas configurações

[34] IDE onde a aplicação foi desenvolvida;
[35] Caso exista curiosidade, esta tecnologia é muito parecida em sintaxe com HTML.

pertencem a um grupo com opções e *checkbox*, que permitem alterar definições nas configurações do conversor do código. Vamos abordar cada uma dessas opções e o que elas significam:

1. Manter comentários na conversão – esta opção quando ativa, mantém os comentários que existiam na versão do ficheiro do *KR C1* no ficheiro gerado do *KR C4*. Quando desativa, estes comentários não são copiados para a versão do código gerado;
2. Adicionar comentários auxiliares – o orientador da empresa pediu que esta opção fosse adicionada ao programa, iremos abordar mais à frente o que esta opção significa;
3. Notificar quando a sintaxe for desconhecida – esta opção abre uma janela auxiliar caso o conversor não consiga fazer a conversão de uma expressão. Esta janela, tem várias opções sobre como prosseguir relativamente a esta expressão, como apresentado na figura seguinte;

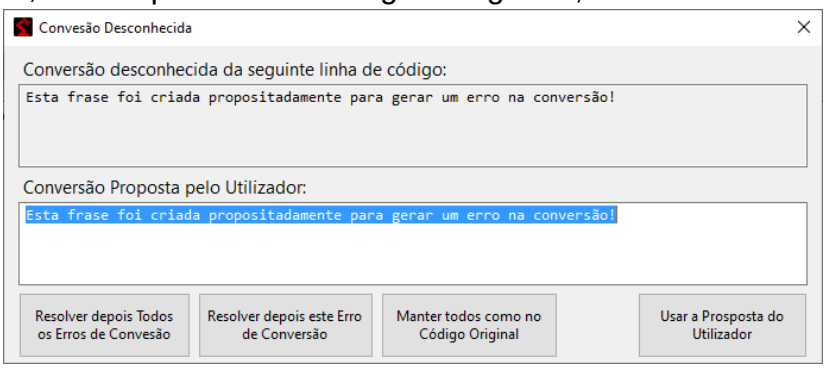


Figura 59 - Janela de "Conversão Desconhecida". Fonte: Própria

4. Controlo da presença de “FOLD INI” – esta opção também foi pedida para ser colocada, uma vez que cada ficheiro de código deste cliente contém esta pasta de código com a inicialização e definição de algumas definições do programa. Por motivos de confidencialidade não é possível explicar mais acerca desta opção, para além de que esta opção apenas habilita e desabilita o controlo da existência desta pasta;
5. Converter ficheiros **.SRC* importados – esta opção quando ativa e é importado um ficheiro de código, converte diretamente o ficheiro, caso a opção não esteja ativa, o ficheiro apenas é importado, mas não é convertido;
6. Usar *Lengths Marks* (;#RRS) – esta opção ativa umas marcações em pontos que sejam necessários ser editados.

Quantos aos botões disponíveis, cima para baixo, da direita para a esquerda, temos os seguintes botões:

- Importar Ficheiro **.SRC* – Importa um ficheiro do tipo **.SRC*. Se a opção 5 do conversor estiver ativa, o ficheiro é convertido diretamente;
- Converter Múltiplos Ficheiros **.SRC* – Abre uma nova janela que permite importar múltiplos ficheiros de uma diretória e convertê-los;
- Importar Eventos e Ordens por ficheiro *Excel* – Este botão permite importar um ficheiro *Excel* (que é exportado no botão seguinte), onde pode ser

declarado um mapa com comentários para o que cada *bit*, das ordens e dos eventos, significa. Cada vez que existe uma ordem ou um evento, se a opção 2 do conversor estiver ativa, é apresentada como comentário do código, o que utilizador escreveu no ficheiro *Excel* para aquele(s) *bit(s)*;

- Exportar Eventos e Ordens por ficheiros *Excel* – Exporta um ficheiro *Excel* com os nomes dados aos eventos e ordens atuais. Este ficheiro pode ser editado e por fim novamente importado com o botão anterior;
- Ver Lista de Eventos – Abre uma nova janela com uma tabela com o número do evento e o nome atribuído ao evento;
- Ver Lista de Ordens – Abre uma nova janela com uma tabela com o número da ordem e o nome atribuído à ordem.
- Converter Código – Este botão permite converter código, se este não tiver sido ainda convertido, ou se foi alterado manualmente no programa;
- Exportar ficheiro – Exporta um ficheiro do tipo **.SRC* com o código convertido e cria um ficheiro **.dat* com os dados anteriormente referidos;
- Procurar *Length Marks* no Código – Permite que o cursor do código convertido selecione a próxima *Length Mark* presente no código a partir do ponto onde está o cursor.

3.1.4.4 – Modo de funcionamento do programa e C#

Existem informações importantes sobre a linguagem C# de se referir em relação a este projeto. A primeira é qual versão de C# foi desenvolvida esta aplicação, sendo que a versão da linguagem de programação foi C# 8.0 e a versão do seu *Runtime* é o *.NET Core 3.*1. Esta versão do *Runtime* é, no momento em que este relatório de estágio foi redigido, uma versão que recebe um suporte de longo termo. Esta versão mais recente, também se diferencia da versão do *Runtime .NET Framework 4.8* pelo facto de agora ser compatível com outros sistemas operativos para além do *Windows*. Outra diferença importante, é que o *Runtime .NET Core 3.1*, quando exportada a aplicação, esta já contem a sua máquina virtual incluída no ficheiro, enquanto o *.NET Framework* necessitava que o utilizador tivesse a máquina virtual da respetiva versão instalada no computador[36].

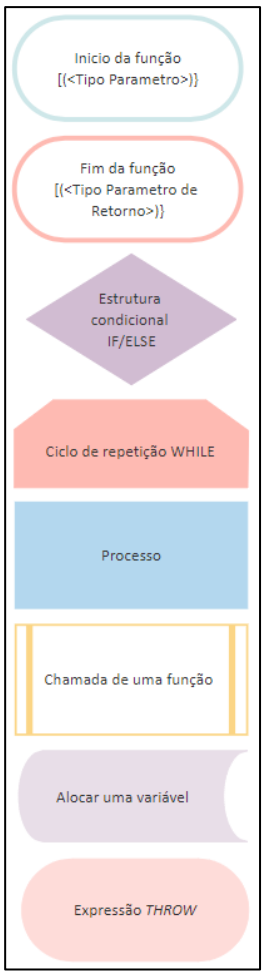
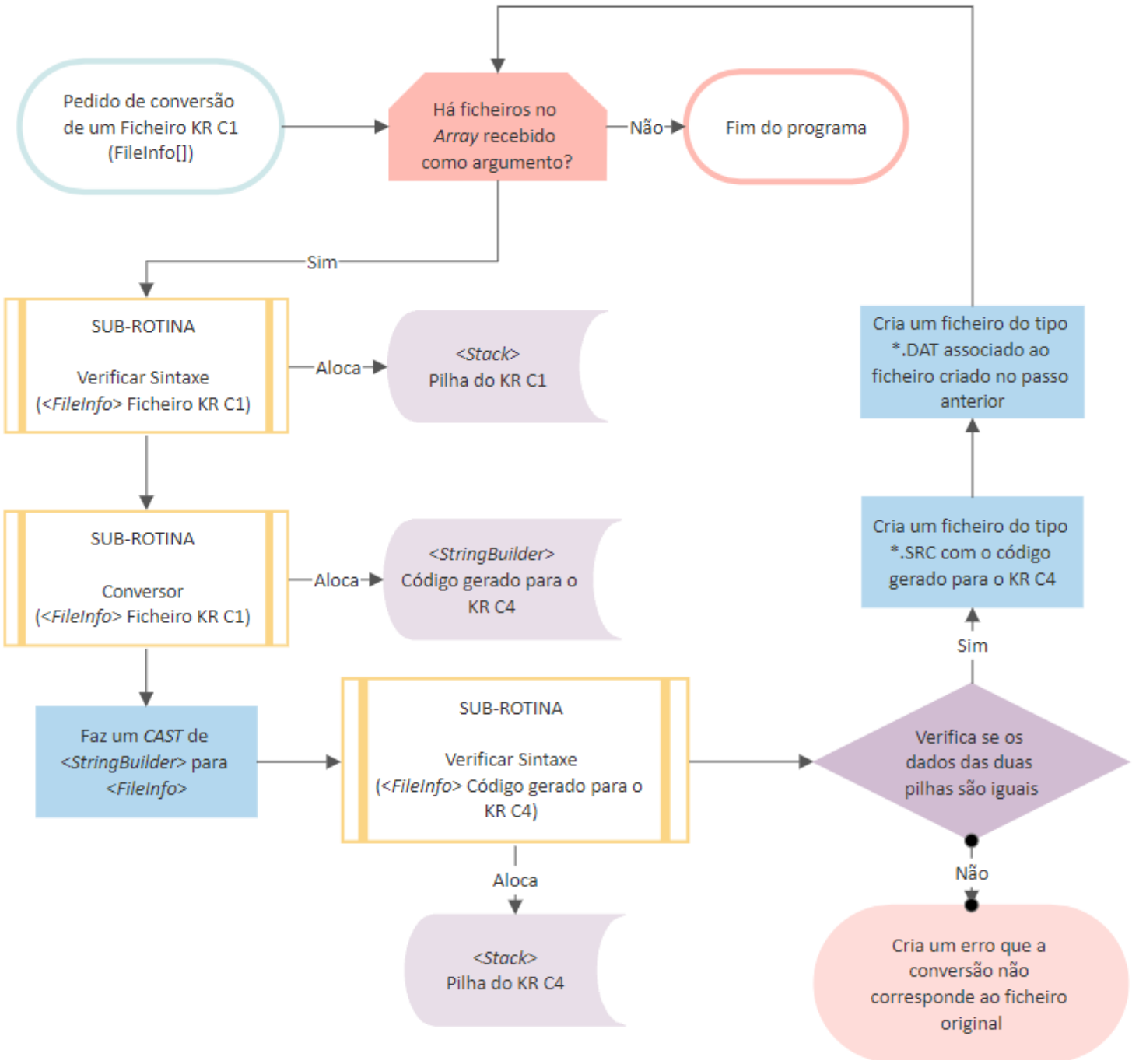


Figura 60 - Legenda dos blocos para os fluxogramas. Fonte: Própria

De forma a tornar a leitura do relatório mais fluída, vamos explicar o modo de funcionamento do conversor recorrendo a fluxogramas acompanhados com a descrição e explicação. *Nota: a Figura 60 apresenta a legenda dos blocos do fluxograma.*

No Fluxograma 1 apresenta-se como é feita a conversão de um ficheiro ou de múltiplos ficheiros, a partir do momento em que existe um pedido de conversão. Quando há um pedido de conversão, este recebe como argumento um *array* do tipo *FileInfo*. O tipo *FileInfo*, é uma classe pertencente à livraria padrão do *C#* (pertencentes ao *namespace System.IO*). Os objetos instanciados desta classe, contêm dados relativamente a ficheiros armazenados no computador. Os *FileInfo* dentro do *array* passado como argumento, são instâncias dos ficheiros **.SRC* que foram pedidos para serem convertidos.

[36] Quando nos referimos a máquina virtual, é ao sistema que correr o código *CLR* (*Common Language Runtime*) compilado em primeira fase do código C#. Esta máquina virtual, compila o código *CLR* com um compilador JIT e converte este em IL que por fim é processado pelo processador. Por estas duas fases distintas de compilação e execução da aplicação, C# é uma linguagem híbrida.



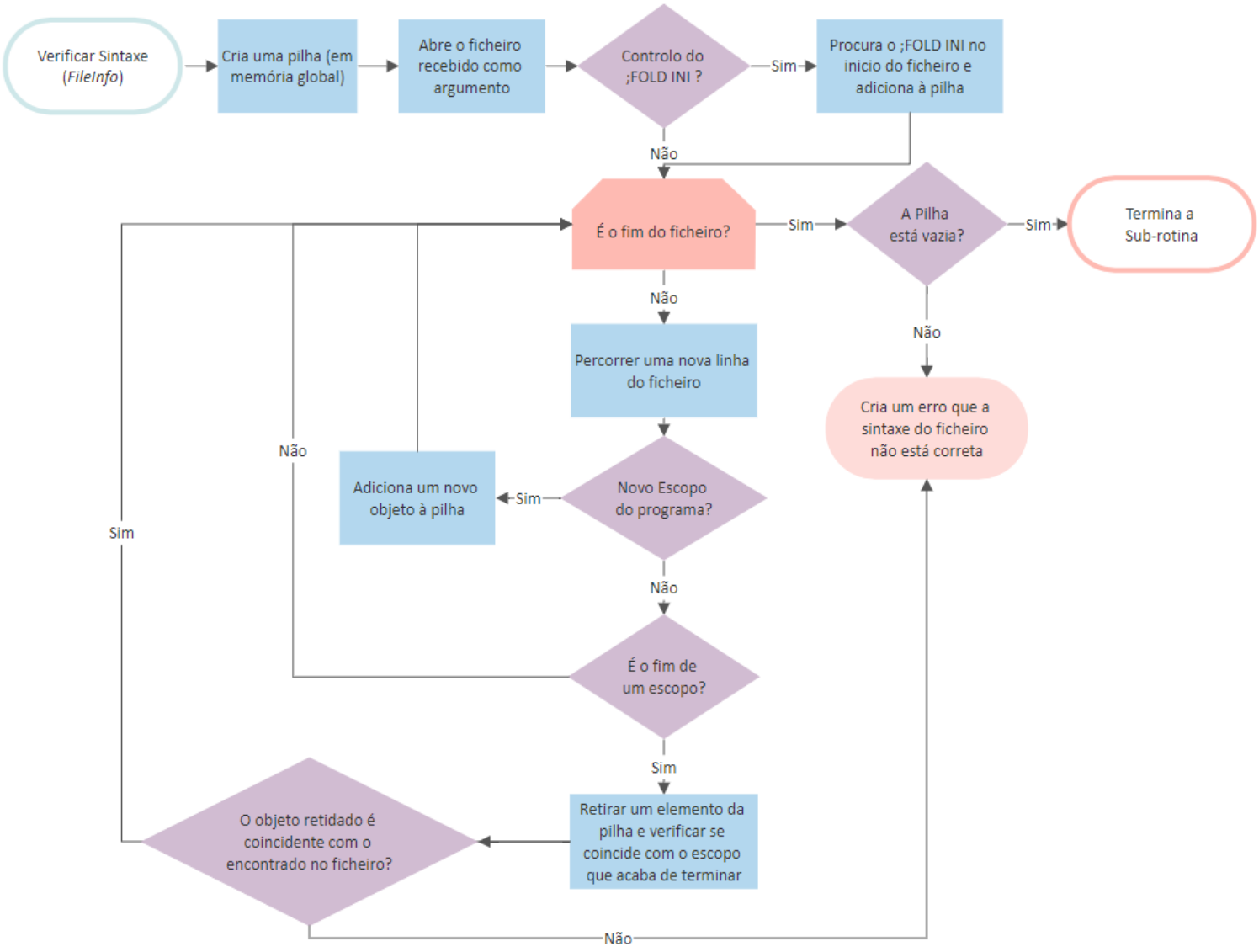
Fluxograma 1 – Visão geral do programa de converter código KR C1 em código KR C4.

A função começa com uma estrutura de repetição que a cada execução percorre um elemento do *array*, até que este se encontre vazio. Um exemplo de uma implementação de percorrer todos os *FileInfo*, dentro do *array* é apresentado no Bloco de Código 13 em *C#*. Quando este se encontrar vazio, o programa acaba.

```
foreach (FileInfo ficheiro in ArrayDeFicheiros) //Exemplo
```

Bloco de Código 13 - Exemplo de como percorrer todos os elementos dentro de um Array<FileInfo>.

Quando este não se encontra vazio, é executado o código dentro da estrutura de repetição. Pegando no exemplo do Bloco de Código 13, o objeto *ficheiro* (com informações do ficheiro de código-fonte *KR C1*) vai ser passado como referência durante a chamada da função "Verificar Sintaxe", no primeiro passo do escopo da estrutura de repetição. Para que seja mais fácil de entender esta função, o seu funcionamento está descrito no Fluxograma 2.



Fluxograma 2 - Modo de funcionamento da verificação da sintaxe do código.

Esta função tem como objetivo de verificar se a sintaxe do código está correta. Para isto, é necessário criar a sua pilha, contendo esta informações sobre o escopo do programa e ordens de instrução dentro de cada escopo.

Para que a pilha possa ser carregada com a informação do código, é criada uma instância de *StreamReader*, classe responsável por ler dados de um ficheiro de texto da memória em *C#*. Vamos referir-nos ao objeto desta instância pelo nome de *leitor*, assim como também foi feito no Bloco de Código 14.

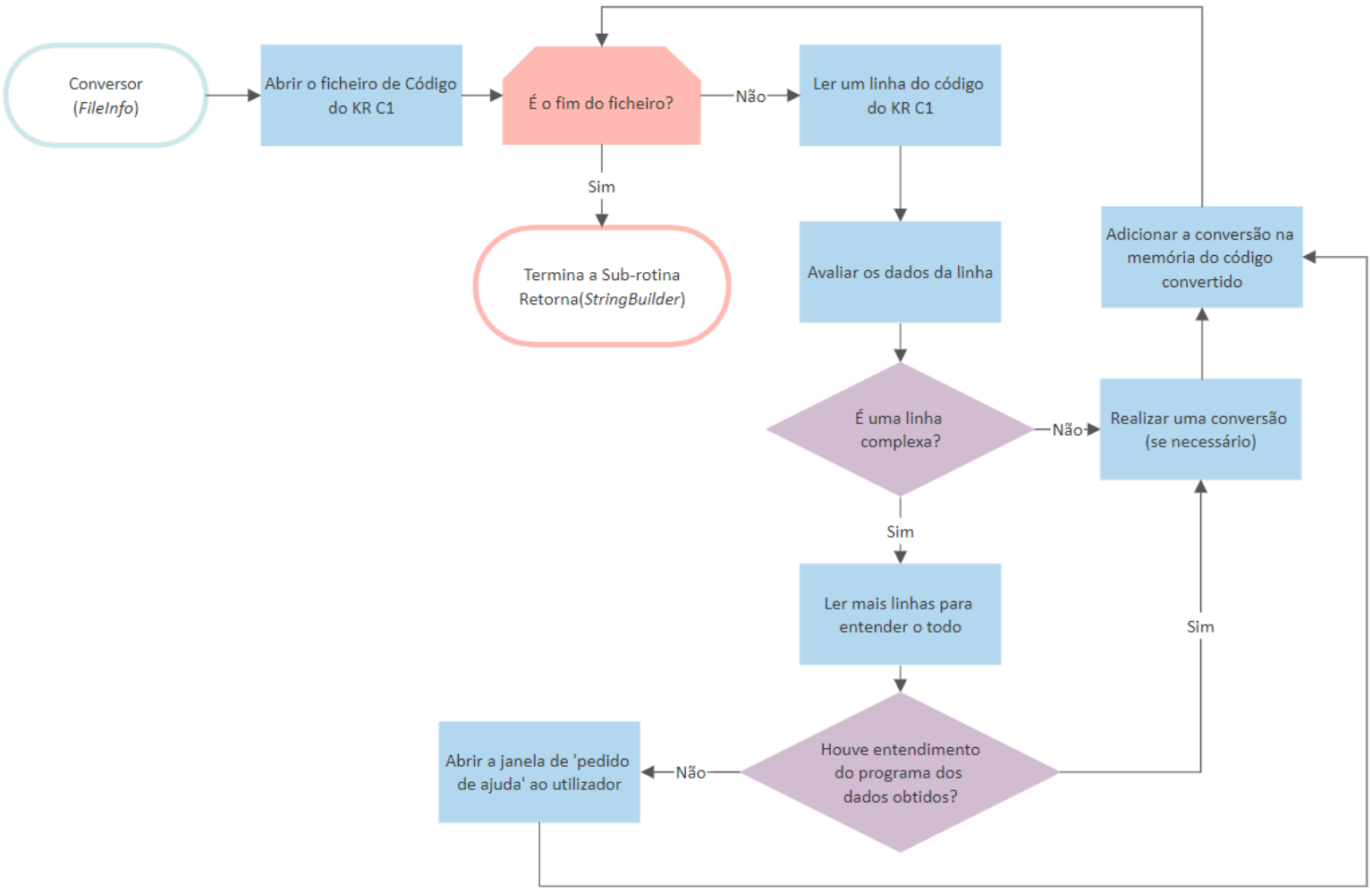
Após a criação do *leitor* é lido linha a linha o código-fonte presente no *ficheiro*. Consoante o conteúdo da linha de código, diferentes ações podem ser realizadas. No caso de ser o início de um novo escopo, um novo elemento é adicionado à pilha, enquanto, que caso a linha de código lida especifique o fim de um escopo, a aplicação verifica se este é o fim do escopo esperado. Caso seja, é retirado um elemento da pilha, caso não seja é lançado um erro que informa o programa que existe um erro na sintaxe do ficheiro. Um exemplo genérico desta implementação é apresentado em baixo.

```
using(StreamReader leitor = new StreamReader(ficheiro))
{
    while(leitor.EndOfStream)
    {
        string linha = leitor.ReadLine();
        //Código restante de verificar escopos e adicionar à pilha
    }
}
```

Bloco de Código 14 - Representação de uma instância de um StreamReader e da leitura, do ficheiro aberto, linha a linha.

Por fim, quando todas as linhas do *ficheiro* são lidas, o *leitor* é desalocado e é verificada se a pilha está vazia. Caso a pilha esteja vazia, a verificação da sintaxe está correta e a função é finalizada. No caso de a pilha não estar vazia, é lançado o erro de verificação da sintaxe, que termina o processo de conversão do *ficheiro*.

No caso de não ser lançado nenhum erro pela função anterior, retorna-se ao Fluxograma 1, onde o próximo passo é a chamada da função “Conversor”. Nesta função é onde os ficheiros de código da linguagem do *KR C1* serão convertidos para a linguagem de *KR C4*. Da mesma forma que a função anterior, o *FileInfo* é passado como argumento. O funcionamento desta função pode ser acompanhado no Fluxograma 3.



Fluxograma 3 - Explicação de como é feita a conversão do programa.

Da mesma forma realizada anteriormente, o ficheiro de código-fonte do *KR C1* é aberto e enquanto houver linhas no ficheiro, estas serão lidas linha a linha.

Cada linha lida é avaliada quando ao tipo de instrução. Os tipos de instruções possíveis são a chamada de uma função, o início de uma estrutura condicional, o final de uma estrutura de repetição, um comentário, um movimento linear do robô, a ordem de ativar uma saída do robô, etc. Cada um desses casos é tratado de uma forma única, no entanto, para a descrição deste processo, podemos agrupá-las em duas categorias, sendo que umas instruções se incluem nas expressões complexas e outras nas expressões simples. Exemplos de instruções serão apresentados no bloco de código seguinte.

```
;Exemplo de expressões simples
WHILE  (E_POUT1==E_POUT2)
IF E_POUT1 THEN
AFFICHE (1,1,"M" )
AFFICHE (3,1,"M" )
ELSE
AFFICHE (2,1,"M" )
AFFICHE (4,1,"M" )
ENDIF
HALT
ENDWHILE

;Exemplo de expressões complexas
;FOLD     LIN     P_pos16      Vel=     2     m/s     CPDAT12;%{PE}%R
2.4.14,%MKUKATPBASIS,%CMOVE,%VLIN,%P   1:LIN,   2:P_pos16,   3:,   5:2,
7:CPDAT12
LDAT_ACT=LCPDAT12
BAS (#CP_DAT )
FDAT_ACT=FP_POS16
BAS (#FRAMES )
BAS (#VEL_CP,2 )
LIN  XP_POS16
;ENDFOLD
;FOLD EVTA.VAL 1, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0, 0,   0, 0, 0,
0;%{PE}%MKUKATPUSER
$ADVANCE=0
G_EVENT1=1
G_EVENT2=0
G_EVENT3=0
G_EVENT4=0
G_EVENT5=0
G_EVENT6=0
G_EVENT7=0
G_EVENT8=0
G_EVENT9=0
G_EVENT10=0
G_EVENT11=0
G_EVENT12=0
G_EVENT13=0
G_EVENT14=0
G_EVENT15=0
G_EVENT16=0
ATT_EVT ( )
$ADVANCE=3
;ENDFOLD
```

Bloco de Código 15 - Exemplos de expressões simples e expressões complexas na sintaxe KRL do KR C1.

É possível entender pelo exemplo fornecido acima que as expressões simples, por norma, numa única linha são compreensivas e não é necessário ler mais código para a sua compreensão. As expressões complexas são o oposto, sendo que se for apenas lida uma linha de código não é possível ao controlador compreender o seu significado e qual o objetivo daquela expressão, pelo que é necessário ler as linhas seguintes de forma a haver uma compreensão sobre o que é pretendido.

De modo igual, quando existe uma linha de código para ser convertida, esta é analisada, para perceber se é uma expressão simples ou complexa. Consoante esta análise, a linha de código é diretamente convertida ou são lidas mais linhas até que seja possível compreender o conjunto do código, de forma a ser possível realizar a conversão, no caso da conversão simples e da conversão complexa, respetivamente.

Por fim, o código convertido é adicionado a um *StringBuilder*, que é uma classe otimizada para construir *strings*. Esta *StringBuilder* é retornada pela função e a execução do programa volta ao Fluxograma 1.

No próximo passo do Fluxograma 1, é feito um *cast* para que seja possível enviar os dados convertidos como parâmetro da função “Verificar Sintaxe”. Neste ponto, recorrendo a duas *threads* sincronizadas, uma *thread* recria a pilha original do código do *KR C1* e a segunda *thread* cria a pilha do código convertido para *KR C4*.

Ambas as *threads* leem os dados e trocam informação uma com a outra de forma síncrona e protegida, sendo que ambas processam ou aguardam pela confirmação dos dados da outra. Durante a criação desta pilha, ambas as pilhas têm de corresponder. Se as pilhas em algum ponto da leitura não corresponderem uma à outra, então é porque o código convertido não corresponde ao código original e é lançado um erro de conversão. Se as pilhas corresponderem, significa que o código convertido está igual ao original, ou seja, correto. Neste caso o código convertido é exportado para um novo ficheiro do tipo **.SRC*, assim como também é gerado um ficheiro **.dat* associado ao ficheiro anteriormente criado.

Por fim, é verificado se existem mais ficheiros para serem convertidos e no caso de haver um novo ficheiro passa pelo processo de conversão. Caso não haja, a conversão é dada como finalizada.

3.1.5 – Conclusões

Após a aplicação ter sido desenvolvida e testa, é atualmente possível realizar conversões dos códigos de forma rápida, versátil, adaptável às situações que lhe são apresentadas e autónoma, sendo que 95% das ações são realizadas com sucesso e sem necessidade da intervenção ou análise humana, cumprindo com o proposto inicialmente. Esta aplicação é inovadora, permitindo automatizar e aumentar em muito a produtividade da empresa, uma vez que permite que não se gaste tempo nem recursos humanos em tarefas chatas e minuciosas.

3.1.6 – Previsões para o futuro

As previsões para o futuro da aplicação é o desenvolvimento, que já está a decorrer, da conversão de linguagens de programação de outros controladores, inclusivamente os de outros fabricantes. Desta forma, a aplicação tem como objetivo, no futuro, ser uma ferramenta em que permite ao utilizador escrever uma vez o código para um robô industrial e esta corra em todos os restantes, sendo a aplicação capaz de traduzir a linguagem inicialmente escrita para qualquer outra, de forma que o robô a compreenda e execute o pretendido, ou seja, ser uma ferramenta 'poliglota'.

3.1.7 – Limitações da conversão

A maior limitação ao programa é o facto de não poder ser mais preciso na conversão de um movimento de um robô, assim como a converter os seus dados relativamente a pontos, garras e outras partes físicas.

Isto porque não existe uma relação conhecida entre um ponto de um robô do controlador *KR C1* e um robô do controlador *KR C4*. Mesmo que os seus eixos sejam iguais, não existem garantias sobre as distâncias, dados dos *encoders* e posição absoluta *versus* posição dada pelo controlador do robô. Deste modo, expressões de movimentos são alteradas durante a conversão por um comentário descritivo daquele movimento, como apresentado no Bloco de Código 16 e Bloco de Código 17.

Outra das razões para que os movimentos sejam substituídos por um comentário é pela evolução dos movimentos no controlador *KR C4*, sendo recomendado, por exemplo, utilizar um *SLIN* ao invés do movimento *LIN* (obsoleto). O movimento *SLIN* é mais recente e não existe no controlador *KR C1*.

```
;Exemplo de movimentos no código fonte KR C1
;FOLD    PTP    P_ant1_pos    CONT    Vel=    100    %    PDAT11;%{PE}%R
2.4.14,%MKUKATPBASIS,%CMOVE,%VPTP,%P  1:PTP,  2:P_ant1_pos,  3:C_PTP,
5:100, 7:PDAT11
PDAT_ACT=PPDAT11
BAS (#PTP_DAT )
FDAT_ACT=FP_ANT1_POS
BAS (#FRAMES )
BAS (#VEL_PTP,100 )
PTP  XP_ANT1_POS C_PTP
;ENDFOLD
;FOLD     LIN     P_pos16      Vel=     2     m/s     CPDAT12;%{PE}%R
2.4.14,%MKUKATPBASIS,%CMOVE,%VLIN,%P  1:LIN,  2:P_pos16,  3:,  5:2,
7:CPDAT12
LDAT_ACT=LCPDAT12
BAS (#CP_DAT )
FDAT_ACT=FP_POS16
BAS (#FRAMES )
BAS (#VEL_CP,2 )
LIN  XP_POS16
;ENDFOLD
```

Bloco de Código 16 - Instrução de um movimento PTP e outro LIN no robô KR C1.

```
;Resultado do bloco anterior ao fim de ser convertido
;#Movimento PTP Continuo para ponto P_ant1_pos;#RRS
;#Movimento LIN para ponto P_pos16;#RRS
```

Bloco de Código 17 - Resultado obtido pela aplicação quando convertido o Bloco de Código 16.

3.2 – Reconfiguração de um posto de linha de montagem

3.2.1 – Visão geral da atividade

A atividade agora apresentada foi desenvolvida para um cliente, cujos dados não podem ser revelados para garantir a confidencialidade do mesmo. Para esta atividade, ainda que existissem restrições criadas pela pandemia, já foi possível realizar o trabalho junto das máquinas, sempre que necessário.

A mesma consistiu em projetar, planear, executar e programar um posto de trabalho de uma linha de montagem de caixas de velocidade. No entanto, esta atividade não parte do ponto zero, uma vez que é um trabalho de reformulação da linha, sendo que este posto vai ser readaptado às novas mudanças colocadas na linha. Desta forma, o objetivo do cliente era, com o mínimo de intervenção possível na parte física e desperdício, modificar o modo de funcionamento do posto, para que este realize autonomamente uma operação que antes era realizada por um operador.

Ainda nesta introdução iremos fazer uma breve descrição dos processos que esta atividade exigiu, seguindo por uma apresentação dos componentes de diversas zonas do posto, assim como os seus problemas e a solução à atividade.

O objetivo desta atividade foi desenvolver-se um programa no robô para que este colocasse uma ogiva no local apontado pela Figura 61. De notar que a foto é meramente ilustrativa e não é coincidente com os modelos das caixas de velocidades com os quais se trabalhou. No entanto, o local onde foi colocada a ogiva é equivalente em todos os sentidos à apontada na Figura 61. Referir ainda que a parte riscada a vermelho é montada posteriormente, sendo que no momento do trabalho do nosso posto esta peça não está ainda montada, pelo que nos referimos apenas ao seletor redondo e não à restante peça.

Figura 61 - Caixa exemplo. Esta não corresponde a uma caixa real. Fonte: (CAIXAS DE VELOCIDADES, s.d.)

Esta ogiva é um elemento cilíndrico, com uma das pontas cónicas e oca por dentro. A sua função é guiar um vedante para que este entre à pressão, exercida por uma prensa, no seu respetivo lugar. O trabalho da prensagem do vedante é realizado no posto seguinte, e o posto posterior a esse, é retirada a ogiva, por um operador, para que esta volte a ser reutilizada.

3.2.1.1 – Planta da fábrica

Vamos apresentar um esquema da planta para ilustrar os elementos presentes na linha.[37]

[37] Nota: esta planta foi realizada explicitamente para este relatório de estágio, pelo que não corresponde na sua integra na planta real do cliente, de modo a respeitar a proteção de dados do mesmo.

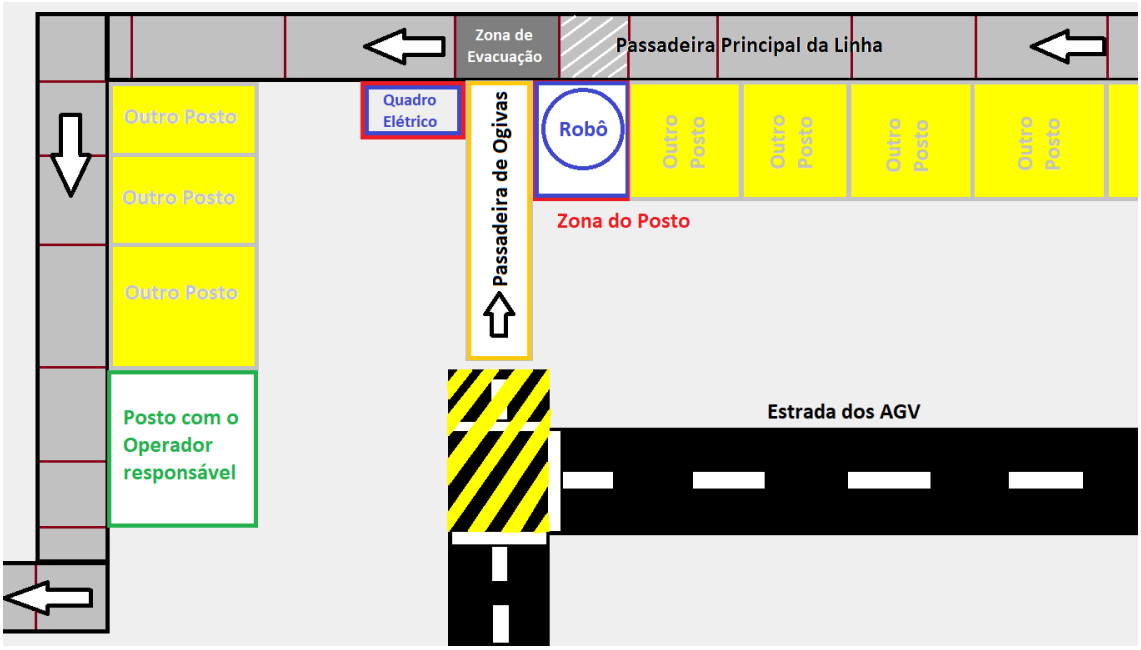


Figura 62 - Planta figurativa da estrutura da fábrica. Fonte: Própria

Esta planta representa o que se encontrava no posto no momento em que a atividade teve início. Podemos ver o posto em que vamos realizar as alterações delimitado a vermelho e azul. Este posto é um posto 100% automático, não tendo um operador junto ao mesmo, no entanto existe um operador no posto diretamente anterior (à direita do robô).

Devido às limitações do espaço e ao facto de haver um operador a trabalhar ao lado, o posto conta com um robô colaborativo *KUKA LBR iiwa 14 R820*, que permite usar funções de segurança para proteção do operador. Este robô será abordado mais à frente.

O posto também conta com um PLC *Schneider/Telemecanique*, dentro do quadro, um tapete transportador de correias descrito como “passadeira de Ogivas”, entre outros elementos presentes nesse transportador, assim como no transportador principal. O transportador principal é de correntes. Neste pode ser encontrado, a um cinzento relativamente mais escuro, a zona onde a palete com a caixa de velocidade é forçada a parar de modo que o robô possa realizar o trabalho daquele posto.

O PLC foi descrito anteriormente como sendo da *Schneider/Telemecanique*, sendo que apesar do PLC ser da *Telemecanique*, a *Schneider Electric* comprou esta empresa. Por esse motivo algumas cartas para este PLC são *Schneider* e outras *Telemecanique*. Por isto, nos vamos referir a este (quando não usada a abreviatura PLC) pelas duas marcas.

3.2.1.2 – Linha principal e a Palete das Caixas de Velocidades

Uma vez que o transportador principal da linha contém muitos postos com um tempo de ciclo diferente, este encontra-se em contínuo movimento. Para que as paletes parem em cada posto da linha, existem *stoppers.* Estes, também conhecidos como batentes, estão apresentados com uma linha vermelha perpendicular ao transportador na Figura 62 e apresentado um exemplo real na Figura 63.

Figura 63 - Exemplo de um stopper. Fonte: (Batente bloco industrial de Palete - Imagem em Alta Resolução, 2015)

Para além destes *stoppers*, a nossa zona de paragem na linha conta também com um indexador. O indexador, é responsável por levantar a palete de forma que esta fique suspensa no ar e sem contacto com as correias do transportador. Este processo é essencial porque sem ele, a palete está a ser continuamente empurrada contra o seu *stopper*. Esta força contínua a empurrar a palete, pode provocar um comportamento imprevisível, sendo que esta pode ficar junto do seu *stopper*, como também pode voltar para trás com a força do embate.

A situação em que a palete bate e recua para trás é limitada, uma vez que cada *stopper* também conta com um antirretorno. O antirretorno é um elemento mecânico que obriga as paletes a circularem no sentido correto da linha, funcionando como um batente no sentido oposto. No entanto, existe sempre um espaço de folga entre o *stopper* e o seu antirretorno. Este espaço permite pequenas movimentações da palete de trás para a frente, o que impossibilita o robô de trabalhar de forma precisa, uma vez que a palete não se encontra sempre na mesma posição. Por este motivo, é

necessário um indexador que garante sempre a mesma posição da palete, assim como impede que haja uma força horizontal a ser aplicada na mesma.

Enquanto o antirretorno é um elemento totalmente mecânico, o *stopper* e o indexador são elementos mecânicos e pneumáticos. A parte pneumática conta com válvulas direcionais, que são acionadas pelo PLC. Para que o PLC consiga gerir a presença da palete no posto, este está equipado com sensores.

O primeiro sensor que é apresentado na linha é o sensor de presença de palete. Este é um sensor magnético, o que permite uma boa fiabilidade, tanto da posição da palete como ao sujo da linha. A fiabilidade quanto ao posicionamento da palete deve-se ao facto de esta ser de alumínio (metal sem propriedades magnéticas), com uma única chapa de aço inoxidável ferrítico (metal com propriedades magnéticas). Na Figura 64 é possível ver uma ilustração lateral, realizada para este relatório de estágio, da palete da caixa de velocidades. A posição da chapa de aço está representada a cinzento mais escuro.

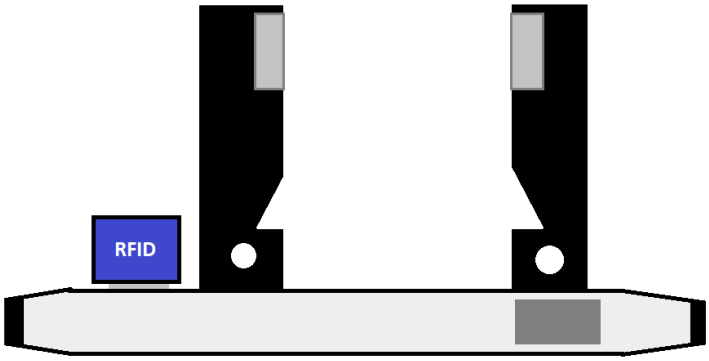


Figura 64 - Exemplo criado não real da palete da caixa de velocidades. Fonte: Próprio

O segundo sensor mais importante é um sensor RFID capaz de ler e escrever, que cada posto da linha contém. Do mesmo modo, cada palete das caixas de velocidade tem uma etiqueta RFID que pode ser lida e escrita. Esta etiqueta contém informações relevantes que podem responder às seguintes perguntas:

1. Qual o modelo da caixa de velocidades?
2. Próximo trabalho a ser realizado na caixa?
3. Último trabalho a ser realizado na caixa?
4. A caixa tem algum defeito?
5. A palete tem de ser evacuada?

Estas informações são interpretadas pelo PLC e consoante as mesmas, diversas ações podem ser despoletadas. No final do trabalho, o PLC também escreve na etiqueta as informações que são necessárias para a caixa prosseguir o seu caminho.

Por fim, o último sensor de interesse que se encontra na linha, foi instalado durante a nossa intervenção no posto. Este é uma fotocélula e será abordada qual a sua finalidade mais à frente.

3.2.1.3 – Transportador de ogivas, palete de ogivas e ogivas

A passadeira de ogivas era inicialmente carregada com paletes por um AGV, no entanto este processo ia passar a ser descontinuado. A nossa intervenção tinha também de solucionar esta questão, sendo que agora era o operador do posto o responsável (consultar Figura 62) que tinha de recarregar as ogivas.

Os elementos com que este transportador já contava eram o próprio transportador, sendo este de correias, três *stoppers* (pneumáticos), um indexador (pneumático) e fotocélulas posicionadas para gerir os múltiplos *stoppers*. Estes *stoppers* e fotocélulas eram usados para gerir as paletes no transportador e as que entravam pelos AGV. Contudo, o AGV já não ia mais abastecer este transportador e o seu caminho também foi alterado.

A palete das ogivas é uma palete pequena, que tem dimensões de 120x500x100mm e era capaz de carregar 20 ogivas. Uma ilustração desta palete pode ser vista na Figura 65.

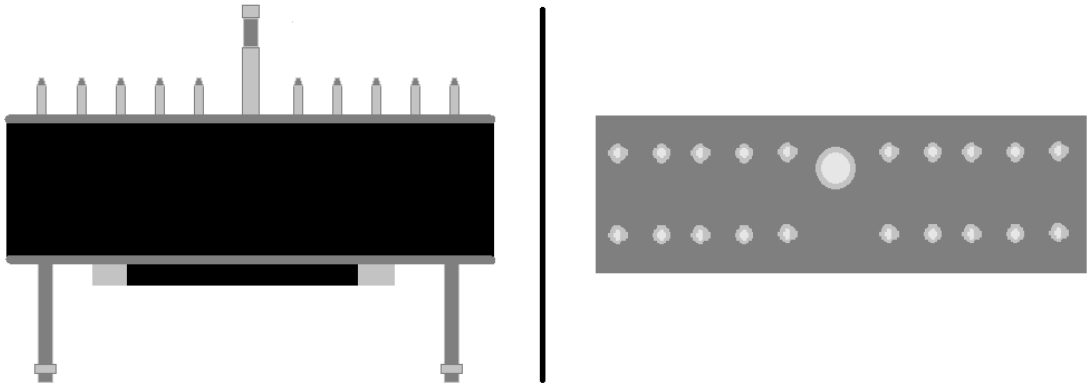


Figura 65 - Esquema figurativo da vista frontal e superior, respetivamente, da palete de ogivas. Fonte: Próprio

Esta palete tem então 20 pinos onde são colocadas as ogivas e no centro tem um pino mais alto e mais grosso que é destinado ao robô, para que este transporte a palete. Na parte inferior da palete consegue-se ver duas pernas, estas eram usadas para encaixar a palete das ogivas na palete das caixas de velocidades.

Numa fase inicial, projetou-se o robô agarrar a palete na sua totalidade e colocá-la na palete da caixa de velocidade, de seguida a palete seguia até ao operador responsável e este retirava-a, recarregava-a e voltava a colocá-la no transportador. Contudo, esta ideia foi abandonada, pela má posição do operador, pelo peso, perda de tempo, distancia e dificuldade de tirar a palete de ogivas da palete de caixa de velocidade. A resolução deste problema será apresentada mais à frente.

As ogivas são um cilindro com uma cabeça cónica, feito de alumínio oco. Estas devem ter cerca de 40mm de altura e 15mm de diâmetro. Uma imagem real de uma ogiva pode ser vista na Figura 66.

Figura 66 - Exemplo de uma ogiva pousada em um dos pinos da palete. Fonte: Própria

Posto isto, a solução passou por alterar o modo de funcionamento do transportador e permitir apenas que duas paletes estivessem ao mesmo tempo no transportador.

3.2.1.4 – Robô Colaborativo

Neste projeto foi usado um robô colaborativo, o *KUKA LBR iiwa 14 R820*, uma vez que não houve a possibilidade de usar um robô convencional. O motivo que impossibilitou a utilização de um robô convencional foi o pouco espaço disponível, uma vez que este não era suficiente para poder instalar cercas de proteção para o mesmo. A solução foi usar um robô colaborativo, uma vez que “as cercas de proteção do espaço de trabalho dão espaço à colaboração homem-robô” (KUKA AG, 2018). A presença de um robô nesta atividade é um elemento-chave, permitindo que o trabalho seja 100% autónomo.

A diferença entre um robô industrial tradicional e um robô colaborativo é o facto de que “os robôs colaborativos são uma nova geração de robôs articulados em 6 ou mais eixos, projetados para operar de modo seguro em instalações industriais partilhadas com seres humanos” (Dhouibi, 2019). Por serem desenvolvidos para isto, são de rápida reação “graças aos sensores de torque nas articulações, o *LBR iiwa* reconhece imediatamente contactos e reduz instantaneamente a força e a velocidade. Ele manuseia componentes sensíveis através do controle de posição e de resiliência sem pontos de cisalhamento e de esmagamento” (KUKA AG, 2018).

Figura 67 - Imagem do robô colaborativo KUKA iiwa 14 R820. Fonte: (KUKA Robots IBÉRICA, S.A., 2020)

Além de toda a tecnologia de controlo no robô, toda a sua estrutura também é pensada e desenhada para trabalhar com seres humanos, sendo que este é totalmente arredondado e sem arestas. A sua garra também tem de ser personalizada, de modo que seja impossível de magoar alguém com a mesma. Para isto, a nossa garra está coberta por uma proteção completamente arredondada com um único buraco para que seja possível entrar nas peças como demonstrado na Figura 68.

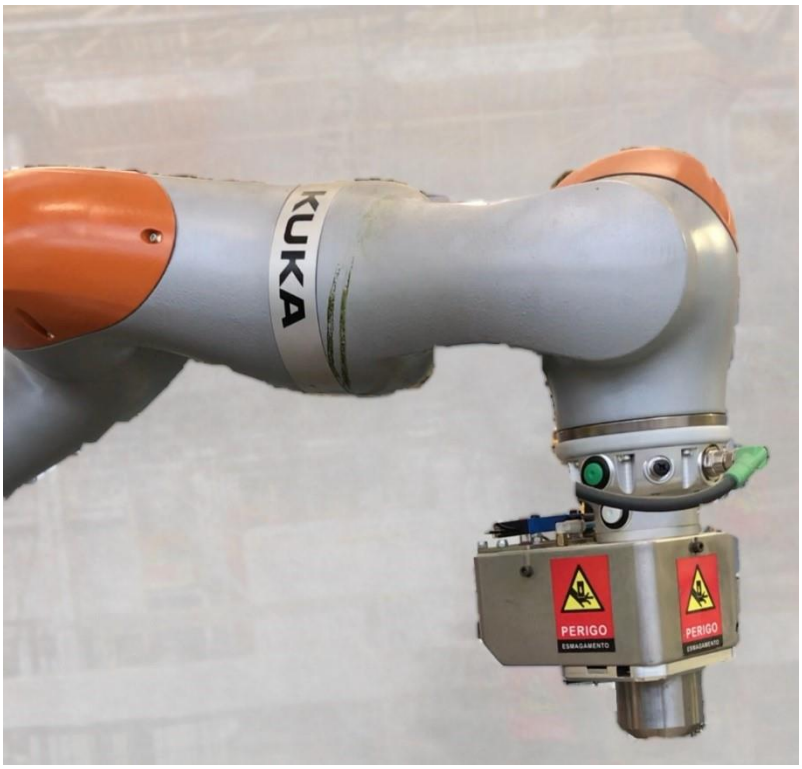


Figura 68 - Proteção da garra do robô colaborativo. Fonte: Próprio

A garra em si é uma pinça que permite ao robô agarrar em uma ogiva como também pegar na palete das ogivas. Esta garra é pneumática e controlada pelo robô.

A programação deste robô é feita através do *software Sunrise Workbench (apresentado na* Figura 69*)*, sendo que este é completamente programado na linguagem *Java*. Contudo, é possível abstraímos da linguagem de programação *Java* e programar em *sequencial charts* com blocos já pré-feitos, sendo ainda possível criar os nossos próprios blocos em *Java*. O *sequencial chart* (Figura 70) é a forma mais fácil e pratica de programar o robô, sendo que programando o robô desta forma torna-o muito mais simples de programar que um robô industrial tradicional. Esta facilidade na programação permite que os operadores, com pouca formação, consigam reprogramar as tarefas do robô e os seus trajetos.

Figura 69 - Logótipo do software KUKA Sunrise Workbench. Fonte: (Kuka LBR iiwa Programming Learn Module, s.d.)

Figura 70 - Exemplo de um fragmento de um programa em sequencial chart. Fonte: Próprio

Quanto a questões de segurança, a aplicação desenvolvida permite diferentes configurações de segurança para que o sistema robótico cumpra as normas de segurança em vigor. Vamos, de seguida, abordar todas as condições de segurança da aplicação desenvolvida.

Em primeiro lugar é necessário entender quais são as zonas de interesse para o robô, sendo estas o acesso às palete de ogivas e o acesso às paletes da linha. Todas as restantes posições não são posições de interesse e por isso vamos descartá-las. Desta forma, pegando na planta da fábrica conseguimos traçar a zona de interesse para o robô se movimentar como ilustrado na Figura 71 - A área a verde

representa a área permitida para o robô se movimentar, o espaço colaborativo, e o círculo a laranja representa o alcance máximo do robô.

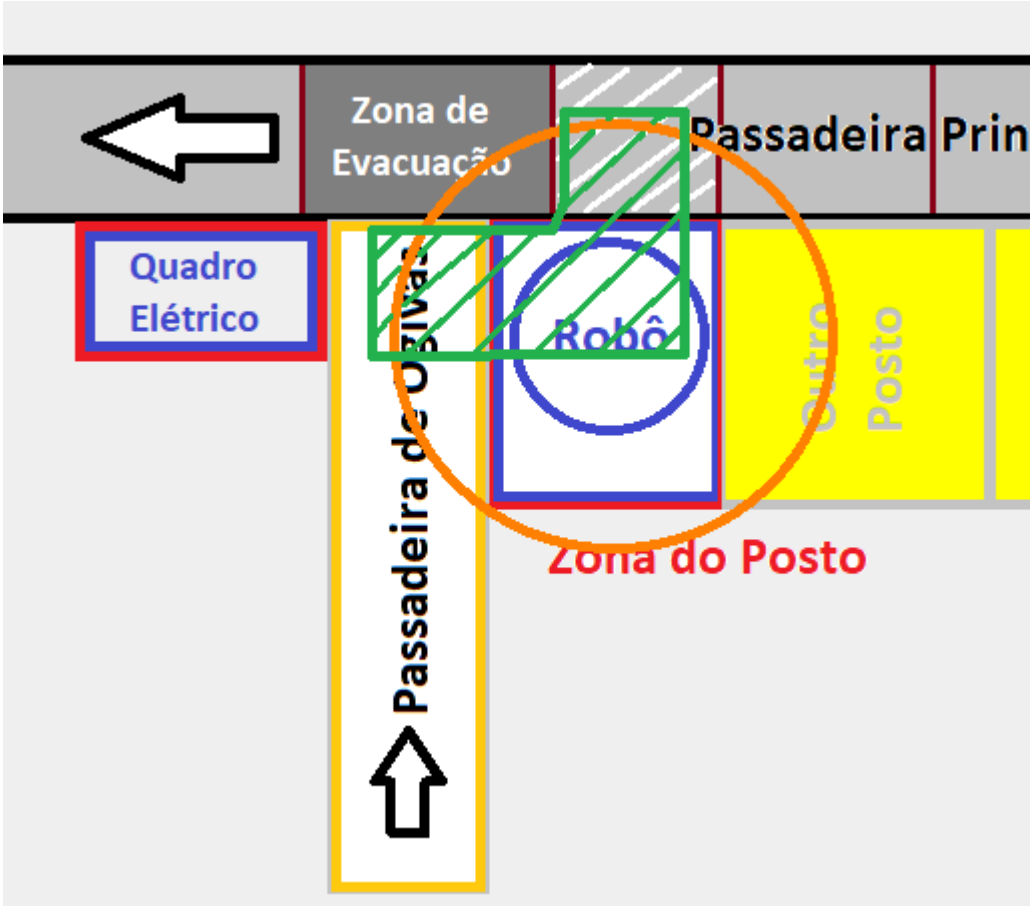


Figura 71 - A área a verde representa a área permitida para o robô se movimentar, o espaço colaborativo, e o círculo a laranja representa o alcance máximo do robô. Fonte: Própria

Definida a área de interesse e verificado que esta não invade a área do operador é agora necessário programar o robô nas suas configurações de segurança. Para isto precisamos de compreender como funcionam as configurações de segurança do robô. Consultando o manual de instruções *KUKA Sunrise.OS 1.11/KUKA Sunrise.Workbench 1.11/Operating and Programming Instructions for System Integrators* é possível encontrar as explicações necessárias.

A primeira noção a mencionar é que o sistema robótico está permanentemente a se auto-monitorizar. As condições que este auto-monitoriza são definidas pelo programador no *Sunrise Workbench* através de funções de segurança. Cada função de segurança contém três condições e uma reação. Caso as condições sejam violadas a reação é executada. As condições são chamadas pelo *software* de AMF (*Atomic Monitoring Function*) e estas podem ser ligadas até três em série por cada função de segurança, como apresentado na Figura 72. As AMF podem ter apenas dois valores:

- Valor 0: AMF violado;
- Valor 1: AMF não violado.

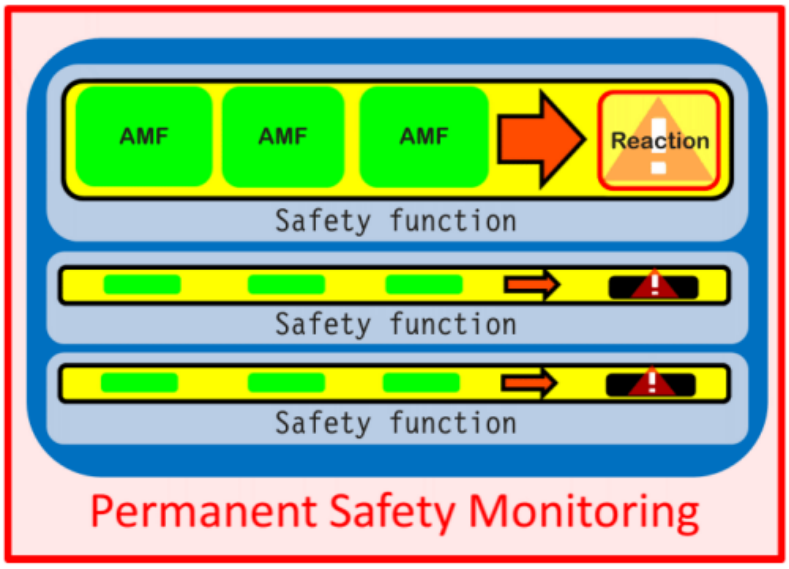


Figura 72 - Imagem retirada do manual de instruções do robô. Fonte: (KUKA AG, 2016, p. 206)

A forma mais simples de se demonstrar quando a reação é executada, sendo que a reação, normalmente, é a paragem imediata do sistema robótico, é recorrendo à Figura 73 com uma linha de *Ladder Logic*. Avaliando a linha de *Ladder Logic*, consegue-se compreender o anteriormente dito, sendo que cada função de segurança definida, representa uma nova linha igual à apresentada. Por fim, referir apenas que as AMF não definidas, em cada função de segurança, atuam como se estivessem violadas, tendo sempre o valor 0.

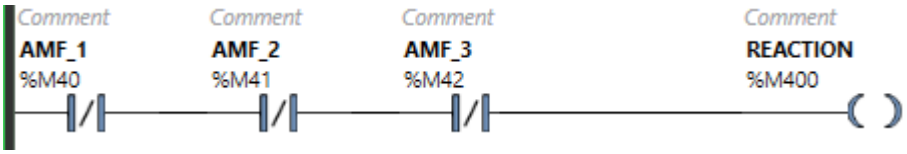


Figura 73 – Exemplo em Ladder Logic de como funciona as configurações de segurança do robô colaborativo. Fonte: Própria

Entendido o modo de funcionamento da segurança do robô, olhamos agora aos parâmetros necessários de preencher para configurar uma função de segurança. Para auxiliar a visualização foi retirado um *printscreen* do *Sunrise Workbench* apresentado na Figura 74.

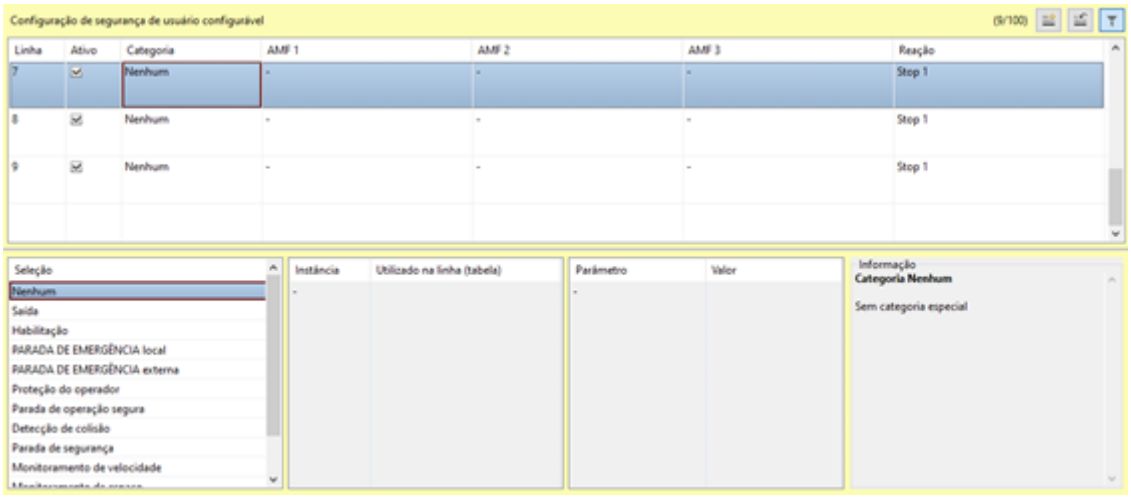


Figura 74 - Printscreen das configurações de segurança do Sunrise Workbench. Fonte: Próprio

Como visualizado, existem sete configurações em uma função de segurança que se podem programar. Cada linha da tabela representa uma função de segurança. A primeira coluna representa apenas o número da função de segurança, sendo que a linha atual selecionada é a linha 7. Na coluna seguinte tem-se uma *checkbox* que permite ativar e desativar a função de segurança.

A coluna 'Categoria' representa a categoria que é mostrada no controlador quando todas as condições de segurança são violadas. Isto é algo apenas estético e representa o que vai ser mostrado ao utilizador, sendo que a sua configuração não é crítica para o funcionamento da função de segurança. Caso suscite interesse, as categorias podem ser consultadas nesta fonte (KUKA Sunrise.OS 1.11 KUKA Sunrise.Workbench 1.11 Operating and Programming Instructions for System Integrators, 2016, p. 207).

As quatro restantes colunas representam as configurações da função de segurança, sendo que as primeiras três colunas representam as AMF e a última representa a reação a ter quando as três condições da linha forem violadas.

Abordando mais a fundo as AMF estas podem ser de diversos tipos. Existem diferentes configurações de AMF:

- Puramente de controlo e que não fazem sentido estarem sozinhas e como a da Figura 75 à esquerda;
- Que tanto podem funcionar seguidas de uma AMF de controlo como também podem ser avaliadas por si só, como a da Figura 75 ao centro;
- Que não faz sentido estarem seguidas por outras, como por exemplo a configuração padrão da betoneira de emergência, Figura 75 à direita.

Aos que suscite maior interesse sobre os AMF, todos os tipos de AMF podem ser consultados nesta referência (KUKA AG, 2016, pp. 208-271).

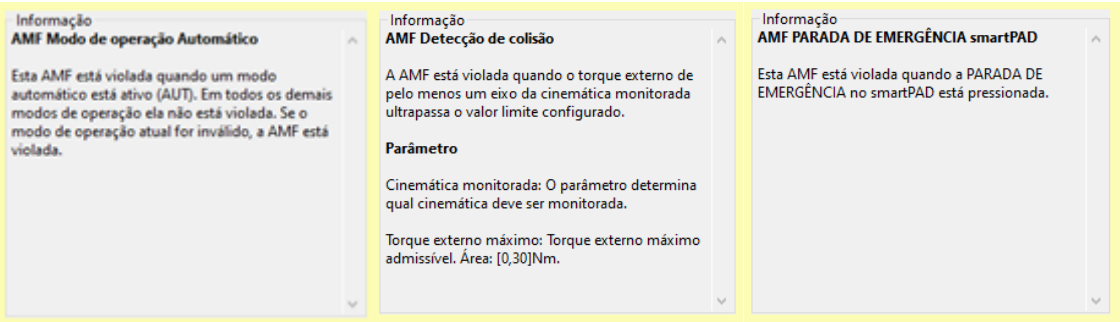


Figura 75 – Diferentes modos de AMF. Fonte: Próprio

Quanto às reações, quando as AMF são violadas podem ser quatro diferentes, pelo que o *Sunrise Workbench* as descreve da seguinte forma:

- Stop 0: Se a linha estiver violada, é executada uma parada da categoria 0. Aqui a energia de acionamento é imediatamente desligada e os freios fechados;
- Stop 1: Se a linha estiver violada, é executada uma parada da categoria 1 (não fiel à trajetória). Aqui o robô é conduzido à parada com os acionamentos. Tão logo um eixo pare, a energia de acionamento é desligada e os freios do eixo são fechados. O procedimento de frenagem é monitorado orientado à segurança e, em caso de erro, é realizado um Stop 0;
- Stop 1 (fiel à trajetória): Se a linha estiver violada, é executada uma parada fiel à trajetória da categoria 1. Aqui o robô é conduzido à parada com os acionamentos. Após atingir a parada, a energia de acionamento é desligada e os freios são fechados. O procedimento de frenagem é monitorado orientado à segurança e, em caso de erro, é realizado um Stop 1;
- Saída: Se a linha estiver violada, o nível de saída da saída configurada é definido como LOW.

Entendido agora como funcionam as configurações de segurança do robô, vamos demonstrar como foram feitas as nossas configurações de segurança.

No nosso robô, as primeiras duas funções de segurança genéricas, sendo que estas são a paragem imediata do robô caso o botão de emergência do *SmartPAD* do robô estivesse pressionado ou caso o botão de emergência do posto tivesse pressionado. Para a primeira função de segurança referida existe uma AMF direta enquanto para a segunda função de segurança foi usado a AMF do tipo entrada, que é usada uma entrada de canal duplo da ficha *CIB_SR* e esta está conectada a um relé de segurança. Estas configurações podem ser visualizadas na figura seguinte.

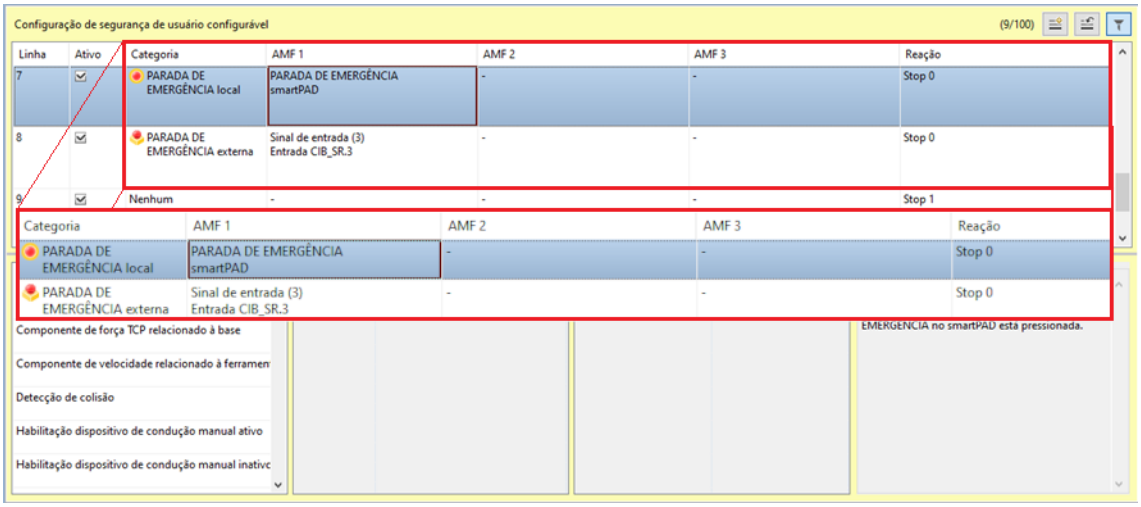


Figura 76 - Representação das primeiras duas linhas de configurações de segurança. Fonte: Próprio

A terceira função de segurança tem como objetivo de proteger o operador quando este necessita de aceder à linha. Esta deve parar o robô sempre que este esteja em automático e a porta que existe na 'zona de evacuação' da Figura 62 da página 100, for aberta. Esta porta está equipada com um fecho de segurança de canal duplo, sendo que este circuito alimenta um relé de segurança. Esse relé de segurança tem ligado a si mais uma entrada CIB_SR do robô, pelo que quando o fecho de segurança da porta de acesso à linha é aberto o relé de segurança para de ser alimentado nas suas bobines e por sua vez os seus contactos abrem, levando a que a entrada CIB_SR do robô fique sem os 24V, violando assim o AMF. A representação desta configuração é apresentada na figura seguinte.

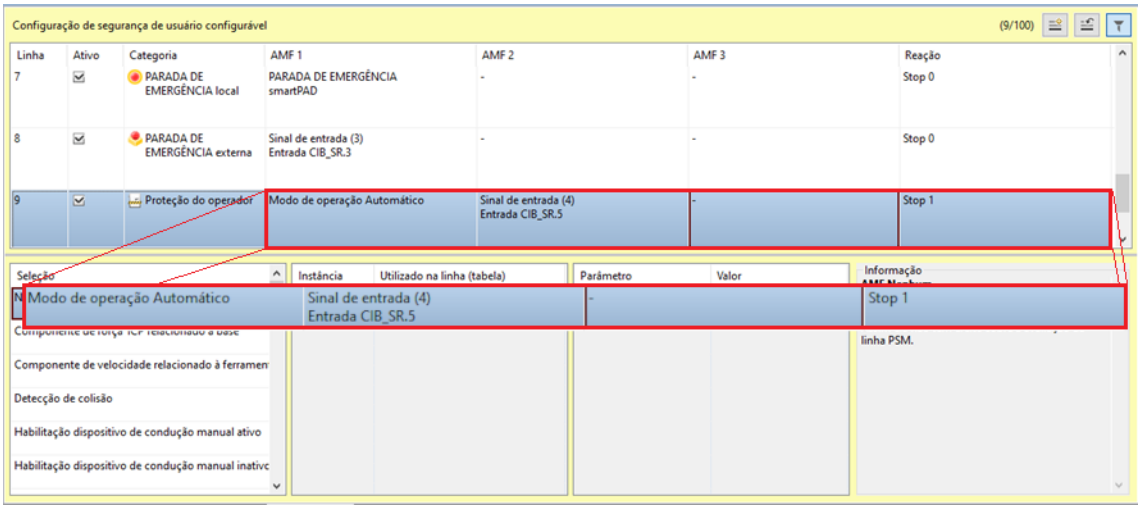


Figura 77 - Configuração de segurança do robô. Fonte: Próprio

A próxima função de configuração é mais complicada e pretende-se que quando o robô se encontra em um modo de alta velocidade (modo automático ou modo manual com velocidade nominal), o robô apenas se possa deslocar dentro da área verde definida na Figura 71 da página 107. Esta função vai ser composta por três AMF, sendo que a primeira delas é a AMF Modo de operação com alta velocidade, o que o *Sunrise Workbench* afirma acerca disto é o seguinte:

- **AMF Modo de operação com alta velocidade:** Esta AMF está violada quando está ativo um modo de operação, no qual a velocidade não está reduzida (T2, AUT). Em todos os demais modos de operação ela não está violada. Se o modo de operação atual for inválido, a AMF está violada.

Sendo que já conseguimos uma AMF para controlar a posição do robô consoante o seu modo de funcionamento, temos agora de construir uma gaiola virtual para que o robô não possa sair de dentro desta. Para isto serão usadas duas AMF Monitoramento do espaço de trabalho cartesiano. O *Sunrise Workbench* descreve esta AMF da seguinte forma:

- **AMF Monitoramento do espaço de trabalho cartesiano:** A AMF está violada quando a estrutura monitorada da cinemática monitorada não se encontra totalmente dentro do espaço de trabalho configurado.
    - Parâmetros: Cinemática monitorada: O parâmetro determina qual cinemática deve ser monitorada.
    - Estrutura monitorada: Estrutura que é monitorada por esta AMF.
        - X: Origem do espaço de trabalho (coordenada X) Área: [-1000000,1000000] mm.
        - Y: Origem do espaço de trabalho (coordenada Y) Área: [-1000000,1000000] mm.
        - Z: Origem do espaço de trabalho (coordenada Z) Área: [-1000000,1000000] mm.
        - A: Rotação do espaço de trabalho em torno do eixo Z Área: [0,359]°.
        - B: Rotação do espaço de trabalho em torno do eixo Y Área: [0,359]°.
        - C: Rotação do espaço de trabalho em torno do eixo X Área: [0,359]°.
        - Comprimento: Comprimento do espaço de trabalho (ao longo do eixo X) Área: [0,1000000] mm.
        - Largura: Largura do espaço de trabalho (ao longo do eixo Y) Área: [0,1000000] mm.
        - Altura: Altura do espaço de trabalho (ao longo do eixo Z) Área: [0,1000000] mm.

Desta forma, esta AMF é parametrizada e permite-nos construir a nossa gaiola virtual que define a área onde o nosso robô pode trabalhar, assim como demonstrada na Figura 78.

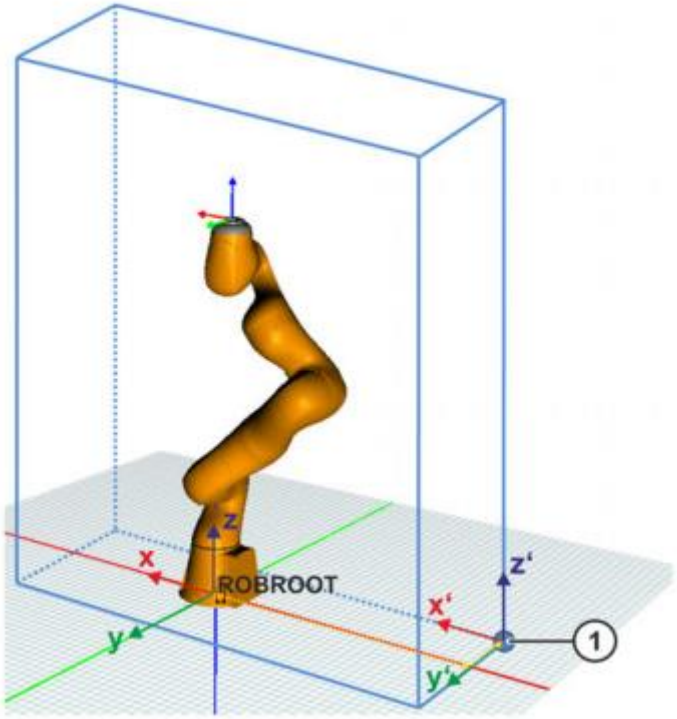


Figura 78 - Representação da gaiola virtual. Fonte: (KUKA AG, 2016, p. 252)

Desta forma criando duas gaiolas virtuais é possível cobrir a nossa área de trabalho pretendida, sendo que a primeira gaiola definida é apresentada na figura abaixo à esquerda sendo representado pela zona a verde e a segunda gaiola é definida pelo segundo paralelepípedo a verde na figura abaixo à direita.

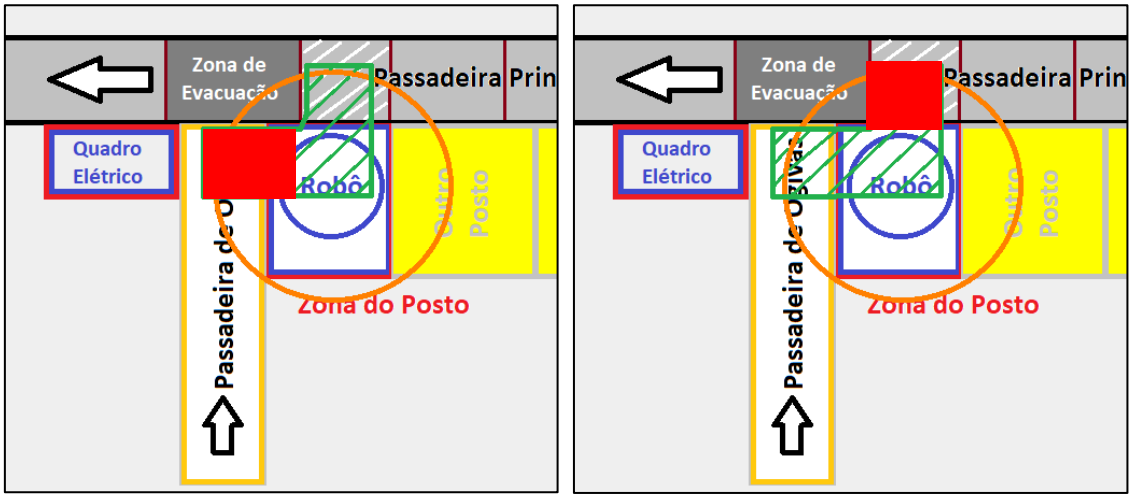


Figura 79 - Representação das áreas definidas por cada gaiola. Fonte: Própria

Desta forma e colocando estas duas gaiolas virtuais em série é possível cumprir eficazmente a segurança do robô e a segurança do operador quando este se encontra no seu posto, restringindo a área de trabalho do sistema robótico. Se o operador, em modo colaborativo, pretender utilizar o robô fora das suas limitações virtuais de segurança, o sistema ativa a sua reação, por ter ultrapassado a sua designada área de trabalho virtual. A apresentação da Figura 80 demonstra os possíveis casos para a gaiola.

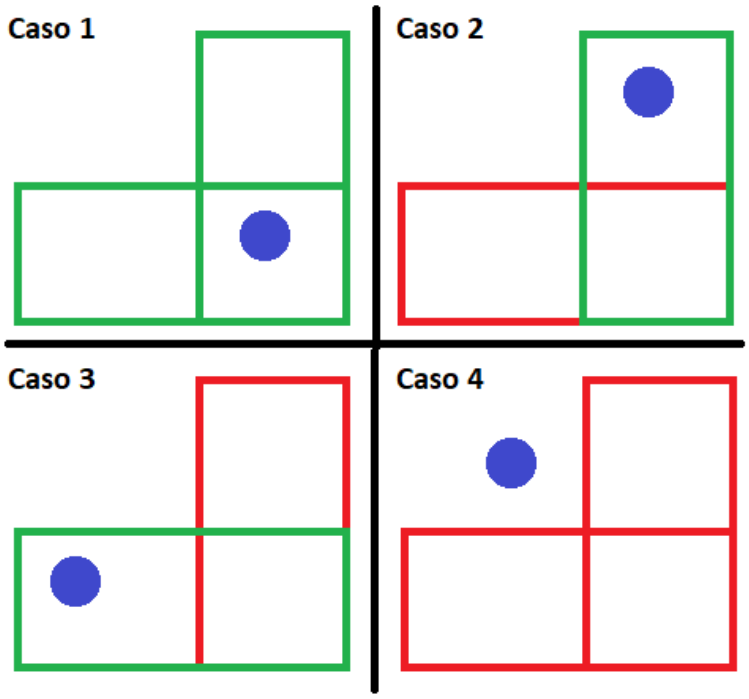


Figura 80 - Diferentes casos possíveis para o estado das gaiolas virtuais, sendo cada paralelepípedo uma gaiola e a bola a azul representa a posição da garra do robô. Fonte: Própria

Estes dados podem ser analisados na seguinte tabela de verdade, sendo que o 1 representa uma gaiola não violada e o 0 uma gaiola violada. O paralelepípedo na horizontal é a gaiola 1, o paralelepípedo na vertical a gaiola 2.

	Gaiola 1	**Gaiola 2**	**Resultado**
Caso 1	1	1	1
Caso 2	0	1	1
Caso 3	1	0	1
Caso 4	0	0	0

Enquanto a função de segurança anterior era para definir as áreas por onde o robô se podia movimentar, esta última configuração é para prevenir acidentes caso alguém invada a área de trabalho do robô. Para isto foi usada uma nova linha configurado apenas com uma AMF, sendo esta a AMF Deteção de colisão. O *Sunrise Workbench* descreve esta AMF da seguinte forma.

- **AMF Deteção de colisão:** A AMF está violada quando o torque externo de pelo menos um eixo da cinemática monitorada ultrapassa o valor limite configurado.

O valor desta AMF pode ser configurado entre a gama de 1 e 30Nm, sendo que esta com 20Nm é suficiente mente sensível para que parar quando algo o retira da sua trajetória sem grande esforço. Esta força foi testada com uma folha de papel em cima da palete das ogivas e o robô ao embater no papel sentiu o esforço e parou, sem rasgar o mesmo. Existe apenas é a necessidade de colocar um bloco quando o robô

transporta a palete, que adiciona o peso da palete à sua garra, de outra forma o robô ao transportar a palete deteta como uma colisão pela força exercida pelo peso da palete. Na figura seguinte são apresentadas as configurações das duas últimas linhas.

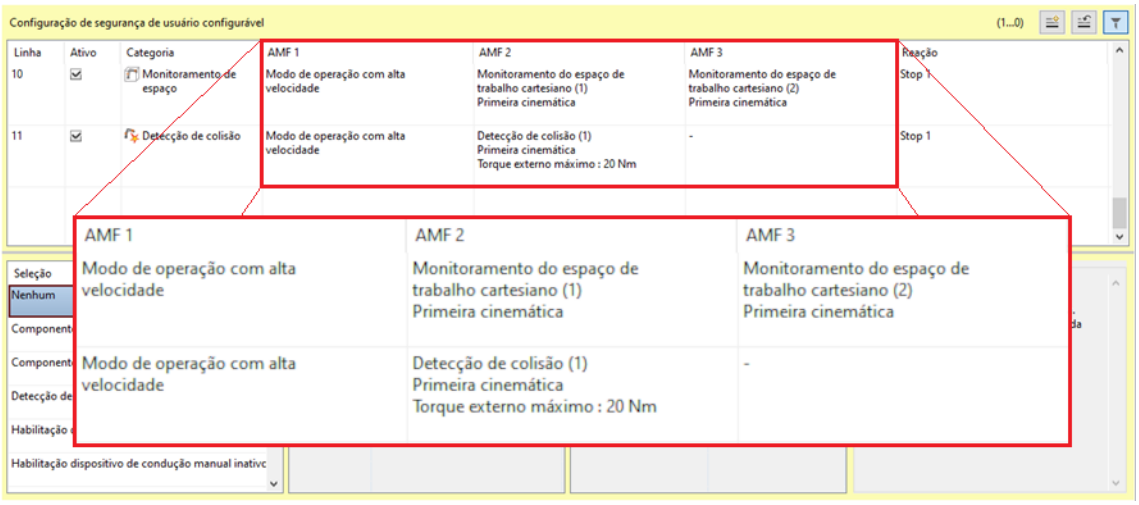


Figura 81 - Configurações de segurança do robô. Fonte: Própria

O robô tem a capacidade de comunicar por *PROFINET*, sendo que é desta forma que este comunica com o PLC. Contudo o PLC não comunica *PROFINET*, mas contem uma carta de *MODBUS TCP/IP*. Para que ambos sejam capazes de comunicar um com o outro, é necessário que existe uma *gateway* entre os dois, como apresentado na Figura 82.

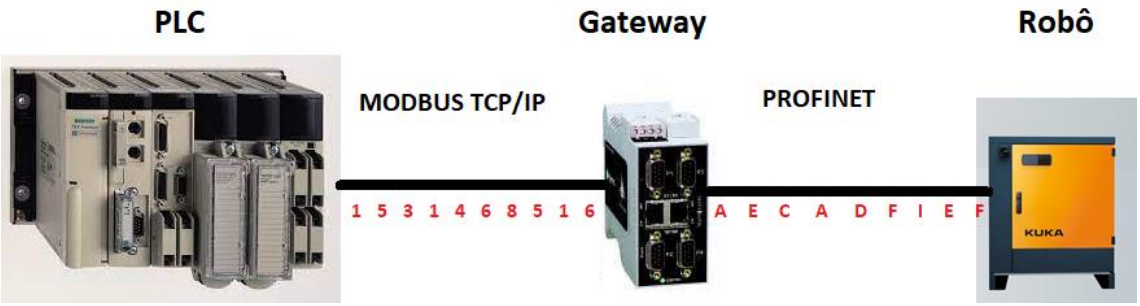


Figura 82 - Demonstração da interligação do PLC com o robô e vice-versa. Fonte: Própria

Desta forma, a *gateway* é responsável por traduzir os dados enviados pelo PLC no protocolo *MODBUS* para *PROFINET* de forma que o robô os compreenda. Da mesma forma, a *gateway* é responsável por converter os pacotes *PROFINET* enviados pelo robô e convertê-los para *MODBUS* para que o PLC os compreenda.

3.2.1.5 – PLC

O PLC usado neste projeto é um *Schneider/Telemecanique TSX 573623A* e é o cérebro das operações. Este faz a gestão e controlo de todos os elementos do sistema, ainda que elementos como o robô e o HMI sejam capazes de ser independentes, estes vão agir como uma espécie de arquitetura de mestre/escravo, onde o PLC é o mestre e os restantes elementos são os escravos que apenas executam código quando há ordem do mestre.

Este PLC encontra-se montado em uma *rack* que é alimentada pela fonte de alimentação *Schneider/Telemecanique TSX PSY 1610*, portanto o PLC é alimentado por uma fonte de alimentação externa. Esta fonte de alimentação também alimenta os restantes dispositivos da *rack*.

Figura 83 - Exemplo de uma rack com o PLC utilizado nesta atividade e as suas cartas. Fonte: (AUTOMATE TSXP573623 TELEMECANIQUE SCHNEIDER, s.d.)

O PLC também se encontra ligado a outras cartas de diversos tipos e para diferentes funcionalidades. As cartas estão conectadas ao PLC através da sua *rack* que as segura e garante a conexão da pinagem. Já a parte de configuração e programação das cartas é feita no *software* de programação do PLC, sendo este o *PL7 Pro 4.5*.

No caso do nosso PLC nenhuma carta foi adicionada, sendo que todas as cartas que o PLC contém já estavam previamente instaladas. Na figura seguinte é demonstrado um printscreen do *PL7 Pro* onde este apresenta as cartas colocadas na sua *rack*.

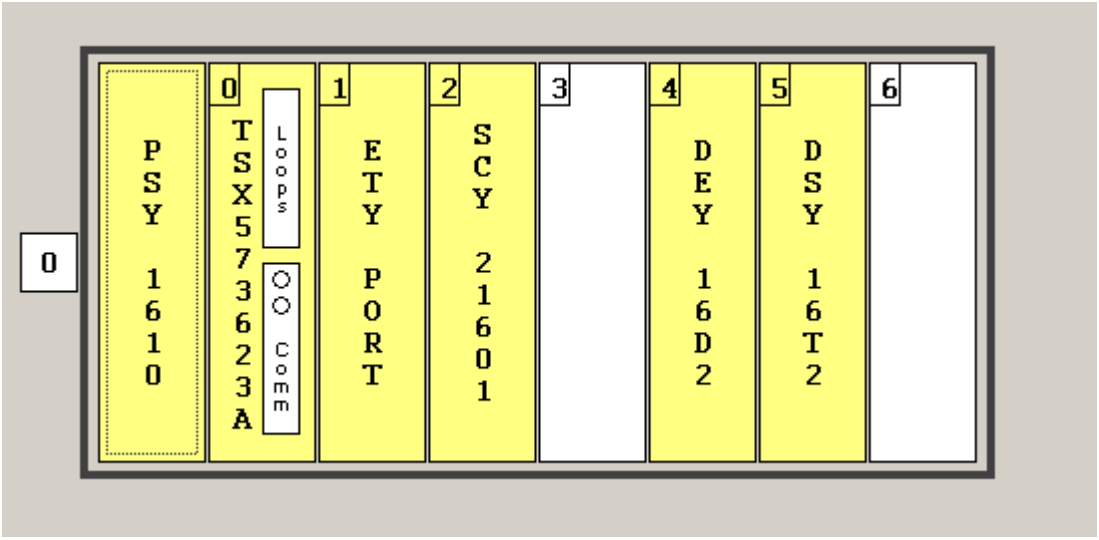


Figura 84 - Configuração da rack do PLC. Fonte: Própria

Já abordámos os dois primeiros *slots* da *rack*, sendo que estes são ocupados respetivamente pela fonte de alimentação e o PLC. Na posição 1 encontramos a carta *ETY PORT*, esta carta permite usar comunicações do tipo *TCP/IP*, sendo esta a possibilidade de usar *unitelway* (protocolo de comunicação proprietário da *Schneider/Telemecanique* que atualmente está descontinuado) e o *MODBUS TPC/IP* no modo de mestre. Esta carta é assim usada para comunicar com outros PLC da linha e comunicar com os seus dispositivos escravos através de *MODBUS*.

Para se configurar esta carta existem duas janelas principais, uma onde são colocadas definições de conexões e o tipo de protocolo. A segunda janela, define a área onde se lê e escreve dos outros dispositivos.

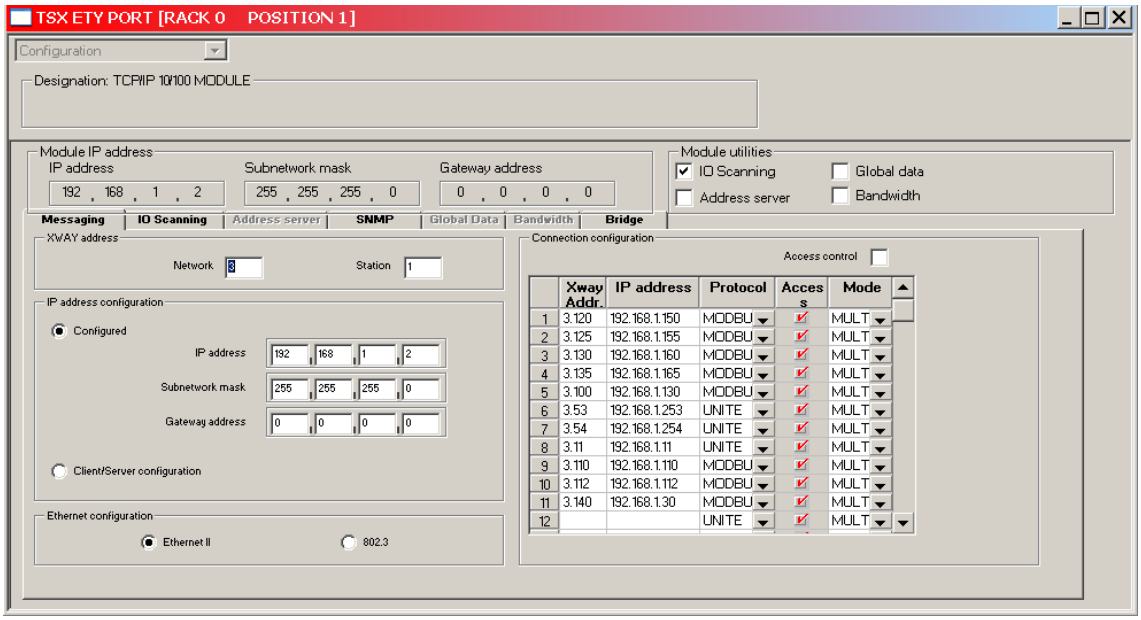


Figura 85 - Primeira página de configurações da carta ETY PORT. Fonte: Própria

A imagem acima refere-se à primeira página, sendo que aqui se configura o *IP* da carta e os endereços dos outros elementos a se comunicar, o protocolo a ser usado e o seu *XWAY Address*. O *XWAY Address* é um método usado pela

Schneider/Telemecanique de identificar os dispositivos por um endereço universal, independentemente do protocolo de comunicação usado. Desta forma, o *XWAY Address* divide-se em *<rede de trabalho (network).estação (station)>,* sendo que dispositivos dentro da mesma rede de trabalho podem comunicar entre si, contactando-se pelo seu número de estação único.

A segunda página refere-se à zona de leitura e escrita dos dados. A figura seguinte apresenta um exemplo desta página.

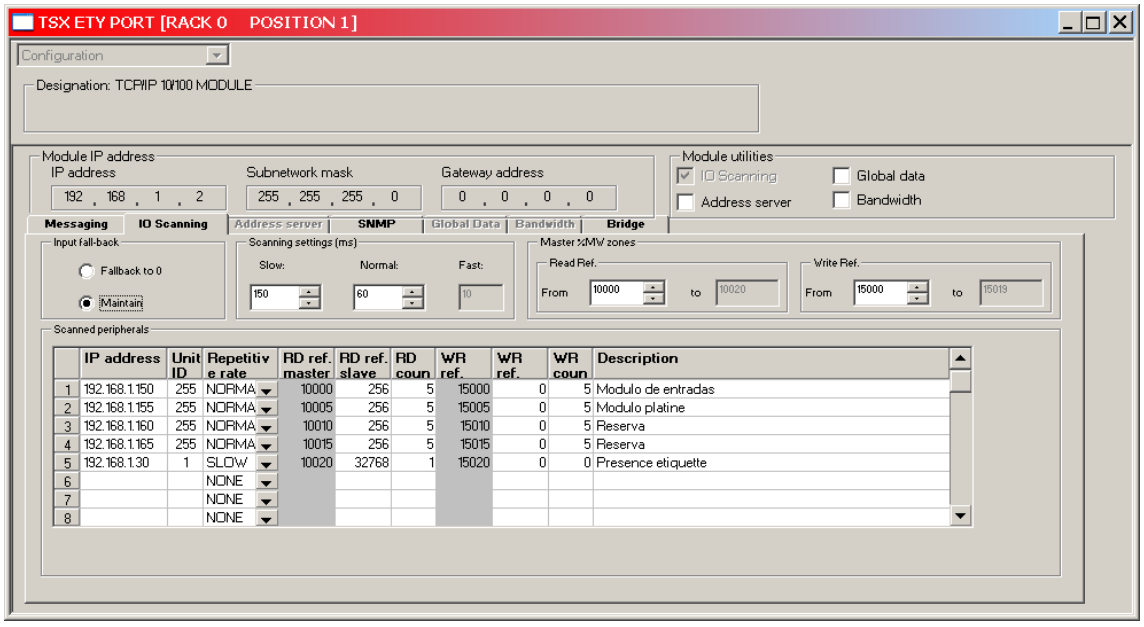


Figura 86 - Segunda página de configurações da carta ETY PORT. Fonte: Própria

Olhando de cima para baixo da esquerda para a direita, entende-se que a primeira configuração é '*Input fall-back*', esta diz respeito ao que acontece aos valores dos dados lidos caso o PLC entre no modo STOP ou existe uma falha na comunicação, sendo possível configurar como '*fallback to 0*' ou '*maintain*'. Será explicado a seguir o que cada uma destas definições faz no caso descrito acima:

- ***Fallback to 0*:** Se este modo estiver selecionado, os valores lidos são todos colocados a zero;
- ***Maintain*:** Se este modo estiver selecionado, os valores lidos mantêm os últimos dados registados.

Neste caso, foi usado o modo de '*maintain*' uma vez que é mais estável. Um exemplo desta estabilidade, é no caso de haver um ciclo de comunicação que falhe, no modo '*maintain*', os dados não são diretamente apagados, enquanto no modo '*fallback to 0*' estes dados são apagados. Caso o modo '*fallback to 0*' esteja selecionado, numa falha de comunicação em que os valores são apagados, saídas que possam depender destes, podem se desligar abruptamente e de forma inesperada, algo que não é desejado na maior parte dos casos.

A próxima configuração a ser feita é o tempo de atualização das entradas e saídas em cada tipo de velocidade, sendo que é possível escrever dados de forma rápida, normal ou lenta.

A próxima configuração é uma das mais importantes, sendo que esta representa a área onde os valores que devem ser enviados e os valores lidos são armazenados. Desta forma existe alguma atenção necessária aquando da sua configuração. A primeira é a necessidade de escolher uma área de variáveis que esteja livre, de forma que não seja escrito por outras funcionalidades não relativas à comunicação. Outra necessidade a se ter atenção é que todos os dados lidos e enviados são do tipo *Word* (relembrando os tipos de dados impostos pelo IEC 61131-3, este representa uma cadeia de 16 bits) que, dentro do *PL7 Pro*, podem ser interpretados como um tipo inteiro com sinal de 16 bits. O grande problema começa quando existe uma troca de informação de valores inteiros entre o PLC e sistemas mais atuais, sendo que quando a troca de dados é apenas de nível binário, este problema não existe.

Para compreender melhor o problema dos dados abordado anteriormente, temos de compreender que a arquitetura dos PLC ainda é bastante antiga, sendo que estes por norma têm uma arquitetura do processador de 16 ou 32 *bits*, enquanto a maioria dos sistemas atuais têm uma arquitetura superior, sendo de 32 ou 64 *bits*. Isto faz com que, por exemplo, o valor de um inteiro que os fabricantes se referem por norma têm 32 *bits* e não os 16 *bits* como no caso de um tipo inteiro/*word* no PLC. Uma forma simples de verificarmos isto é escrevendo um pequeno programa na linguagem *C*, sendo este apresentado no exemplo seguinte.

```c
#include <stdio.h>
#include <limits.h>

//Definir os tipos de variáveis no estilo de um
//PLC independentemente da arquitetura
#if UINT_MAX == 4294967295
typedef short int           Word;
typedef int                 DWord;
typedef long int            LWord;
#else
typedef int                 Word;
typedef long int            DWord;
typedef long int       LWord;
#endif

int main()
{
    printf("Tamanho de um inteiro na arquitetura do meu computador:
           %d Bytes\n", sizeof(int));
     printf("Tamanho de um inteiro na arquitetura de um PLC: %d
           Bytes\n", sizeof(Word));
    return 0;
}

//////////////////////OUTPUT NA CONSOLA//////////////////////////
//Tamanho de um inteiro na arquitetura do meu computador: 4 Bytes
//Tamanho de um inteiro na arquitetura de um PLC: 2 Bytes
```

Bloco de Código 18 - Comparar tamanho de diferentes tipos em diferentes arquiteturas, em linguagem C.

Esta diferença no tamanho de *bytes* que diferentes arquiteturas de processadores têm, pode ser um problema grave caso não se tenha este pormenor em consideração, por exemplo na comunicação entre o PLC e o robô. Isto porque no *PL7 Pro* as *Words* e *DWords* usam o mesmo espaço de memória, o que significa que o endereço da *DWord %MD0* é o mesmo que o das *Words %MW0* e *%MW1*. Isto é mais facilmente compreendido com o auxílio da Tabela 4, onde ao meio da tabela conseguimos ver o endereço das *words*, em cima temos a representação das *dwords* pares incluindo o zero e em baixo temos as *dwords* ímpares.

Tabela 4 - Cruzamento de dados de words e dwords no PL7 Pro 4.5.

Endereço de DWords Pares	%MD0		%MD2	
Valor das DWords Pares	1		131075	
Valor das Words em Binário	0000_0000_0000_0000	0000_0000_0000_0001	0000_0000_0000_0010	0000_0000_0000_0011
Valores das Words	0	1	2	3
Representação dos MSB e LSB para DWords Pares	MSB	LSB	MSB	LSB
Endereços das Words	%MW0	%MW1	%MW2	%MW3
Representação dos MSB e LSB para DWords Impares		MSB	LSB	MSB
Valores das Words	0	1	2	3
Valor das Words em Binário	0000_0000_0000_0000	0000_0000_0000_0001	0000_0000_0000_0010	0000_0000_0000_0011
Valor das Dwords Impares		65538		196
Endereço de Dwords Impares		%MD1		%M

Olhando à tabela, o valor da *%MD0* é 1, isto porque a *word %MW0*, que representa os bits mais significativos da *%MD0*, está a 0 e o a *word %MW1* está a 1. Desta forma, quando juntamos as duas *words* obtemos a seguinte sequencia de *bits* 0000_0000_0000_0000_0000_0000_0000_0001, sendo que o resultado desta cadeia de *bits* em decimal é 1.

No caso da *%MD1* (na parte de baixo da tabela) esta contém o valor de 65538, isto porque é constituído pelo *word %MW1* para os bits mais significativos com o valor 1 e *%MW2* para os bits menos significativos com o valor 2, quando as juntamos para construir a *dword* obtemos a seguinte cadeia de *bits* 0000_0000_0000_0001_0000_0000_0000_0010, sendo que este representa o valor 65538 em decimal.

Deste modo, quando enviamos um valor ao robô, temos obrigatoriamente de usar duas *words*, ainda que a que represente os valores mais significativos muito possivelmente fique a zero. Existe outro método de solucionar isto, do lado do recetor ler apenas o tamanho de uma *word*, no entanto muitos dispositivos não é possível realizar isto, sendo o robô um destes casos. Ainda existem outras soluções do lado do recetor, por exemplo, fazer um *cast* para um tipo que represente um inteiro de 16 *bits* no sistema recetor, deste modo os *bits* mais significativos são destacados. De qualquer modo, é importante ter-se isto em consideração sempre que vamos trocar dados com outro tipo de arquitetura.

Por fim, a parte de baixo da segunda página de configurações, indica o *IP* do dispositivo onde se é para ler/escrever, seguindo pelo seu *ID*, sendo este o endereço de um escravo do MODBUS. A próxima coluna apresenta a velocidade de atualização dos dados. Por fim temos três colunas respetivas à área onde é para se ler e três colunas relativas à área onde é para se escrever, sendo que em ambos casos funciona da mesma forma.

A primeira indica onde os dados ficam armazenados e acessíveis ao PLC. Consoante se é para escrita ou para leitura, é onde os dados são lidos ou escritos, respetivamente. A próxima coluna representa o primeiro endereço onde se é para ler

no escravo ou o primeiro endereço onde é para se escrever no escravo. A última coluna representa o número de *words* que são para se ler ou para se escrever respetivamente.

Usando esta carta, o PLC comunica com as suas entradas e saídas a módulos que se comunicam por *MODBUS*.

A próxima carta, *SCY 21601*, representa uma carta que permite comunicar pelo protocolo *unitelway* através de uma porta RS485, sendo que esta porta é usada para comunicar com o HMI.

As restantes cartas são de entradas e saídas discretas do PLC.

3.2.1.6 – HMI

O HMI usado foi um *Schneider HMIGK2310*, sendo este um HMI que fica no meio entre os dois apresentados no estado de arte, não é o mais primitivo, já sendo possível criar ecrãs personalizáveis e usar o seu ecrã tátil, mas também não chega a ter as capacidades do HMI da *Beijer*.

Figura 87 – Schneider HMIGK2310. Fonte: (Schneider Electric HMIGK2310, s.d.)

Como já foi referido, o HMI comunica diretamente com o PLC por uma porta RS485 através do protocolo *unitelway*. Este, permite aos operadores colocarem a máquina em modo automático ou manual, acionar os atuadores de forma manual, verificar a etiqueta da palete, etc.

Não será especificado mais nada acerca do HMI porque não houve intervenção com o mesmo, uma vez que o projeto que se encontrava carregado nele cumpria as necessidades de funcionamento do posto e dos controlos manuais, mesmo depois das intervenções do posto terem sido realizadas.

3.2.2 - Apresentação dos problemas na gestão das paletes de ogivas

Como referido, anteriormente o transportador de ogivas era abastecido por paletes através de um AGV, contudo, por mudanças na linha, os AGV alteraram as suas rotas e estes já não abastecem este transportador de ogivas. Junto com estas mudanças, o cliente também quis colocar estas ogivas com o robô do nosso posto, trabalho que antes era realizado manualmente por um operador.

Existem dois problemas que surgem com a utilização do robô. O primeiro já foi referido anteriormente, é qual o destino das paletes das ogivas assim que estas não contiverem mais ogivas. O segundo é um problema relativo ao tamanho do robô, que como se pode compreender pela Figura 79, a sua garra não alcança todo o comprimento da palete.

Vamos abordar a solução encontrada para estes dois problemas nos próximos subcapítulos.

3.2.2.1 - Processo de troca de paletes no transportador de ogivas

Inicialmente, a ideia do cliente, relativamente ao que acontecia à palete de ogivas quando estas se encontravam vazias, era o robô pegar nelas e colocá-las na palete da caixa de velocidades, ficando estas de pé pelas suas pernas usando uns buracos existentes na palete da caixa de velocidades já desenvolvidos para este âmbito. Contudo, chegou-se à conclusão de que pelas dimensões da palete de ogivas não existia muito espaço ao operador responsável para retirar a palete de ogivas. Outro problema era o peso da palete e a má posição do operador para retirar a mesma.

Após várias soluções postas em cima da mesa, chegou-se a uma solução ótima. Esta permitia ao operador não carregar a palete manualmente, nunca haver escassez de ogivas para o robô e, durante a nossa intervenção, uma baixa intervenção na estrutura da linha ou do posto a nível físico e mecânico.

Esta solução consiste em funcionar apenas com duas paletes de ogivas, perfazendo um total de 40 ogivas (20 por palete) e gerir corretamente o transportador, fazendo este avançar ou recuar uma palete enquanto a outra se encontrava sem contacto com o transportador, agarrada pelo robô ou suspenso pelo indexador. Isto foi possível porque o transportador era de correias, que permite que o transportador se mova para os dois lados, e o seu atuador era um motor trifásico.

Implementando esta solução a única mudança física necessária de se fazer foi a alteração do relé que ativava a rotação da passadeira para um conjunto de relés inversores e mecanicamente encravados, sendo que o primeiro relé faz rodar o

transportador no sentido normal e o segundo faz rodar no sentido inverso. Para o segundo relé mover a passadeira no sentido inverso, foram invertidas duas fases na ligação trifásica relativamente ao primeiro relé. Deste modo é importante usar relés mecanicamente encravados, o que permite que apenas um dos relés esteja a alimentar o motor trifásico, prevenindo assim um curto-circuito de fases no caso de acidentalmente os dois relês serem atuados em simultâneo. Por fim, a saída que ativava o movimento normal da passadeira foi ligada no primeiro relé e foi usada uma nova saída disponível para esta se conectar ao segundo relé que permite ao PLC mover o transportador no sentido inverso.

Agora que foi configurado a parte física da passadeira é necessário configurar o PLC para que este funcione corretamente consoante os estados das paletes. Para isto vamos olhar inicialmente à linha das ogivas de forma que se possa compreender onde se encontram os *stoppers*, os sensores e o indexador.

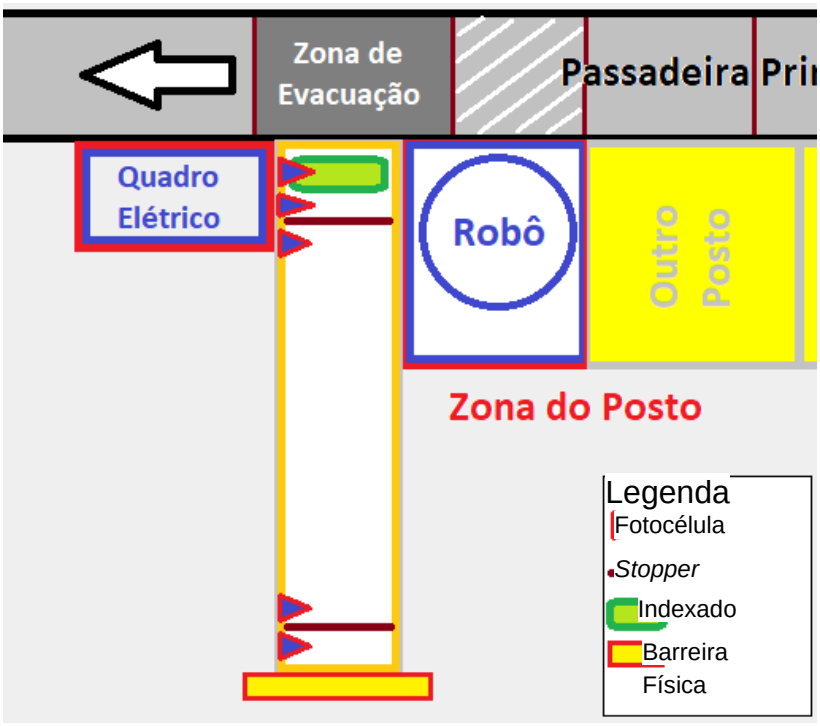
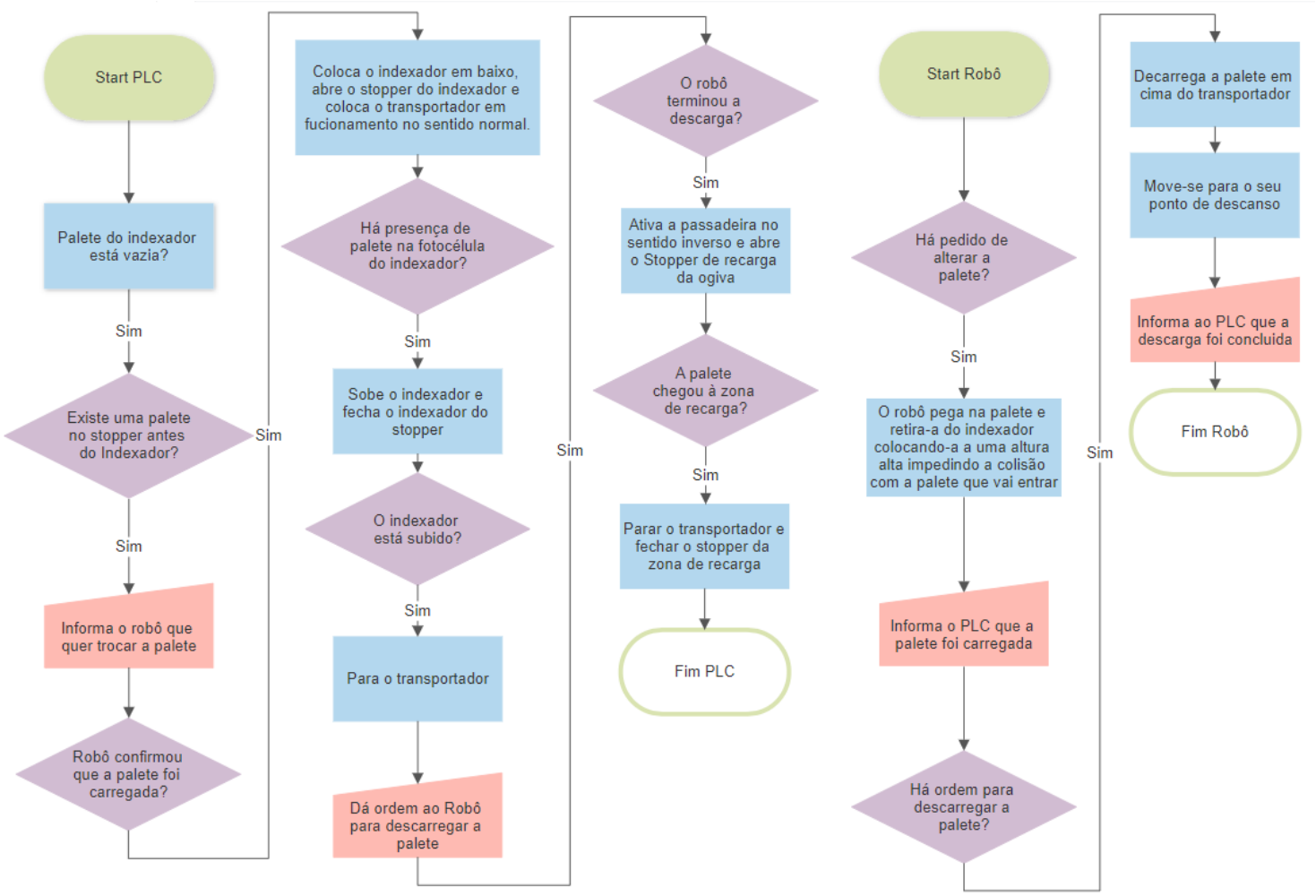


Figura 88 - Esquema mais detalhado do transportador de ogivas. Fonte: Própria

Analisando a figura acima, consegue-se perceber que esta conta com cinco fotocélulas, dois *stoppers*, um indexador e uma barreira física. Começando do fim para o início, a barreira física foi colocada para que não fosse possível a palete de ogivas cair no caso de o transportador não parar durante o movimento inverso. Os restantes elementos são todos geridos pelo PLC.

A existência de uma fotocélula de cada lado de um *stopper* permite que esta funcione como presença de palete antes do *stopper* e de 'a palete passou o *stopper*', sendo que esta sabe se já pode abrir e se já pode fechar o *stopper*, respetivamente.

Para que se compreenda melhor vamos apresentar o modo de funcionamento desta gestão, sendo apresentado no Fluxograma 4, do lado esquerdo, o funcionamento do PLC e, do lado direito, o funcionamento para o robô. Em primeira fase vamos apresentar o cenário de como funciona a troca de paletes do indexador.



Fluxograma 4 - Descrição do programa do PLC e do robô no processo de troca de paletes de ogivas.

Olhando ao fluxograma, pode compreender-se que é um processo complexo. A dificuldade é acrescida pela necessidade de sincronismo, que é feita com a troca de informação entre o PLC e o robô. A forma mais simples de acompanhar a leitura do fluxograma é começar pelo início do PLC, que é o mestre, e quando encontrar um bloco de comunicação (os que têm cor de salmão) trocar para o fluxo do robô. Quando for encontrado um bloco de comunicação do lado do robô volta-se ao fluxo do PLC a partir do ponto onde foi abandonado anteriormente.

Analisado o fluxograma é possível compreender que existe uma palete vazia no indexador, o PLC pede ao robô para que este a tire, o robô tira a palete do indexador, coloca-se numa posição segura e informa o PLC que carregou a palete. O PLC ao receber esta informação ativa o transportador no sentido normal para colocar a palete previamente carregada no indexador. Após a palete carregada ser indexada,

o PLC para o movimento do transportador e informa o robô que este pode descarregar a palete vazia. O robô quando recebe a informação, faz um movimento de forma a colocar a palete em cima da passadeira, abre a pinça, desloca-se para uma posição segura (já sem a palete) e informa o PLC que a descarga foi realizada. Finalmente o PLC ativa o transportador no sentido inverso até que este chega à sua zona de carga.

Entendido o procedimento de troca de paletes de ogivas, existe apenas a necessidade de entender como funciona o processo de recarregamento da palete de ogivas e o processo de colocar as ogivas nas caixas de velocidades.

O processo de recarga da palete de ogivas é muito simples, sendo que é o operador responsável por retirar as ogivas que vai ao início do transportador recarregar a palete, relembrando que as ogivas são reutilizáveis. Quando a palete se encontra carregada, este pressiona um botão adicionado no início do transportador que informa o PLC de que a palete foi carregada, abrindo o *stopper* da zona de recarga e alimentando o transportador para este se movimente no sentido normal. Quando a palete chega à fotocélula do *stopper* do indexador, o *stopper* da zona de recarga fecha e o transportador para.

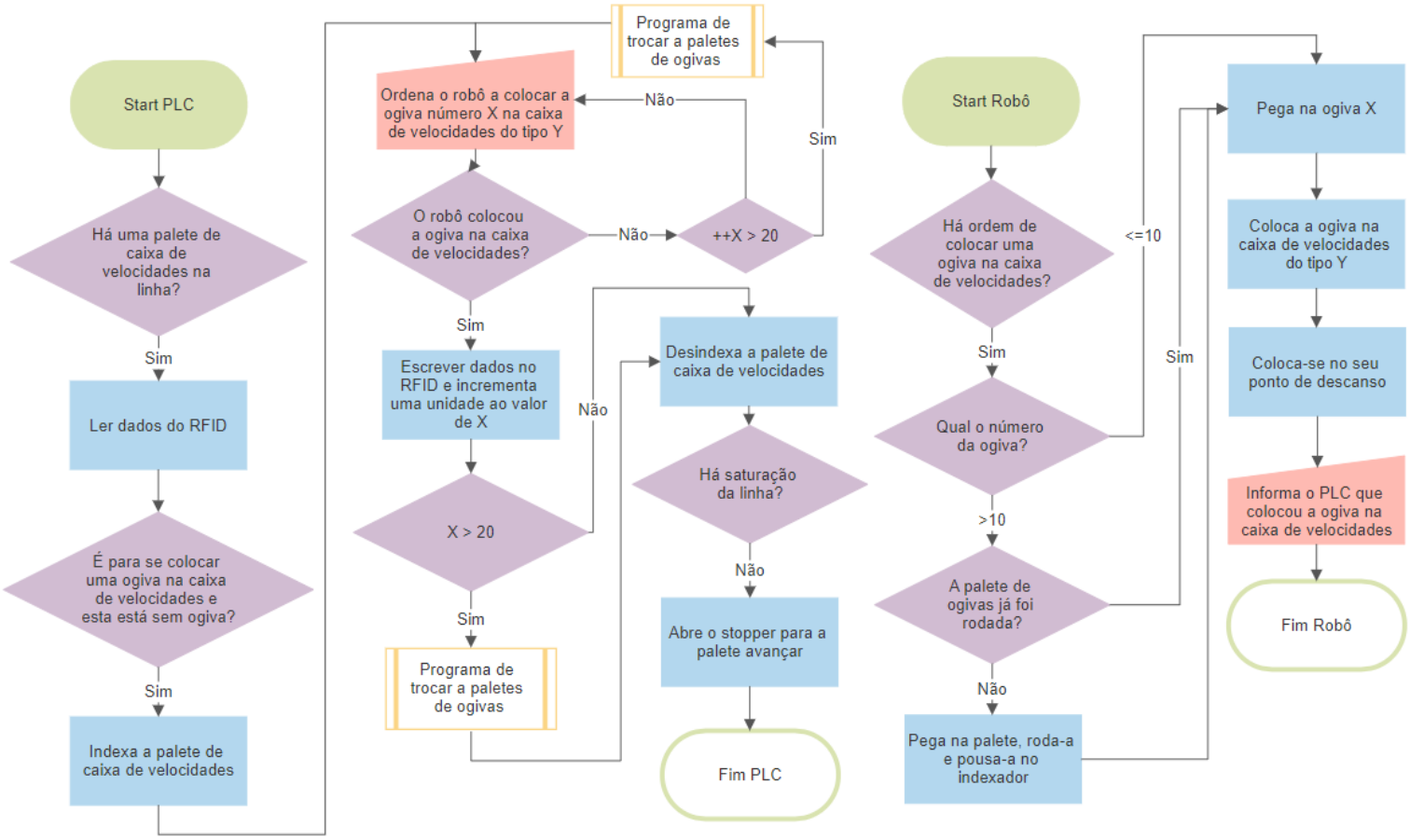
Quanto ao processo de colocar ogivas na caixa de velocidade, este será abordado no próximo subcapítulo.

3.2.2.2 – Processo de colocar as ogivas na caixa de velocidades

O processo de colocar as ogivas na caixa de velocidades era relativamente simples se o robô tivesse o alcance de chegar a toda a sua palete. Contudo, isto não sucede, pelo que é necessário fazer uma rotação à palete para que seja possível utilizar as ogivas fora do alcance do robô.

Desta forma, o robô tem um programa consoante o número de ogivas que este vai pegar, esta gestão do programa é realizada pelo PLC, que é responsável por gerir o número de ogiva que está a ser carregado, assim como dar as ordens de pegar numa ogiva, deixar a ogiva na caixa de velocidades, rodar a palete, etc.

Apresenta-se um fluxograma que descreve os fluxos dos programas, do lado do PLC e do lado do robô.



Fluxograma 5 - Explicação gráfica de como funciona a gestão e colocação de uma ogiva na caixa de velocidades.

Analisado o fluxograma pode-se salientar, mais uma vez, que o PLC é o mestre, dando ordens, e o robô é o escravo, informando apenas que as ordens foram realizadas. Desta forma, o escravo não inicia nenhuma tarefa sem que esta tenha sido ordenada.

O processo inicia com a chegada de uma palete à linha. Se esta for validada para trabalhar e não tiver ogiva (o controlo de presença de ogiva é realizado por uma fotocélula instalada para este efeito, dado que no modo degradado do posto, um modo que quando ativo inibe o funcionamento do posto, é o operador que realiza esta tarefa) é enviada uma ordem para o robô. Esta ordem leva a informação do número de ogiva que é para se carregar, sendo o PLC que gere isso, e o tipo de caixa de velocidades, uma vez que alguns modelos são alguns milímetros mais baixos. As ogivas são numeradas consoante a sua posição na palete, como pode ser visualizado na Figura 89.

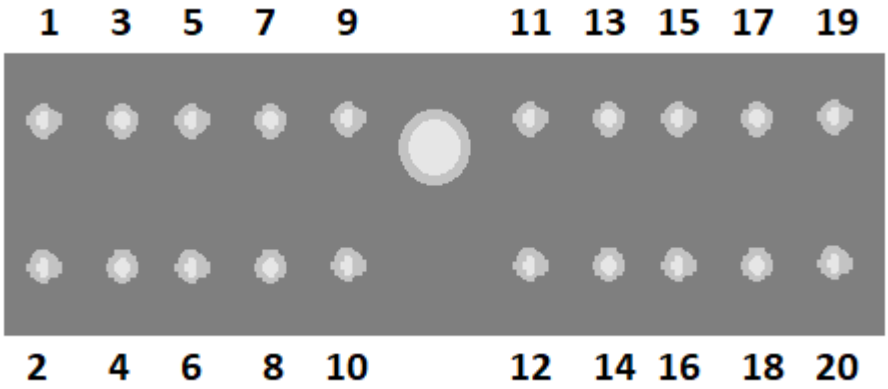


Figura 89 - Numeração das ogivas consoante a sua posição na palete. Fonte: Própria

Recebida a ordem, do lado do robô, este avalia o número da ogiva que é para pegar, uma vez que as paletes entram sempre da mesma forma no indexador o robô apenas necessita de guardar a informação se já rodou a palete ou não. Se o número for inferior a 10, inclusive, o robô desloca-se aos pontos feitos para aquela ogiva e pega na ogiva, por fim lança outro programa para colocar a ogiva na caixa de velocidades. Neste programa avalia o tipo de caixa, se esta corresponder a uma 'caixa baixa' o robô move-se até um ponto antes de abrir a pinça, se não for move-se a um ponto relativamente mais alto ao anterior antes de abrir a pinça. Por fim o robô move-se para o seu ponto de descanso de forma a estar num ponto seguro independente do próximo programa lançado. Se o número da ogiva for superior a 10, exclusive, este verifica se a palete já foi rodada, se não foi lança um programa que roda a palete antes de realizar o ciclo anteriormente descrito, se já tiver a palete rodada, este realiza de imediato o ciclo de pegar na ogiva pretendida e a colocar na caixa de velocidades.

Quando o valor da ogiva é superior a 20, exclusive, o PLC lança o programa de trocar paletes descrito no subcapítulo anterior.

3.2.3 – Conclusões

Com esta atividade pode-se concluir que é possível interligar equipamento mais antigo, como o PLC, a equipamento mais moderno, como o robô colaborativo, e fazer uma aplicação funcional e que oferece flexibilidade à linha, como pretendido pela indústria 4.0. Outra vantagem da reutilização desta tecnologia antiga em conjunto com uma mais recente, é a diminuição de custos ao cliente e a diminuição da pegada ecológica.

A otimização de processos é um elemento também muito importante para o cliente, que muitas vezes espera que as ideias venham do lado de cá, das empresas que realizam os serviços de automação industrial e robótica, sendo sempre importante pensar na melhor forma de automatizar e rentabilizar um processo. Este posto é um bom exemplo disso, sendo que com poucas alterações foi possível poupar dinheiro

ao cliente e satisfazer o mesmo, algo que o cliente pensava que não seria possível sem uma grande intervenção física no sistema e com muito dinheiro envolvido.

Por fim, o benefício de utilizar um robô colaborativo em uma aplicação que não necessariamente exige a colaboração, ainda que esta opção está disponível caso o cliente pretenda, mas sim em coexistência e cooperação (como se este tambem fosse um operador) sem que exista uma necessidade física de barreiras para garantir a segurança de tudo e todos. Por este motivo, torna esta aplicação uma pioneira na forma como, acredito eu, no futuro se irão usar os robôs, sem que estes estejam separados fisicamente dos humanos nem a realizar trabalho colaborativo. Um passo que se vê nesse sentido é a aplicação desenvolvida pela *KUKA* conhecida como *AIRSKIN*.

CAPÍTULO 4: CONSIDERAÇÕES FINAIS

Neste último capítulo, apresentam-se as conclusões teóricas e as limitações encontradas no decorrer do relatório de estágio.

4.1 – Conclusões

A automação industrial e robótica é uma área extensa, pelo que exige uma contínua aprendizagem, independentemente do grau de experiência, tanto que todas estas diferentes tecnologias têm a sua curva de aprendizagem. O desafio nesta área deve-se à necessidade de compreender inúmeros processos, sensores, atuadores, esquemas elétricos, materiais, dispositivos, linguagens de programação, protocolos de comunicação, tecnologias, etc.

Ainda que o foco deste relatório de estágio tenha sido mais a área de automação e robótica, é inevitável que para se ser um bom engenheiro de automação industrial e robótica seja necessário o domínio e conhecimento de diversas áreas além da eletrotécnica.

Durante o período de estágio profissional senti que de algum modo me encaixei na área e fui bem-sucedido, sendo que após o estágio curricular foi-me proposto um estágio profissional na mesma empresa que me permitirá continuar a trabalhar na área de automação e robótica, continuando as atividades começadas no estágio curricular e começando novas.

Por fim, cada vez mais vai ser necessário os engenheiros de automação industrial e robótica conhecerem e dominarem uma linguagem de programação que seja flexível, como por exemplo o C#, uma vez que a indústria 4.0 cada vez mais vai exigir soluções que possam caber no nosso bolso e com a capacidade de comunicar connosco e com o sistema independentemente da distância a que estes se encontrem.

4.2 – Limitações do relatório de estágio

Apresentam-se ainda algumas limitações que se consideram relevantes. A primeira limitação é de natureza legal, de proteção de dados, o que impossibilita de partilha de diversos conteúdos. A segunda limitação é a dimensão exigida do relatório de estágio. Como última limitação aponta-se a complexidade de apresentar convenientemente todos os conceitos abordados nas atividades do estágio.

REFERÊNCIAS

(6 de setembro de 2021). Obtido de Google Imagens: https://www.google.com/search?q=TELEMECANIQUE+SCHNEIDER+XBT+E013010&tbm=isch&ved=2ahUKEwjJlf3pxOryAhUD4RoKHTvcDxEQ2-cCegQIABAA&oq=TELEMECANIQUE+SCHNEIDER+XBT+E013010&gs_lcp=CgNpbWcQA1CtzgFYrc4BYOPSAWgAcAB4AIABrAGIAawBkgEDMC4xmAEAoAEBqgELZ3dzLXdpei1pbWfAAQ

(2021). Obtido de Google Trends: https://trends.google.pt/trends/?geo=PT

Abreu, P. (2001). *Robótica Industrial: Fundamentos da Robótica e Aspectos Tecnológicos da Robótica.* Obtido de FEUP: https://paginas.fe.up.pt/~aml/maic_files/introd.pdf

Alcantara, M. (31 de março de 2020). *Medir posição com encoders*. Obtido de Mokka-Sensors: https://www.mokka-sensors.com.br/2020/03/31/medir-posicao-com-encoders/

Antonsen, T. M. (2020). *PLC Controls with Structured Text (ST), V3: IEC 61131-3 and best practice ST programming* (3ª ed.). Randers, Dinamarca: Books on Demand.

Aprendendo eletrica. (2019). *O que é um relé temporizador e como funciona?* Obtido de Aprendendo Eletrica: https://aprendendoeletrica.com/o-que-sao-reles-temporizadores/

AUTOMATE TSXP573623 TELEMECANIQUE SCHNEIDER. (s.d.). Obtido em 9 de novembro de 2021, de Eletro-Standard: https://www.electrostandard-electronique.com/automate-tsxp573623-telemecanique-schneider-b989.html

Batente bloco industrial de Palete - Imagem em Alta Resolução. (25 de maio de 2015). Obtido de IStock Photo: https://www.istockphoto.com/br/foto/batente-bloco-industrial-de-palete-gm474973862-65021121

Beckhoff. (29 de julho de 2021). *TC1200 | TwinCAT 3 PLC*. Obtido de Beckhoff: https://www.beckhoff.com/en-us/products/automation/twincat/tc1xxx-twincat-3-base/tc1200.html

Bogner, E., Voelklein, T., Schroedel, O., & Franke, J. (2016). Study based analysis on the current digitalization degree in the manufacturing industry in Germany. *Study based analysis on the current digitalization degree in the manufacturing industry in Germany*, pp. 14-19.

Bongas. (20 de outubro de 2021). *VÁLVULA SOLENÓIDE: TIPOS, FUNCIONAMENTO E APLICAÇÕES*. Obtido de Bongas: https://bongas.com.br/valvula-solenoide-tipos-funcionamento-e-aplicacoes/#2

Branco, A. L. (6 de fevereiro de 2012). *Revoluções industriais - Primeira, segunda e terceira revoluções*. Obtido de UOL Educação: https://educacao.uol.com.br/disciplinas/geografia/revolucoes-industriais-primeira-segunda-e-terceira-revolucoes.htmhttps://educacao.uol.com.br/disciplinas/geografia/revolucoes-industriais-primeira-segunda-e-terceira-revolucoes.htm

CAIXAS DE VELOCIDADES. (s.d.). Obtido em 28 de 10 de 2021, de Motor Portugal: http://www.motorportugal.com/caixas_de_velocidades.html

Chaíça, I. (2 de junho de 2020). *Primeiro caso em Portugal foi há três meses. Dos infectados, 60% já recuperaram*. Obtido de Publico: https://www.publico.pt/2020/06/02/sociedade/noticia/caso-portugal-ha-tres-meses-infectados-60-ja-recuperaram-1918976

Classificação dos robôs. (24 de abril de 2011). Obtido em 8 de novembro de 2021, de tegruposete7: https://tegruposete7.wordpress.com/classificacao-dos-robos/

Crispin, A. J. (1997). *Programmable Logic Controllers and their Engineering Applications* (2º ed.). Shoppenhangers Road, Maidenhead, Berkshire, SL6 2QL, England: McGaw-Hill Publishing Company.

De Negri, V. J., Kinceler, R., & Silveira, J. (1998). *Automação e Controlo Experimental em Hidráulica e Pneumática.* Florianópolis: Universidade Federal de Santa Catarina.

Demartini, F. (28 de abril de 2021). *Windows 10 está em 1,3 bilhão de PCs no mundo*. Fonte: Canal Tech: https://canaltech.com.br/resultados-financeiros/windows-10-esta-em-13-bilhao-de-pcs-no-mundo-183713/

Dhouibi, H. (17 de abril de 2019). *Que robô colaborativo escolher?* Obtido em 29 de 10 de 2021, de Direct Industry: https://guide.directindustry.com/pt/que-robo-colaborativo-escolher/

Dias, B. (19 de julho de 2017). *Ranking dos 14 maiores fabricantes mundiais de robôs industriais*. Obtido de ServNews - Robótica e Automação: https://servnews.com.br/site/ranking-dos-14-maiores-fabricantes-mundiais-de-robos-industriais/

Estudar com Você. (22 de agosto de 2017). *1ª Revolução Industrial*. Obtido em 21 de junho de 2021, de Estudar com Você: https://estudar.com.vc/conceitos/1-e-2-revolucoes-industriais/14837-1-revolucao-industrial

European Commission. (13 de setembro de 2017). *COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT, THE EUROPEAN COUNCIL,*

THE COUNCIL, THE EUROPEAN ECONOMIC AND SOCIAL COMMITTEE, THE COMMITTEE OF THE REGIONS AND THE EUROPEAN INVESTMENT BANK Investing in a smart, innovative and sustainable Ind. Obtido em 1 de novembro de 2021, de European Commission: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=COM:2017:0479:FIN

Fanuc. (2021). *Robôs SCARA para uma maior produtividade*. Obtido de Fanux: https://www.fanuc.eu/pt/pt/robôs/página-filtro-robôs/scara-series

Francisco, A. (2015). *Autómatos Programáveis* (5º ed.). Av. Praia da Vitória, 14 - 1000-247 Lisboa: ETEP – Edições Técnicas e Profissionais.

Gotz, W. (1991). Hidráulica: Teoria e Aplicações da Bosch. Em F. REVER, *Apontamentos da disciplina de Sistemas Hidráulicos e Pneumáticos - Óleo-Hidráulica* (p. 1 (Valvulas Direccionais)). Porto: Faculdade de Engenharia da Universidade do Porto.

Inteligencia Artificial. (janeiro de 2021). *2021 Próximas tendências e futuro do mercado de robótica industrial*. Obtido de Inteligencia Artificial - Portal de Tecnologia: https://www.inteligenciaartificial.me/2021-proximas-tendencias-e-futuro-do-mercado-de-robotica-industrial/

International Federation of Robotics. (2021). *Executive Summary World Robotics 2021 Industrial Robots - Chapter 1*. Obtido de International Federation of Robotics: https://ifr.org/img/worldrobotics/WR_Industrial_Robots_2021_Chapter_1.pdf

International Federation of Robotics. (28 de outubro de 2021). *Welcome to the presentation of World Robotics 2021*. Obtido em 8 de outubro de 2021, de International Federation of Robotics: https://ifr.org/downloads/press2018/2021_10_28_WR_PK_Presentation_long_version.pdf

International Organization for Standardization. (2012). *Robots and robotic devices.* Obtido de ISO Standard No. 8373:2012: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en

International Organization for Standardization. (fevereiro de 2016). *ISO/TS 15066:2016 - Robots and robotic devices — Collaborative robots.* Obtido em 8 de novembro de 2021, de International Organization for Standardization: https://www.iso.org/standard/62996.html

Jack, H. (2008). *Automating Manufacturing Systems with PLCs.* Free Software Foundation. Obtido de http://www.freeinfosociety.com/media/pdf/2908.pdf

JHernando. (5 de março de 2020). *Tipos de tapetes para a indústria*. Obtido de JHernando: https://www.jhernando.es/pt/blog/tipos-de-tapetes-para-a-industria

Jogo Quatro em linha tridimensional. (5 de novembro de 2021). Obtido em 10 de novembro de 2021, de CustoJusto.pt: https://www.custojusto.pt/setubal/desporto-lazer/brinquedos-jogos/jogo-quatro-em-linha-tridimens-30203293

KUKA AG. (29 de julho de 2016). *KUKA Sunrise.OS 1.11 KUKA Sunrise.Workbench 1.11 Operating and Programming Instructions for System Integrators.* Obtido em 29 de 10 de 2021, de KUKA AG: http://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/KUKA_SunriseOS_111_SI_en.pdf

KUKA AG. (23 de outubro de 2017). *KUKA KR C4: The Power of Control*. Obtido de KUKA AG: https://www.kuka.com/pt-br/produtos-serviços/sistemas-de-robô/unidades-de-comando-de-robô/kr-c4

KUKA AG. (25 de janeiro de 2018). *LBR iiwa | KUKA AG*. Obtido em 29 de 10 de 2021, de KUKA AG: https://www.kuka.com/pt-pt/produtos-serviços/sistemas-de-robô/robôs-industriais/lbr-iiwa

KUKA KCP KR C1 69-000-422 colgante de enseñanza KRC1 usado. (s.d.). Obtido em 10 de novembro de 2021, de AliExpress: https://es.aliexpress.com/item/32785448720.html

Kuka LBR iiwa Programming Learn Module. (s.d.). Obtido em 9 de novembro de 2021, de UNSW Making: https://www.making.unsw.edu.au/learn/kuka-lbr-iiwa-programming-learn-module/

KUKA Robots IBÉRICA, S.A. (15 de setembro de 2020). *Robô articulado LBR iiwa series*. Obtido de Direct Industry: https://www.directindustry.com/pt/prod/kuka-ag/product-17587-1650349.html

ladderlogicworld.com. (2017). *PLC Manufacturers: The Latest PLC Brands, Rankings & Revenues*. Obtido em 05 de setembro de 2021, de Ladder Logic World: https://ladderlogicworld.com/plc-manufacturers/

Loureiro, H. (2017). *C# 7.0 com Visual Studio Curso Completo.* Lisboa: FCA - Editora de Informática, Lda.

Nunes, E. (14 de Maio de 2015). *Tear Mecânico*. Obtido de Invenções dos Séculos XVIII-XIX: http://invencoesseculoxix.blogspot.com/2015/05/tear-mecanico.html

Oliveira, F. R. (4 de julho de 2019). *Transdutor Digital de Movimento – Encoder*. Obtido de Embarcados: https://www.embarcados.com.br/transdutor-digital-de-movimento-encoder/

Panasonic. (21 de dezembro de 2020). *How does a latching relay work?* Obtido de Panasonic: https://ac-faq.industrial.panasonic.com/en/faq_detail.html?id=41315&category=&page=2

Parr, E. (1993). *Programmable Controllers: An Engineer's Guide.* Oxford: Reed Educational and Professional Publishung Ltd.

Pentikäinen, T., & Richard, S. (junho de 2016). *How to make collaborative robot programming easier: Workflow visualization on a table device.* Obtido de Uppsala Universitet: https://uu.diva-portal.org/smash/get/diva2:955637/FULLTEXT01.pdf

Pereira, J. E. (28 de 10 de 2021). *A Caixa de velocidades Manual e sua lubrificação*. Obtido de TEXOLEO.com: https://blog.texoleo.eu/caixa-velocidades-manual-lubrificacao/

Petruzella, F. D. (2011). *Programmable Logic Controllers* (4ª ed.). 1221 Avenue of the Americas, New York, NY, 10020., Estados Unidos da América: The McGraw-Hill Companies, Inc.

Pilz. (24 de julho de 2021). *Relé de segurança*. Obtido de Pilz: https://www.pilz.com/pt-PT/support/knowhow/lexicon/articles/072106

Pilz. (24 de julho de 2021). *Relé de segurança DC PZE X4 #774585 | Pilz | PZE X4 #774585 | 774585 | Dispositivo monitoramento circuitos eibabo.pt*. Obtido de Eibabo: https://www.eibabo.pt/pilz/rele-de-seguranca-dc-pze-x4-774585-eb10705530?utm_source=Portals&utm_medium=CPC&utm_campaign=eibabo-PT_GoogleShopping_PT&gclid=Cj0KCQjw9O6HBhCrARIsADx5qCSutgcq1kYaaFHJMEQSDq7BkuVBsZNB--lb_N_Eh-P0vBICF7OqhzQaAmiBEALw_wcB

Pinto, J. R. (2021). *Tecnologias de Automação na Industria 4.0.* Lisboa: Lidel.

Pires, J. N. (2018). *ROBÓTICA INDUSTRIAL - INDÚSTRIA 4.0.* Lisboa: LIDEL.

Platbrood, F., & Görnemann, O. (junho de 2018). *SAFE ROBOTICS – A SEGURANÇA EM SISTEMAS.* Obtido de SICK AG ARTIGO TÉCNICO: https://cdn.sick.com/media/docs/5/15/115/whitepaper_safe_robotics_pt_im0073115.pdf

RealPars (Realizador). (2018). *Types of Sensors* [Filme]. Youtube. Obtido de https://www.youtube.com/watch?v=J_KoRp8SnoE

RealPars (Realizador). (2019). *What is a Spool Valve?* [Filme]. Youtube. Obtido de https://www.youtube.com/watch?v=Jfdmrm4A99s

RealPars (Realizador). (2020). *What is a Sensor? Different Types of Sensors, Applications* [Filme]. Youtube. Obtido de https://www.youtube.com/watch?v=XI49uFm5HRE

RobotWorx. (s.d.). *The New KUKA KR C4 Robot Controller*. Obtido em 10 de novembro de 2021, de RobotWorx: https://www.robots.com/blogs/the-new-kuka-kr-c4-robot-controller

Rockwell Automation. (julho de 2015). *Release Notes*. Obtido de Rockwell Automation: https://compatibility.rockwellautomation.com/GeneratedReleaseNote.aspx?v1=54706&v2=54706&o=&pdf=0

Rockwell Automation. (5 de março de 2018). *Press Release Details*. Obtido de Rockwell Automation: https://ir.rockwellautomation.com/press-releases/press-releases-details/2018/Studio-5000-Software-Release-Optimizes-Productivity-and-Reduces-Design-Time/default.aspx

Saber Eletrica. (26 de julho de 2021). *O Que é Chave Fim de Curso e Aplicações*. Obtido de Saber Eletrica: https://www.sabereletrica.com.br/chave-fim-de-curso/

Santos, G. (25 de fevereiro de 2020). *O que é Automação Industrial?* Obtido de Automação Industrial: https://www.automacaoindustrial.info/o-que-e-automacao-industrial/

Santos, V. M. (2003). *Robótica Industrial.* Obtido de Universidade de Aveiro - Departamento de Engenharia Mecânica: http://lars.mec.ua.pt/public/LAR%20Projects/Humanoid/2013_EmilioEstrelinha/Dissertação_Emílio_Estrelinha/Datasheets/RoboticaIndustrial-Sebenta2003-2004-v2a.pdf

Schneider Electric. (8 de maio de 2021). *EcoStruxure Machine Expert-Basic*. Obtido de Schneider Electric: https://www.se.com/ww/en/download/document/Machine_Expert_Basic/

Schneider Electric. (1 de março de 2021). *PL7 PRO V4.5 SERVICE PACK 1*. Obtido de Schneider Electric: https://www.se.com/ww/en/faqs/FA197508/

Schneider Electric HMIGK2310. (s.d.). Obtido em 10 de novembro de 2021, de Mouser Electronics: https://pt.mouser.com/ProductDetail/Schneider-Electric/HMIGK2310?qs=VTAFTc7iVivUu09SkiTUNQ%3D%3D

Schweber, B. (13 de fevereiro de 2018). *Rotary encoder basics and applications, Part 1: Optical encoders*. Obtido de Analog IC Tips: https://www.analogictips.com/rotary-encoders-part-1-optical-encoders/

Siemens. (8 de junho de 2004). *Delivery Release for STEP 7-Micro/WIN V4.0*. Obtido de Siemens Support Industry: https://support.industry.siemens.com/cs/document/19300593/delivery-release-for-step-7-micro-win-v4-0?dti=0&lc=en-WW

Siemens. (23 de novembro de 2010). *Siemens introduces TIA software engineering framework*. Obtido de Automation.com: https://www.automation.com/en-us/products/product13/siemens-introduces-tia-software-engineering-framew

Siemens Brasil. (14 de junho de 2021). *SIMATIC S7-1200 - Pode ser adaptado de forma flexível as suas necessidades*. Obtido de Siemens: https://new.siemens.com/br/pt/produtos/automacao/controladores/s7-1200.html

Significados. (14 de setembro de 2016). *Significado de Automação*. Obtido de Significados: https://www.significados.com.br/automacao/

Technavio. (31 de agosto de 2018). *6 Major Types of Industrial Robots Used in the Global Manufacturing 2018*. Obtido de Technavio: https://blog.technavio.com/blog/major-types-of-industrial-robots

TELE Controls Inc. (Realizador). (2020). *Switching Relay Basics | TELE Session 15* [Filme]. Youtube. Obtido de https://www.youtube.com/watch?v=WqpJASWK60k

The Engineering Minset (Realizador). (2020). *How Relays Work - Basic working principle electronics engineering electrician amp* [Filme]. Youtube. Obtido de https://www.youtube.com/watch?v=n594CkrP6xE&t=1s

Tisserant, E., Bessard, L., & Sousa, M. d. (23 de junho de 2007). *An Open Source IEC 61131-3 Integrated Development Environment*. Obtido de IEEEXplore: https://ieeexplore.ieee.org/abstract/document/4384753

Trimble, J. (11 de julho de 2013). *Building a Quadrature Rotary Encoder*. Obtido de John Trimble - Tinkerer, Problem Solver, Software Engineer: http://johntrimble.com/blog/2013/07/11/building-an-incremental-rotary-encoder/

Universidade de São Paulo. (s.d.). *A Norma IEC 61131*. Obtido em 29 de julho de 2021, de Universidade de São Paulo: https://edisciplinas.usp.br/pluginfile.php/1881421/mod_resource/content/0/Aula2_1131.pdf

Wisdom Jobs India. (11 de junho de 2011). *INSTRUCTION LISTS - PROGRAMMABLE LOGIC CONTROLLERS*. Obtido em 7 de novembro de 2021, de Programmable Logic Controllers Tutorial: https://www.wisdomjobs.com/e-university/programmable-logic-contrlles-tutorial-523/instruction-lists-14693.html

ANEXOS

INDICE DE ANEXOS

Anexo 1 - Relés temporizados

Neste anexo, serão abordados relés temporizados, não sendo o entendimento deste tópico importante para a compreensão dos restantes conteúdos deste relatório de estágio, ainda que seja importante o seu conhecimento para alguém que trabalhe na área de automação industrial.

Os relés temporizados, consoante a energização do lado primário, afetam de um modo temporal o que se sucede no lado secundário do relé. Os relés temporizados mais comuns são:

- **Relé temporizador com retardo na energização (*on delay*):** retarda com um tempo predefinido a energização do lado secundário assim que o lado primário é energizado;

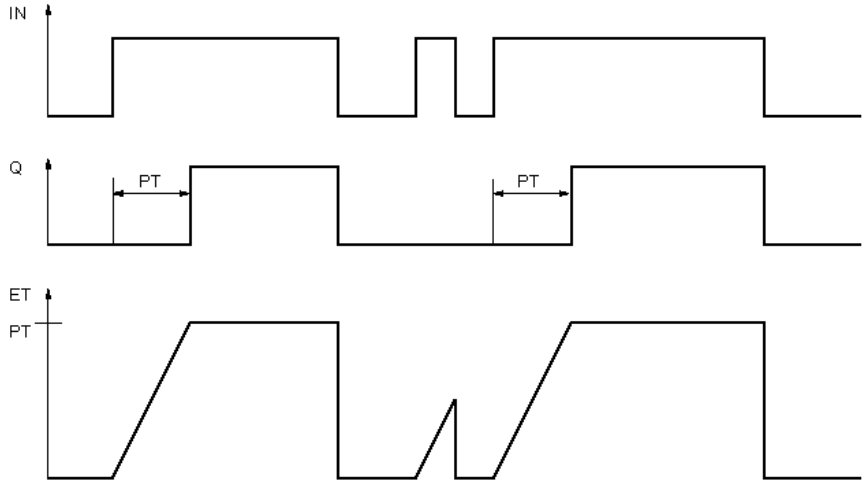


Figura 90 -Gráfico de funcionamento do relé temporizador com retardo na energização. Fonte: manual do utilizador TIA-Portal V15.1 - TON.

- **Relé temporizador com retardo na desenergização (*off delay*):** retarda com um tempo predefinido a desenergização do lado secundário assim que o lado primário é desenergizado;

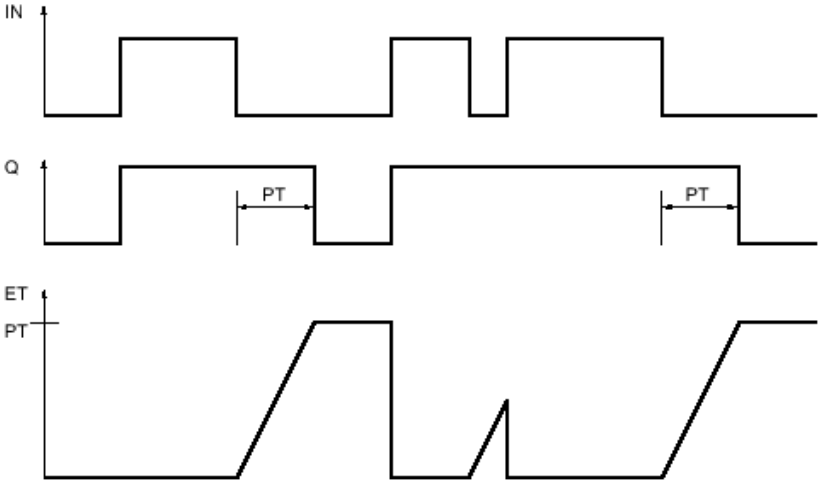


Figura 91 -Gráfico de funcionamento do relé temporizador com retardo na desenergização. Fonte: manual do utilizador TIA-Portal V15.1 - TOF.

- **Relé temporizador com pulso na energização:** energiza por um período de tempo predefinido o lado secundário a partir do momento em que há um pulso positivo do lado primário (independentemente de que este pulso seja mais curto ou mais longo que a duração predefinida do lado secundário, permanece o lado secundário energizado apenas o tempo predefinido até que volte a haver um novo pulso positivo do lado primário);

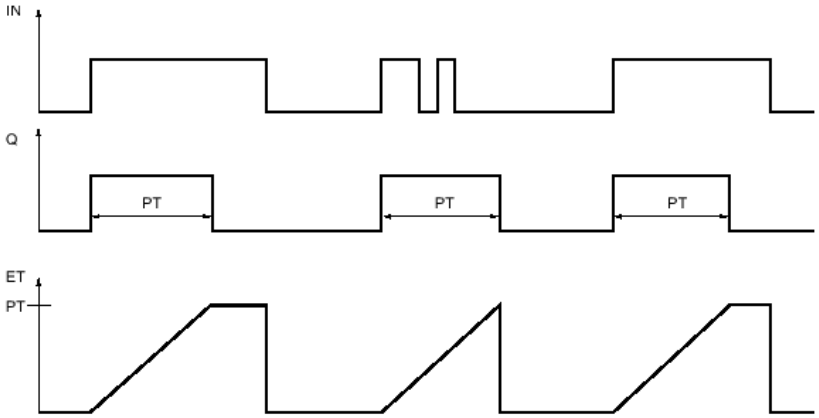


Figura 92 -Gráfico de funcionamento do relé temporizador com pulso de energização. Fonte: manual do utilizador TIA-Portal V15.1 - TP.

- **Relé temporizador cíclico:** quando energizado do lado primário o lado secundário gera pulsos regulares baseados em um tempo predefinido (Aprendendo eletrica, 2019);

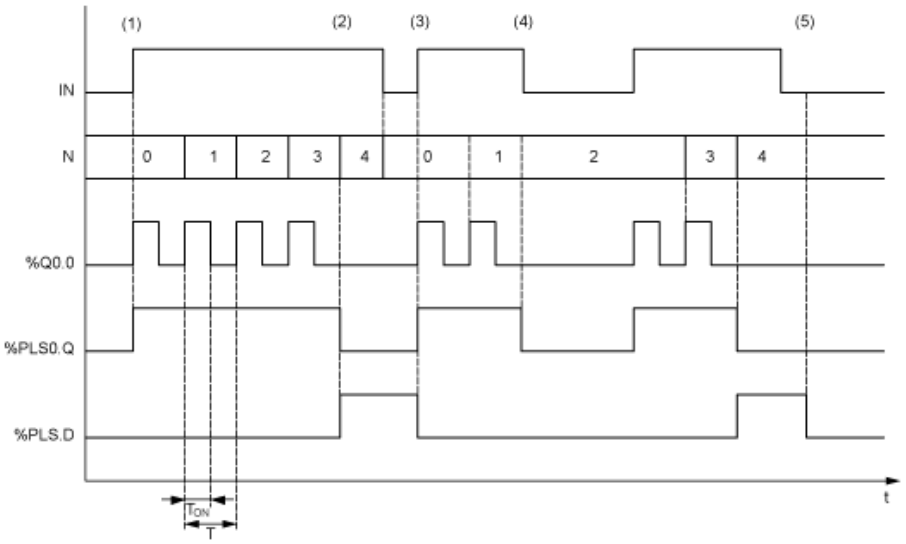


Figura 93 - Gráfico de funcionamento do relé temporizado cíclico. Fonte: Imagem retirada do manual do utilizador de EcoStruxure Machine Expert-Basic.

Anexo 2 - Por dentro de *Ladder Logic*

Este anexo tem como objetivo aprofundar o tema da linguagem de programação *Ladder Logic*, tentando-se demonstrar o funcionamento de cada um dos blocos principais referidos na página 41, explicando o seu funcionamento comparativamente a um relé e também compilando em código de alto nível, sendo que será usada a linguagem de programação *C#* para se alcançar este objetivo. Explicando apenas brevemente o código desenvolvido em *C#*, utilizando os recursos da programação orientada a objetos, foi criada uma classe base abstrata, do qual todas as restantes classes irão herdar. Esta classe chama-se de *BlocoLadder*. Apresenta-se a sua implementação no Bloco de Código 19.

```
abstract class BlocoLadder
{
    public bool EN; //Estado da Energização à entrada
    public bool ENO;    //Estado da Energização à saída

    // Função de apresenta os valores antes da execução do bloco
    protected void MostraValores_Entrada(in bool Variavel)
    {
        EN.Dump("Valor da entrada do bloco antes de ser "
                            +   "executado bloco");
        Variavel.Dump("Valor da Variável associada a este bloco "
                    + "antes de ser executado o bloco");
        ENO.Dump("Valor da saída do bloco antes de ser "
                          + "executado o bloco");
    "================================".Dump("A executar...");
    }

    // Função de apresenta os valores depois da execução do bloco
    protected void MostraValores_Saida(in bool Variavel)
    {
EN.Dump("Valor da entrada do bloco depois de ser "
                       +  "executado o bloco");
        Variavel.Dump("Valor da Variável associada a este bloco "
                    + "depois de ser executado o bloco");
        ENO.Dump("Valor da saída do bloco depois de ser "
                            + "executado o bloco");
        "\n\n\n".Dump();
    }
}
```

Bloco de Código 19 - Implementação da classe base dos Blocos Ladder para exemplificar o seu funcionamento. Fonte: própria.

Nota: O programa usado para obter o resultado foi o *LINQPad 6*, sendo que todas as sobrecargas do método *Dump* são um método de extensão do tipo *object*, que permite apresentar os resultados no ecrã e pertentem à livraria do programa.

Contacto normalmente fechado

O contacto normalmente fechado, em *ladder logic*, tem a mesma funcionalidade do contacto normalmente fechado no relé. Consoante o estado de energização do lado primário, também o contacto normalmente fechado, do lado secundário, está energizado ou não. Importante notar que esta é uma representação do lado secundário do relé e que por isso, não escreve no estado da variável associado ao bloco, mas apenas lê este.

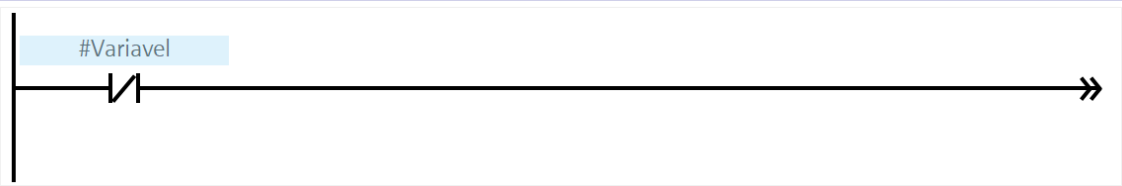


Figura 94 - Representação em ladder logic o contacto normalmente fechado. Fonte: própria, TIA Portal V15.1.

Desta forma, existem duas informações booleanas que definem o modo de funcionamento de este bloco, sendo elas o estado da energização na entrada do bloco (do lado esquerdo) sendo normalmente chamado de *EN* e a outra variável, é a variável associada ao bloco (a sua *tag*). Consoante o resultado de estas duas variáveis o bloco define o valor da sua saída, normalmente chamado de *ENO*. Desta forma, conseguimos obter a seguinte tabela de valores:

Tabela 5 - Tabela de valores lógicos para o Contacto normalmente fechado em ladder logic.

EN	Tag	ENO
0	0	0
1	0	1
0	1	0
1	1	0

Desta forma, conseguimos entender que se não houver energização na entrada do bloco, a sua saída nunca tem energia, independentemente do valor da *Tag*. Para a saída ser energizada, é necessário que a sua entrada esteja energizada e a sua *Tag* não.

Apresentamos no Bloco de Código 20 a implementação da classe deste bloco na linguagem de alto nível.

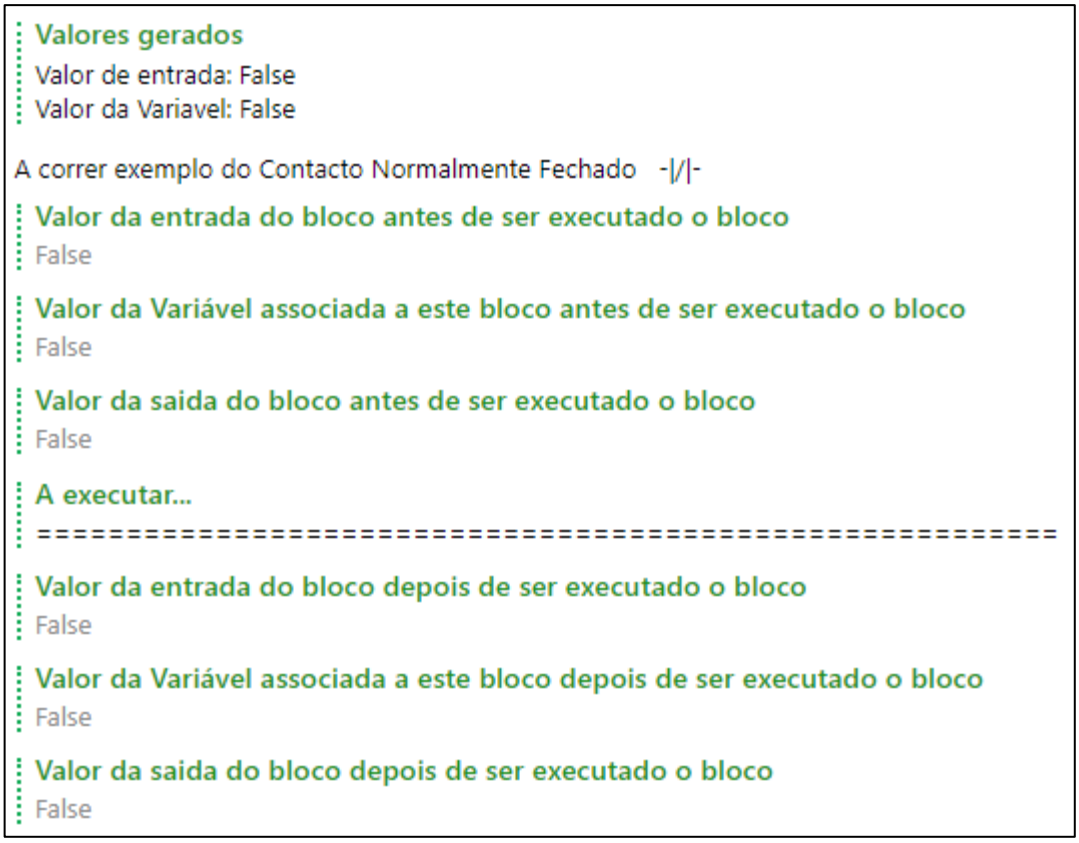
```
// Classe que representa o Contacto Normalmente Fechado
// Este classe herda do BlocoLadder
class Contacto_NC : BlocoLadder
{
    //Função que faz executar as funções do bloco
    //Parâmetros:
    //  Variavel: Recebe por referência a Variável
    public void ExecutarBloco(in bool Variavel)
    {
        MostraValores_Entrada(Variavel);
        if(EN)
            ENO = !Variavel ? true : false;
        else
            ENO = false;
        MostraValores_Saida(Variavel);
    }
}
```

Bloco de Código 20 - Implementação da classe do contacto normalmente fechado, de ladder logic, em uma linguagem de alto nível. Fonte: Própria.

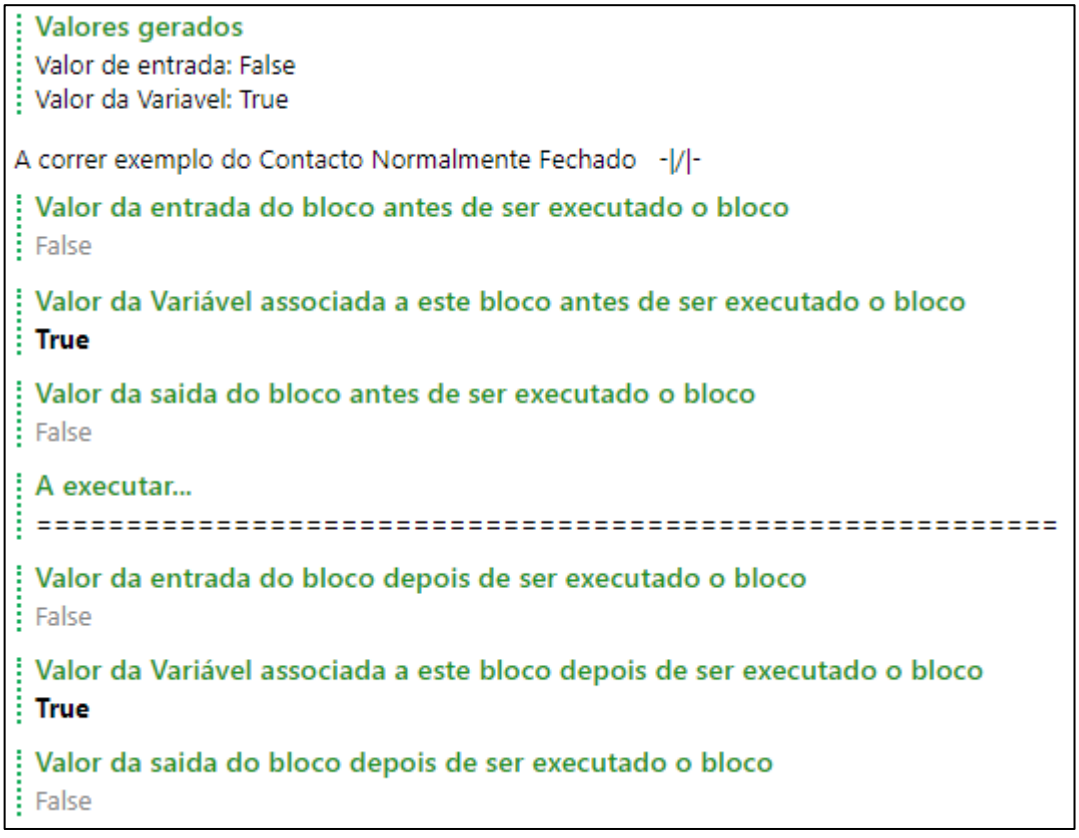
Quando executado o Bloco de Código 21 obtemos os seguintes resultados apresentados no Resultado 1, Resultado 2, Resultado 3 e Resultado 4.

```
void Main()
{
    //Instancia um novo objeto do tipo Contacto_NC
    Contacto_NC Contacto_NC = new Contacto_NC();

    for(int i = 0; i < 4; i++)
    {
        //gera a sequencia -> 0011
        bool ValorDeEntrada = i >= 2 ? true : false;
        //gera a sequencia -> 0101
        bool Variavel = i % 2 != 0 ? true : false;

        $"Valor de entrada: {ValorDeEntrada}\nValor da " +
            $"Variavel: {Variavel}".Dump("Valores gerados");
        //Exemplo do contacto normalmente fechado
        Contacto_NC.EN = ValorDeEntrada;
        "A correr exemplo do Contacto Normalmente Fechado   -"
                            +   "|/|-".Dump();
        Contacto_NC.ExecutarBloco(Variavel);
    }
}
```

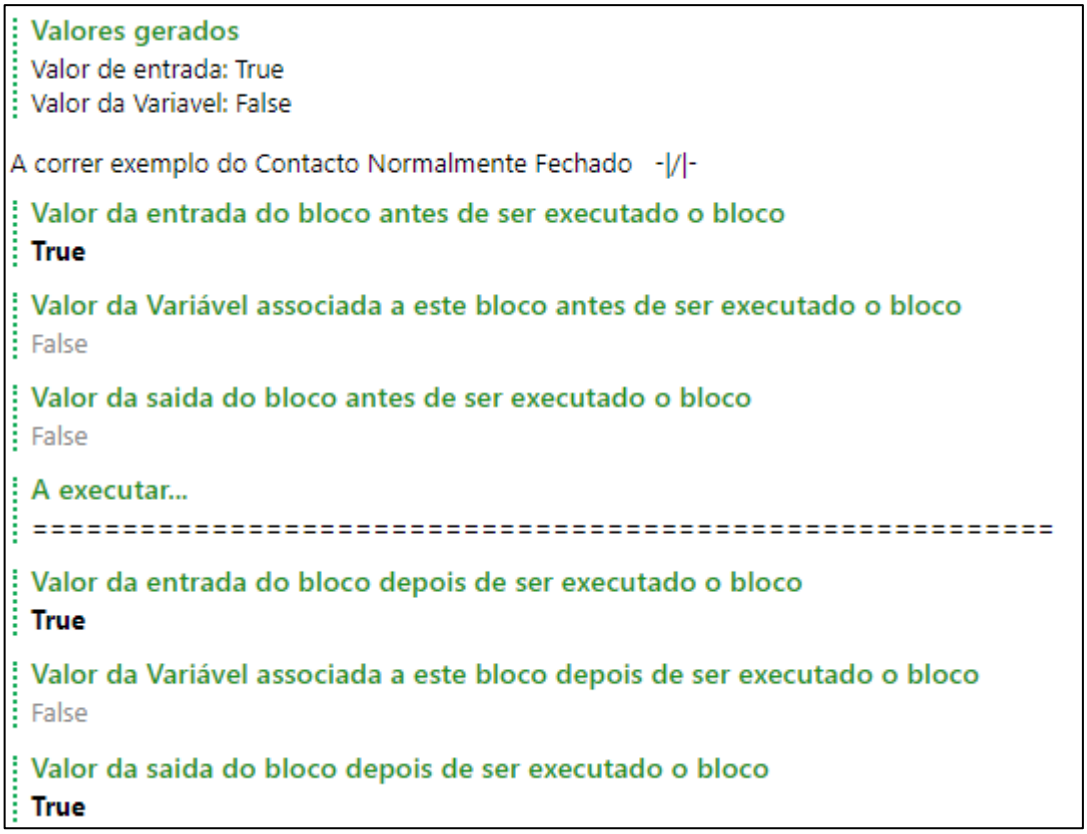
Bloco de Código 21 - Bloco de código que corre para cada um dos estados para exemplificar o funcionamento do bloco normalmente fechado.



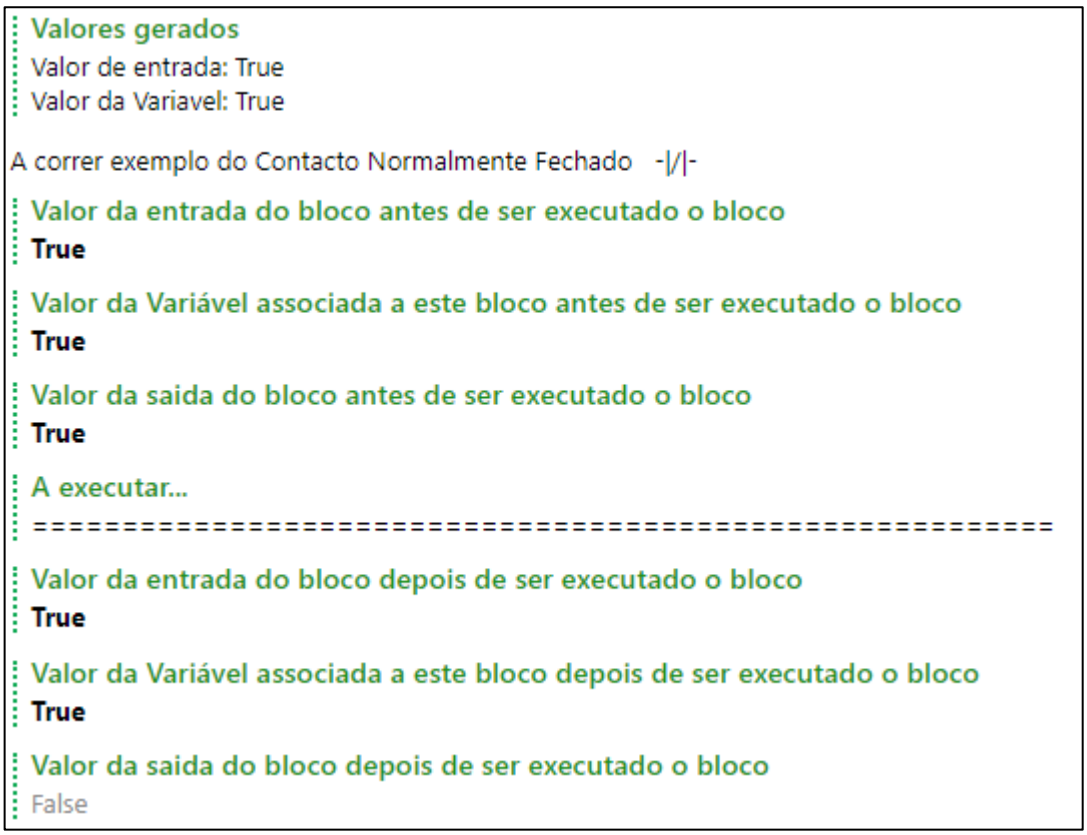
Resultado 1 – Resultado ¼ do Bloco de Código 21.



Resultado 2 – Resultado 2/4 do Bloco de Código 21.



Resultado 3 – Resultado ¾ do Bloco de Código 21.



Resultado 4 -Resultado 4/4 do Bloco de Código 21.

Contacto normalmente aberto

Seguindo a mesma lógica do contacto normalmente fechado, o funcionamento do contacto normalmente aberto é o inverso do abordado anteriormente. Desta forma, nunca existe escrita na variável, assim como no bloco anterior.

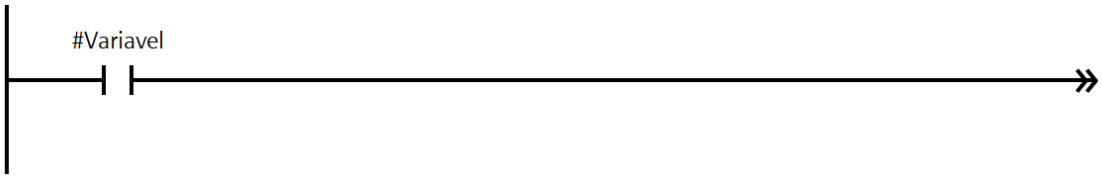


Figura 95 - Representação em ladder logic o contacto normalmente aberto. Fonte: própria, TIA Portal V15.1.

Da mesma forma que mostrado anteriormente, obtemos então a seguinte tabela de valores consoante o estado das suas entradas:

Tabela 6 - Tabela de valores lógicos para o Contacto normalmente aberto em ladder logic.

EN	Tag	ENO
0	0	0
1	0	0
0	1	0
1	1	1

Olhando aos resultados da Tabela 6, estes não diferem muito dos da Tabela 5, da mesma forma, o código da linguagem de alto nível, também não apresenta significativas mudanças.

```
// Classe que representa o Contacto Normalmente Aberto
// Este classe herda do BlocoLadder
class Contacto_NO : BlocoLadder
{
    //Função que faz executar as funções do bloco
    //Parâmetros:
    //  Variavel: Recebe por referência a Variavel
    public void ExecutarBloco(in bool Variavel)
    {
        MostraValores_Entrada(Variavel);
        if(EN)
            ENO = Variavel ? true : false;
        else
            ENO = false;
        MostraValores_Saida(Variavel);
    }
}
```

Bloco de Código 22 - Implementação da classe do contacto normalmente aberto, de ladder logic, em uma linguagem de alto nível. Fonte: Própria.

Com uma pequena alteração no método do bloco, Bloco de Código 22, conseguimos obter o resultado que se pretende, podendo assim ser testado no Bloco de Código 23.

```
void Main()
{
    //Instancia um novo objeto do tipo Contacto_NO
    Contacto_NO Contacto_NO = new Contacto_NO();

    for(int i = 0; i < 4; i++)
    {
        //gera a sequencia -> 0011
        bool ValorDeEntrada = i >= 2 ? true : false;
        //gera a sequencia -> 0101
        bool Variavel = i % 2 != 0 ? true : false;

        $"Valor de entrada: {ValorDeEntrada}\nValor da "
            +"Variavel: {Variavel}".Dump("Valores gerados");
        //Exemplo do contacto normalmente aberto
        Contacto_NO.EN = ValorDeEntrada;
        "A correr exemplo do Contacto Normalmente Aberto   -| "
                                +   "|-".Dump();
        Contacto_NO.ExecutarBloco(Variavel);
    }
}
```

Bloco de Código 23 - Bloco de código que corre para cada um dos estados para exemplificar o funcionamento do bloco normalmente aberto.

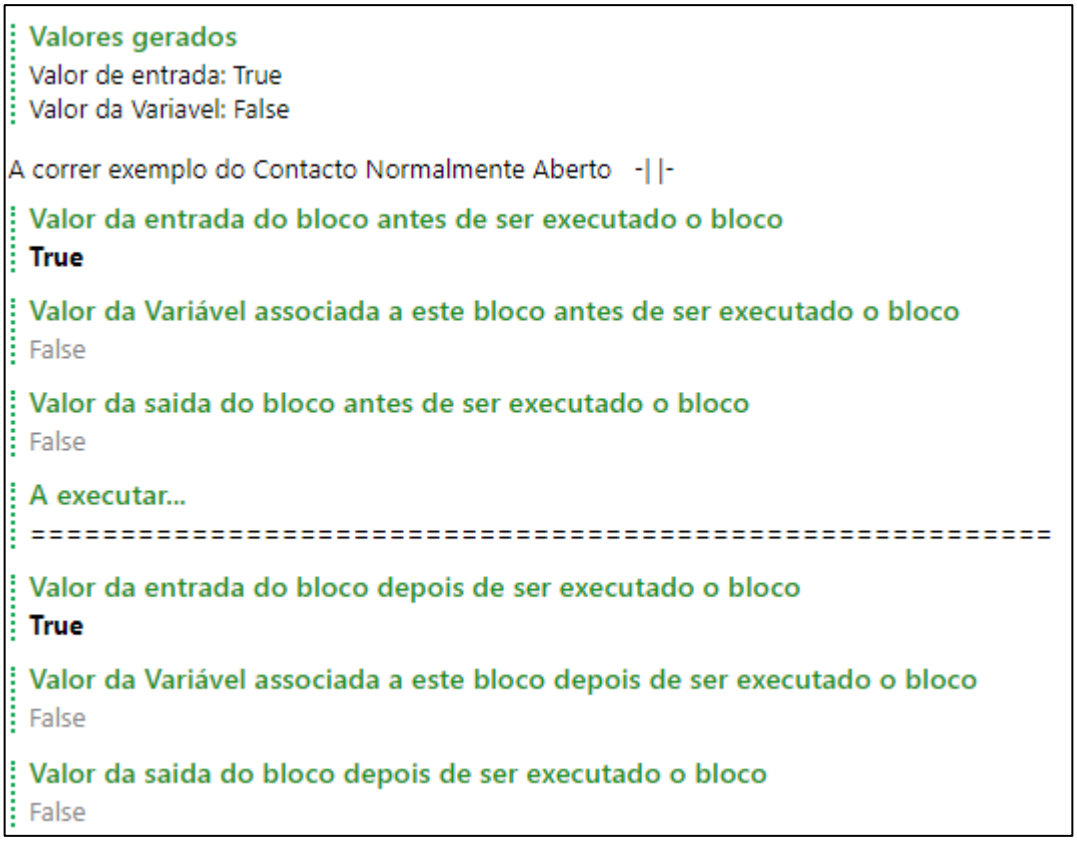
Os resultados obtidos são apresentados no Resultado 5, Resultado 6, Resultado 7 e Resultado 8.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: False

A correr exemplo do Contacto Normalmente Aberto  -| |-
Valor da entrada do bloco antes de ser executado o bloco
False

Valor da Variável associada a este bloco antes de ser executado o bloco
False

Valor da saida do bloco antes de ser executado o bloco
False

A executar...
============================================================

Valor da entrada do bloco depois de ser executado o bloco
False

Valor da Variável associada a este bloco depois de ser executado o bloco
False

Valor da saida do bloco depois de ser executado o bloco
False
```

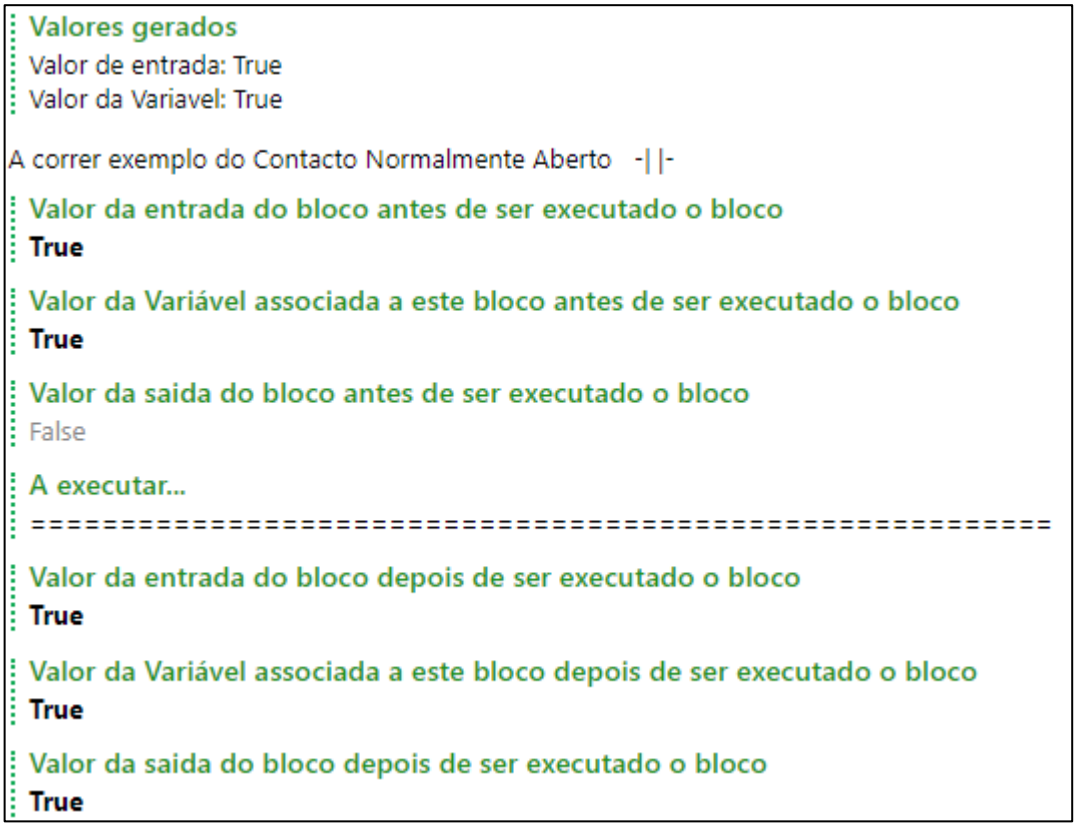
Resultado 5 - Resultado ¼ do Bloco de Código 23.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: True

A correr exemplo do Contacto Normalmente Aberto  -| |-
Valor da entrada do bloco antes de ser executado o bloco
False

Valor da Variável associada a este bloco antes de ser executado o bloco
True

Valor da saida do bloco antes de ser executado o bloco
False

A executar...
============================================================

Valor da entrada do bloco depois de ser executado o bloco
False

Valor da Variável associada a este bloco depois de ser executado o bloco
True

Valor da saida do bloco depois de ser executado o bloco
False
```

Resultado 6 - Resultado 2/4 do Bloco de Código 23.



Resultado 7 - Resultado ¾ do Bloco de Código 23.



Resultado 8 - Resultado 4/4 do Bloco de Código 23.

Coil

A *coil*, em *ladder logic*, é equivalente à bobine existente do lado primário no relé. Enquanto no relé, esta quando energizada cria um campo magnético que permite que a corrente flua entre o contacto comum e o contacto normalmente aberto, em *ladder logic* também consoante se está energizada ou não, é escrito na variável (*tag*) verdadeiro ou falso.

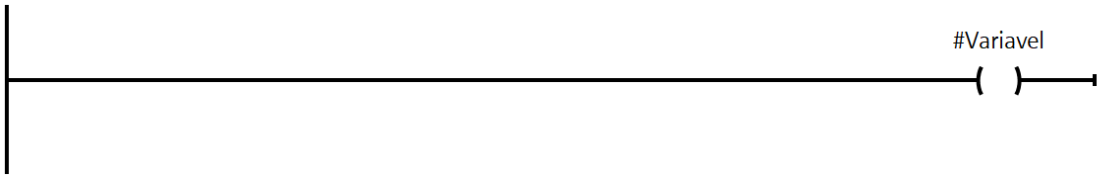


Figura 96 - Representação de uma Coil em ladder logic.

Desta forma, a nossa tabela de valores vai ser diferente uma vez que, ao contrário dos dois elementos anteriores, a *coil* apenas necessita de saber o seu valor de entrada, uma vez que não é dependente do valor da sua variável. Isto porque a *coil* apenas escreve na variável e não lê. Dado isto, temos uma pequena tabela de valores possível.

Tabela 7 - Tabela de resultados possíveis para a coil em ladder logic.

EN	Tag	ENO
0	0	0
1	1	1

Alguns IDE de *ladder logic*, apenas permitem que a *coil* seja o último elemento da linha, não sendo possível colocar um outro bloco à frente deste, pelo que o seu sinal de saída (*ENO*) pode ser ignorado.

Dado a sua simples tabela de valores, também a sua implementação na linguagem de alto nível é simples.

```
// Classe que representa a Coil
// Este classe herda do BlocoLadder
class Coil : BlocoLadder
{
    //Função que faz executar as funções do bloco
    //Parâmetros:
    //  Variavel: Recebe por referência a Variável
    //              e esta pode ser escrita
    public void ExecutarBloco(ref bool Variavel)
    {
        MostraValores_Entrada(Variavel);
        ENO = Variavel = EN ? true : false;
        MostraValores_Saida(Variavel);
    }
}
```

Bloco de Código 24 - Implementação da classe da coil, de ladder logic, em uma linguagem de alto nível. Fonte: Própria.

Como se pode visualizar no Bloco de Código 24, o método que corre o nosso bloco é na verdade uma linha de código com a ajuda de um *ternary operator*[38], sendo que os outros dois métodos que são chamados servem para efeito de visualização do que se sucedeu antes e depois da execução do bloco. Nota também para o facto de nunca haver qualquer leitura do valor da variável (a *tag*), sendo que esta é sempre escrita quando o bloco está energizado.

Executou-se o código desenvolvido no Bloco de Código 8 e são apresentados os seus resultados nos Resultado 9, Resultado 10, Resultado 11 e Resultado 12.

[38] Para mais informação acerca deste operador, também conhecido como *conditional operator*, pode-se consultar o seguinte link: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator

```
void Main()
{
    //Instancia um novo objeto do tipo Coil
    Coil Coil = new Coil();

    for(int i = 0; i < 4; i++)
    {
        //gera a sequencia -> 0011
        bool ValorDeEntrada = i >= 2 ? true : false;
        //gera a sequencia -> 0101
        bool Variavel = i % 2 != 0 ? true : false;

        $"Valor de entrada: {ValorDeEntrada}\nValor da "
            + $"Variavel: {Variavel}".Dump("Valores gerados");
        //Exemplo do contacto normalmente aberto
        Coil.EN = ValorDeEntrada;
        "A correr exemplo da Coil   -( )-".Dump();
        Coil.ExecutarBloco(Variavel);
    }
}
```

Bloco de Código 25 - Bloco de código que corre para cada um dos estados para exemplificar o funcionamento da coil.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: False

A correr exemplo da Coil   -( )-
Valor da entrada do bloco antes de ser executado o bloco
False
Valor da Variável associada a este bloco antes de ser executado o bloco
False
Valor da saida do bloco antes de ser executado o bloco
False
A executar...
============================================================
Valor da entrada do bloco depois de ser executado o bloco
False
Valor da Variável associada a este bloco depois de ser executado o bloco
False
Valor da saida do bloco depois de ser executado o bloco
False
```

Resultado 9 - Resultado 1/4 do Bloco de Código 25.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: True

A correr exemplo da Coil   -( )-
Valor da entrada do bloco antes de ser executado o bloco
False

Valor da Variável associada a este bloco antes de ser executado o bloco
True

Valor da saida do bloco antes de ser executado o bloco
False

A executar...
============================================================

Valor da entrada do bloco depois de ser executado o bloco
False

Valor da Variável associada a este bloco depois de ser executado o bloco
False

Valor da saida do bloco depois de ser executado o bloco
False
```

Resultado 10 - Resultado 2/4 do Bloco de Código 25.

```
Valores gerados
Valor de entrada: True
Valor da Variavel: False

A correr exemplo da Coil   -( )-
Valor da entrada do bloco antes de ser executado o bloco
True

Valor da Variável associada a este bloco antes de ser executado o bloco
False

Valor da saida do bloco antes de ser executado o bloco
False

A executar...
============================================================

Valor da entrada do bloco depois de ser executado o bloco
True

Valor da Variável associada a este bloco depois de ser executado o bloco
True

Valor da saida do bloco depois de ser executado o bloco
True
```

Resultado 11 - Resultado 3/4 do Bloco de Código 25.

```
Valores gerados
Valor de entrada: True
Valor da Variavel: True

A correr exemplo da Coil   -( )-
Valor da entrada do bloco antes de ser executado o bloco
True
Valor da Variável associada a este bloco antes de ser executado o bloco
True
Valor da saida do bloco antes de ser executado o bloco
True
A executar...
============================================================
Valor da entrada do bloco depois de ser executado o bloco
True
Valor da Variável associada a este bloco depois de ser executado o bloco
True
Valor da saida do bloco depois de ser executado o bloco
True
```

Resultado 12 - Resultados 4/4 do Bloco de Código 25.

Set Coil

A *set coil*, em *ladder logic*, é o equivalente ao acionar a *setting coil* no relé de *latching*.

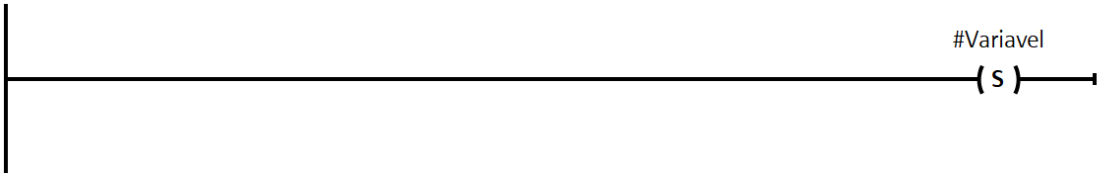


Figura 97 - Representação do bloco set coil em ladder logic. Fonte: Própria

Da mesma forma que a coil, esta também depende apenas da energização na sua entrada para escrever na variável. No entanto, a *set coil* apenas escreve *true* quando energizada e não escreve nada quando não está energizada, exatamente como o relé da *setting coil*. Olhando para a tabela de valores, obtemos os seguintes dados:

Tabela 8 - Tabela de resultados possíveis para a set coil em ladder logic.

EN	Tag	ENO
0	Indiferente	0
1	1	1

Pela tabela e como abordado anteriormente, pode-se entender que este bloco apenas escreve na variável quando energizada do lado primário e quando não energizado este não faz qualquer mudança na variável.

Olhando para o código desenvolvido na linguagem de alto nível, isto se torna mais claro. Apresentamos de seguida a classe desenvolvida para representar o funcionamento deste bloco.

```
// Classe que representa o Set Coil
// Este classe herda do BlocoLadder
class Coil_Set : BlocoLadder
{
    //Função que faz executar as funções do bloco
    //Parâmetros:
    //  Variavel: Recebe por referência a Variavel
    //            e escreve na mesma.
    public void ExecutarBloco(ref bool Variavel)
    {
        MostraValores_Entrada(Variavel);
        if(EN)
            Variavel = true;
        ENO = EN;
        MostraValores_Saida(Variavel);
    }
}
```

Bloco de Código 26 - Implementação da classe set coil, de ladder logic, em uma linguagem de alto nível. Fonte: Própria.

Como se pode ver no Bloco de Código 26, a variável apenas é escrita no caso do EN estar energizado, o que é coincidente com o *relé de latching*, que quando não energizadas nenhuma das bobines, não influenciam o modo de funcionamento do lado secundário.

Os resultados obtidos do código de alto nível apresentado no Bloco de Código 27, são apresentados no Resultado 13, Resultado 14, Resultado 15 e Resultado 16, estes valores batem certo com a tabela de valores.

```
void Main()
{
    //Instancia um novo objeto do tipo Coil_Set
    Coil_Set Coil_Set = new Coil_Set();

    for(int i = 0; i < 4; i++)
    {
        //gera a sequencia -> 0011
        bool ValorDeEntrada = i >= 2 ? true : false;
        //gera a sequencia -> 0101
        bool Variavel = i % 2 != 0 ? true : false;

        $"Valor de entrada: {ValorDeEntrada}\nValor da "
        +   $"Variavel: {Variavel}".Dump("Valores gerados");
        //Exemplo da Set Coil
        Coil_Set.EN = ValorDeEntrada;
        "A correr exemplo da Set Coil   -(S)-".Dump();
        Coil_Set.ExecutarBloco(ref Variavel);
    }
}
```

Bloco de Código 27 - Bloco de código que corre para cada um dos estados para exemplificar o funcionamento do bloco set coil.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: False

A correr exemplo da Set Coil   -(S)-
Valor da entrada do bloco antes de ser executado o bloco
False
Valor da Variável associada a este bloco antes de ser executado o bloco
False
Valor da saida do bloco antes de ser executado o bloco
False
A executar...
==========
Valor da entrada do bloco depois de ser executado o bloco
False
Valor da Variável associada a este bloco depois de ser executado o bloco
False
Valor da saida do bloco depois de ser executado o bloco
False
```

Resultado 13 - Resultado 1/4 do Bloco de Código 27.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: True

A correr exemplo da Set Coil  -(S)-
Valor da entrada do bloco antes de ser executado o bloco
False
Valor da Variável associada a este bloco antes de ser executado o bloco
True
Valor da saida do bloco antes de ser executado o bloco
False
A executar...
==========
Valor da entrada do bloco depois de ser executado o bloco
False
Valor da Variável associada a este bloco depois de ser executado o bloco
True
Valor da saida do bloco depois de ser executado o bloco
False
```

Resultado 14 - Resultado 2/4 do Bloco de Código 27.

```
Valores gerados
Valor de entrada: True
Valor da Variavel: False

A correr exemplo da Set Coil  -(S)-
Valor da entrada do bloco antes de ser executado o bloco
True
Valor da Variável associada a este bloco antes de ser executado o bloco
False
Valor da saida do bloco antes de ser executado o bloco
False
A executar...
==========
Valor da entrada do bloco depois de ser executado o bloco
True
Valor da Variável associada a este bloco depois de ser executado o bloco
True
Valor da saida do bloco depois de ser executado o bloco
True
```

Resultado 15 - Resultado 3/4 do Bloco de Código 27.

```
Valores gerados
Valor de entrada: True
Valor da Variavel: True

A correr exemplo da Set Coil   -(S)-
Valor da entrada do bloco antes de ser executado o bloco
True
Valor da Variável associada a este bloco antes de ser executado o bloco
True
Valor da saida do bloco antes de ser executado o bloco
True
A executar...
============================================================
Valor da entrada do bloco depois de ser executado o bloco
True
Valor da Variável associada a este bloco depois de ser executado o bloco
True
Valor da saida do bloco depois de ser executado o bloco
True
```

Resultado 16 - Resultado 4/4 do Bloco de Código 27.

Reset Coil

Por último temos o bloco do *reset coil*, que é o inverso do *set coil*. Enquanto o *set coil*, do *ladder logic*, está para a *setting coil* do relé de *latching*, o *reset coil*, em *ladder logic*, está para o *resetting coil* em *ladder logic*.

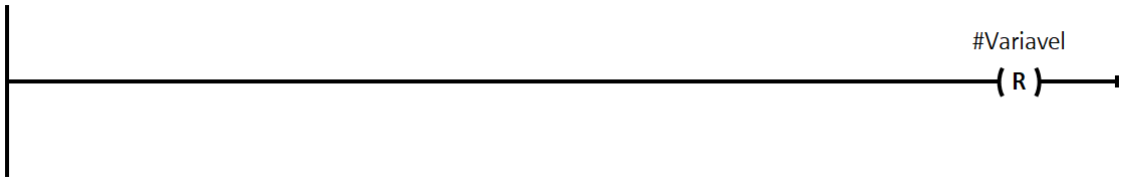


Figura 98 - Representação da reset coil em ladder logic.

Desta forma, obtemos a seguinte tabela de valores possíveis.

Tabela 9 - Tabela de resultados para o bloco reset coil de ladder logic.

EN	Tag	ENO
0	Indiferente	0
1	0	1

Da mesma forma que anteriormente, foi desenvolvido o código equivalente em *C#*. A implementação é muito semelhante com o *set coil*, mudando apenas o valor com que escreve na variável.

Desta forma, podemos testar o código desenvolvido e verificar que os resultados são iguais à tabela de valores.

```
// Classe que representa o Reset Coil
// Este classe herda do BlocoLadder
class Coil_Reset : BlocoLadder
{
    //Função que faz executar as funções do bloco
    //Parâmetros:
    //  Variavel: Recebe por referência a Variavel
    //            e escreve na mesma.
    public void ExecutarBloco(ref bool Variavel)
    {
        MostraValores_Entrada(Variavel);
        if(EN)
            Variavel = false;
        ENO = EN;
        MostraValores_Saida(Variavel);
    }
}
```

Bloco de Código 28 - Implementação da classe da reset coil, de ladder logic, em uma linguagem de alto nível. Fonte: Própria.

```
void Main()
{
    //Instancia um novo objeto do tipo Coil
    Coil_Reset Coil_Reset = new Coil_Reset();

    for(int i = 0; i < 4; i++)
    {
        //gera a sequencia -> 0011
        bool ValorDeEntrada = i >= 2 ? true : false;
        //gera a sequencia -> 0101
        bool Variavel = i % 2 != 0 ? true : false;

        $"Valor de entrada: {ValorDeEntrada}\nValor da "
            +$"Variavel: {Variavel}".Dump("Valores gerados");
        //Exemplo da Reset Coil
        Coil_Reset.EN = ValorDeEntrada;
        "A correr exemplo da Reset Coil   -(R)-".Dump();
        Coil_Reset.ExecutarBloco(ref Variavel);
    }
}
```

Bloco de Código 29 - Bloco de código que corre para cada um dos estados para exemplificar o funcionamento da reset coil.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: False

A correr exemplo da Reset Coil   -(R)-
Valor da entrada do bloco antes de ser executado o bloco
False
Valor da Variável associada a este bloco antes de ser executado o bloco
False
Valor da saida do bloco antes de ser executado o bloco
False
A executar...
============================================================
Valor da entrada do bloco depois de ser executado o bloco
False
Valor da Variável associada a este bloco depois de ser executado o bloco
False
Valor da saida do bloco depois de ser executado o bloco
False
```

Resultado 17 - Resultado 1/4 do Bloco de Código 29.

```
Valores gerados
Valor de entrada: False
Valor da Variavel: True

A correr exemplo da Reset Coil  -(R)-
Valor da entrada do bloco antes de ser executado o bloco
False
Valor da Variável associada a este bloco antes de ser executado o bloco
True
Valor da saida do bloco antes de ser executado o bloco
False
A executar...
============================================================
Valor da entrada do bloco depois de ser executado o bloco
False
Valor da Variável associada a este bloco depois de ser executado o bloco
True
Valor da saida do bloco depois de ser executado o bloco
False
```

Resultado 18 - Resultado 2/4 do Bloco de Código 29.

```
Valores gerados
Valor de entrada: True
Valor da Variavel: False

A correr exemplo da Reset Coil  -(R)-
Valor da entrada do bloco antes de ser executado o bloco
True
Valor da Variável associada a este bloco antes de ser executado o bloco
False
Valor da saida do bloco antes de ser executado o bloco
False
A executar...
============================================================
Valor da entrada do bloco depois de ser executado o bloco
True
Valor da Variável associada a este bloco depois de ser executado o bloco
False
Valor da saida do bloco depois de ser executado o bloco
True
```

Resultado 19 - Resultado 3/4 do Bloco de Código 29.

```
Valores gerados
Valor de entrada: True
Valor da Variavel: True

A correr exemplo da Reset Coil  -(R)-
Valor da entrada do bloco antes de ser executado o bloco
True
Valor da Variável associada a este bloco antes de ser executado o bloco
True
Valor da saida do bloco antes de ser executado o bloco
True
A executar...
==========================================================
Valor da entrada do bloco depois de ser executado o bloco
True
Valor da Variável associada a este bloco depois de ser executado o bloco
False
Valor da saida do bloco depois de ser executado o bloco
True
```

Resultado 20 - Resultado 4/4 do Bloco de Código 29.

Outros elementos *Ladder Logic* importantes de referir

Apesar de estes 5 mencionados acima serem os elementos mais importantes, existem outros que vale a pena serem referidos, sendo eles (os nomes destes blocos podem variar de software para *software*):

- *One Shot Rising*: Permite que apenas que apenas seja ativa a sua saída com um pulso positivo da variável ou da entrada, desde que o outro já se encontre positivo;
- *One Shot Falling*: Permite que apenas que apenas seja ativa a sua saída com um pulso negativo da variável quando energizada a sua entrada;
- *TON*: É o equivalente a relé temporizado de atraso de energização, desta forma, este atrasa a energização de uma bobine ou das condições seguintes a este;
- *TOF*: É o equivalente a relé temporizado de atraso de desenergização, desta forma, este atrasa a desenergização de uma bobine ou das condições seguintes a este;
- *MOV*: Permite copiar valores de uma variável para outra. Estes blocos são usados principalmente em variáveis numéricas;
- Blocos de cálculos, como: ADD, SUB, MUL, DIV, etc. que permitem fazer cálculos;
- Blocos de comparação de variáveis: EQ, GE, GT, NEQ, LE, LT, etc. que permitem comparar duas variáveis.

Anexo 3 - Implementações em Strutured Text comparadas com C#

Este anexo pretende apresentar algumas comparações da linguagem de mais alto-nível padronizada pelo IEC 61131-3, o *Strutured Text*, com uma linguagem de alto-nível usado no mundo do IT, o *C#*.

Serão apresentadas diferenças entre as implementações de cada código para cada uma das linguagens e os problemas existentes na linguagem *Strutured Text*, presente também nas restantes 4 linguagens padronizadas nos PLC.

O Bloco de Código 30 e Bloco de Código 31 apresentam respetivamente a implementação de um código em ST e um código equivalente para *C#*. Estes códigos representam o exemplo de uma função que recebe dois parâmetros do tipo inteiro, soma-os e retorna o seu valor.

Exemplo em ST:

```
FUNCTION_BLOCK SomaDoisNumerosExemplo
    VAR_INPUT
        (*Lista com as variáveis de entrada*)
        Num1 : INT;
        Num2 : INT;
    END_VAR
    VAR_OUTPUT
        (*Lista com as variáveis de saída*)
        Resultado : INT;
    END_VAR

    (*Corpo da função*)
    Resultado := Num1 + Num2;
END_FUNCTION_BLOCK
```

Bloco de Código 30 - Exemplo de uma função de ST que soma dois números e retorna o resultado.

O equivalente em C# é a seguinte função:

```
int SomaDoisNumerosExemplo(int Num1, int Num2)
{
    return Num1 + Num2;
}
```

Bloco de Código 31 - Exemplo de uma função em C# que soma dois números e retorna o valor.

Desta forma, conseguimos entender que o ST é um pouco mais verboso de se escrever código, no entanto, nestes casos simples, consegue-se obter os mesmos resultados sem problemas.

No entanto, à medida que o código necessita de funcionalidades mais avançadas, nota-se uma grande diferença entre estas linguagens de programação.

Um exemplo disto, é caso se pretenda retornar a soma de todos os valores de um *array* do tipo *int*.

Para se realizar esta operação em ST de forma dinâmica, ou seja, sem se saber as dimensões do *array*, nem sempre é fácil, pelo que nos vamos focar em um software, sendo ele o *RSLogix 5000*, onde recorrendo à função *SIZE*, é possível obter o resultado do tamanho do *array*. São apresentados nos Bloco de Código 32 e Bloco de Código 33 as suas implementações.

```
FUNCTION_BLOCK SomaArrayExemplo
    VAR_INPUT
        (*Lista com as variáveis de entrada*)
        Numeros : ARRAY[*] OF INT;
    END_VAR
    VAR
        (*Declaração das variáveis locais*)
        tamanho : INT;
        index : INT;
        somaAux : INT;
    END_VAR
    VAR_OUTPUT
        (*Lista com as variáveis de saída*)
        Resultado : INT;
    END_VAR

    (*Corpo da função*)
    SIZE(Numeros[0], 0, tamanho); //Obtém o tamanho do array.
    //Loop para somar todos os valores
    FOR index := 0 TO tamanho BY 1 DO
        somaAux := somaAux + Numeros[index];
    END_FOR
    Resultado := somaAux;
END_FUNCTION_BLOCK
```

Bloco de Código 32 - Exemplo de código ST para que de forma dinâmica some todos os valores de um array e retorne o valor da soma.

Como podemos observar, é um pouco mais complicado à medida que o código exige uma maior dinâmica e flexibilidade, tendo sido, neste caso, necessário usar funções da livraria do *software* que nem todos os *softwares* disponibilizam. Isto é ainda mais fácil de notar quando olhamos ao código de *C#*.

```
int SomaArrayExemplo(params int[] Numeros)
{
    //Declaração das variáveis locais
    int Resultado = 0;
    //Loop para somar todos os valores
    foreach(int numero in Numeros)
        Resultado += numero;
    return Resultado;
}
```

Bloco de Código 33 - Exemplo de código C# a implementar a mesma operação realizada no Bloco de Código 32.

Estas demonstrações têm o propósito de mostrar que apesar de o ST ser uma linguagem de alto nível (dentro das cinco linguagens do IEC 61131-3), esta é relativamente mais complexa quando comparada com linguagens de programação de alto nível do mundo da informática.

De facto, os PLC ainda apresentam algumas dificuldades, em qualquer uma das cinco linguagens de programação padronizadas pelo IEC 61131-3, para criar um funcionamento dinâmico, flexível e que se ajuste de forma autónoma às informações que processa. A juntar a estes problemas, alguns IDE mais antigos, o seu compilador não avisava nem previne a compilação de programas que tenham erros críticos e que são detetáveis no momento da compilação. Exemplos destes erros podem ser a incompatibilidade de tipos, por exemplo, onde não é possível fazer um *cast* de forma implícita. Este erro será despoletado durante o *runtime*, fazendo com que o programa pare, muitas vezes sem grande informação do motivo da paragem. Isto pode obrigar o programador a fazer horas de *debug*, até que consiga identificar o erro.

Anexo 4 - Código desenvolvido para contabilizar número de conteúdo a ser alterado

```
using System;
using System.Collections.Generic;
using System.IO;

namespace ContarLinhasDeCodigo{
    class Program
    {
        static void Main(string[] args)
        {
            List<LinhaDeCodigo> numeroDeLinhasDeCadaCodigo = new List<LinhaDeCodigo>();
            Console.WriteLine("Programa para calcular o número médio de linhas de código
                                                                - Francisco Cardão\n");
            Console.WriteLine("A iniciar a contagem das linhas de código de cada
                                                                ficheiro...\n");
            string[] pathCodigoFontes = Directory.GetFiles(<PATH>, "*.SRC");
            foreach(string codigoFonte in pathCodigoFontes)
            {
                using(StreamReader ficheiro = new StreamReader(codigoFonte))
                {
                    while(!ficheiro.EndOfStream)
                    {
                        string linha = ficheiro.ReadLine();
                        numeroDeLinhasDeCadaCodigo.Add(new LinhaDeCodigo()
                        {
                            SouUmComentario = linha.StartsWith(';'),
                            NumeroDeZeros = (uint)linha.Split('0').Length - 1,
                            NumeroDeUns = (uint)linha.Split('1').Length - 1
                        });
                    }
                }
            }
            uint numeroZeros = 0, numeroUns = 0, numeroDeComentarios = 0;
            foreach(LinhaDeCodigo linha in numeroDeLinhasDeCadaCodigo)
            {
                numeroZeros += linha.NumeroDeZeros;
                numeroUns += linha.NumeroDeUns;
                if(linha.SouUmComentario)
                    numeroDeComentarios++;
            }
            double      media      =      (double)numeroDeLinhasDeCadaCodigo.Count      /
                                                        pathCodigoFontes.Length;
            Console.WriteLine("Número total de ficheiro de código fonte: " +
                                                        pathCodigoFontes.Length);
            Console.WriteLine("Número       total       de       linha:       "       +
                                                    numeroDeLinhasDeCadaCodigo.Count);
            Console.WriteLine("Número    total    de    linha    de    código:    "    +
                                (numeroDeLinhasDeCadaCodigo.Count - numeroDeComentarios));
            Console.WriteLine("Número    de    linhas    que    são    comentários:    "    +
                                                        numeroDeComentarios);
            Console.WriteLine("Número médio de linhas de código por ficheiro: " + media);
            Console.WriteLine("Número de zeros: " + numeroZeros);
            Console.WriteLine("Número de uns: " + numeroUns);
            Console.WriteLine("Total de zeros e uns: " + (numeroUns + numeroZeros));
            Console.WriteLine("Média de Uns e Zeros por linha: " + (double)(numeroUns +
numeroZeros) / numeroDeLinhasDeCadaCodigo.Count);
        }
        class LinhaDeCodigo
        {
            public bool SouUmComentario { get; set; }
            public uint NumeroDeZeros { get; set; }
            public uint NumeroDeUns { get; set; }
        }
    }
}
```

Bloco de Código 34 - Código que permitiu calcular o número de linhas presentes nos ficheiros *.src do controlador KR C1.

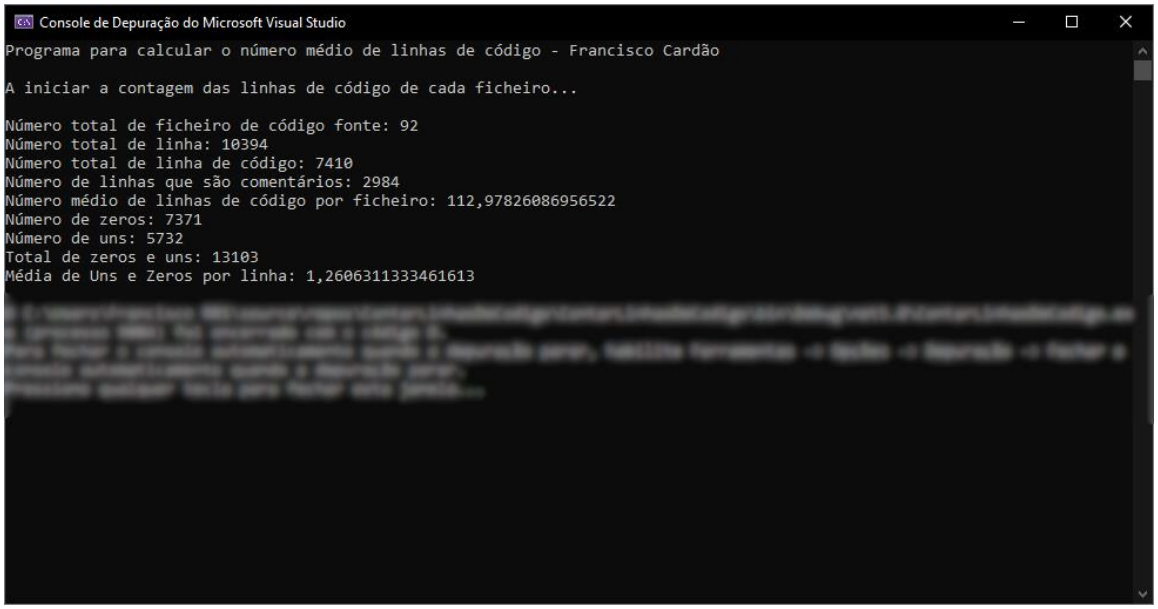


*Figura 99 - Resultados obtidos quando executado o Bloco de Código 34 para ficheiros do tipo *.SRC. Fonte: Própria*

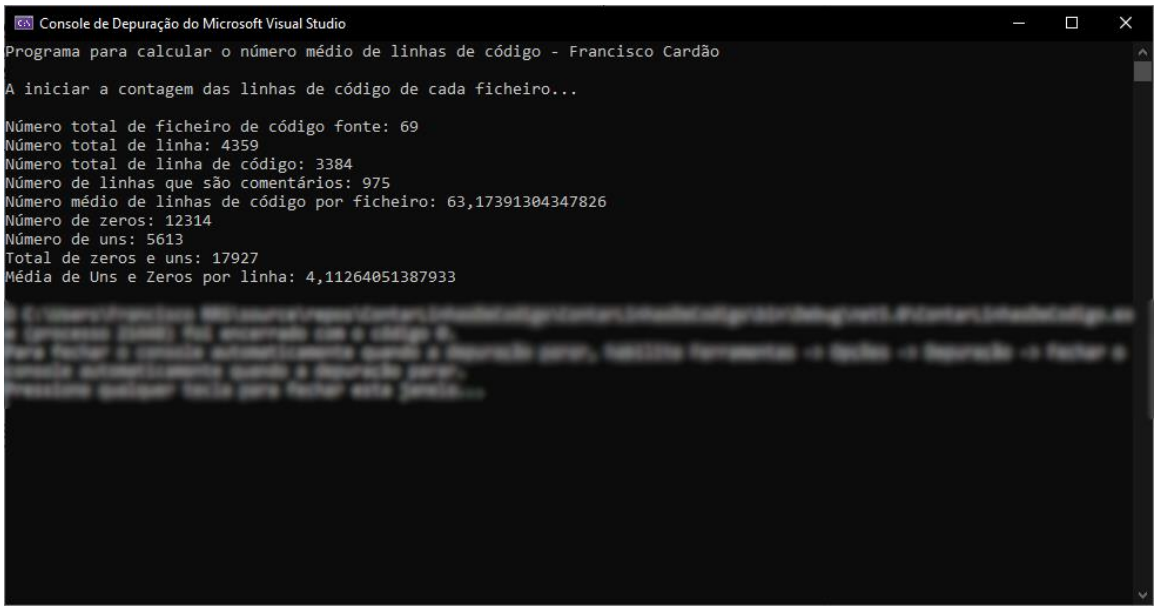


*Figura 100 - Resultados obtidos quando executado o Bloco de Código 34 para ficheiros do tipo *.dat. Fonte: Própria*