Faculdade de Engenharia da Universidade do Porto

Investigação Experimental de Técnicas de Reconhecimento de Padrões em Sistemas Automáticos

Tiago Granja da Silva

DISSERTAÇÃO

Dissertação no âmbito do Mestrado Integrado em Engenharia Mecânica
Secção de Automação, Instrumentação e Controlo

Orientador: Prof. Dr. António Pessoa de Magalhães

Agosto de 2010

Agradecimentos

Gostaria de agradecer principalmente ao meu orientador Prof. António Pessoa de Magalhães, pela paciência e disponibilidade demonstrada no decorrer do desenvolvimento deste projecto. Agradeço também aos restantes professores e colegas, com quem a convivência e discussão de diversos assuntos ao longo destes cinco meses permitiu a realização de um projecto mais consistente.

A nível mais pessoal, aqui deixo o meu agradecimento aos meus amigos de sempre, em particular à Bea, e por fim, mas não menos importante, quero aproveitar para agradecer à minha família pela presença e compreensão em todos os momentos da minha vida.

Resumo

O reconhecimento de formas ou padrões é um requisito primário para um considerável número de aplicações de automação industrial. A existência de Controladores Lógicos Programáveis normalmente associados ao controlo e monitorização dos processos industriais, justifica a pertinência do estudo da sua utilização na aquisição e análises de imagens obtidas por varrimento de um conjunto de sensores. O seu estudo permitiria identificar, testar e validar técnicas simples que sirvam o reconhecimento de formas e que possam ser facilmente implementáveis, a custos reduzidos, em equipamentos e ambientes industriais.

Este trabalho começa por fazer uma descrição do sistema e da forma como este adquire e forma a imagem (o espaço das características), seguindo-se um levantamento de possíveis aplicações industriais onde este possa ser utilizado e o estudo das diferentes abordagens e metodologias de sistemas de reconhecimento de padrões, em maior profundidade dos métodos estruturais por serem os que serão utilizados no desenvolvimento dos algoritmos. A etapa que se segue é a do desenvolvimento de uma plataforma de simulação, e sua utilização enquanto ferramenta de apoio à compreensão dos padrões em estudo. Por último procede-se à investigação experimental, implementando e testando os algoritmos desenvolvidos, seguindo a análise comparada dos resultados da simulação e da experimentação.

O Estudo apresentado reitera o valor da simulação combinado com um conjunto mais limitado de experiencias, permitindo extrair a influência real do ruído introduzido devido às diferentes superfícies de teste das peças como ao processo de aquisição de dados utilizado.

Abstract

The recognition of shapes and patterns is a essential tool for a considerable number of applications in the industrial automation field. The existence of Programmable Logic Controllers usually associated with the control and monitoring of the industrial processes, justifies for itself the study of its application in the acquisition and analyses of images obtained from a scanning done by a group of sensors. Studies in this area would identify, test and validate simple procedures able to recognize shapes and easy to implement, at a low cost level, in industrial equipments and environments.

This report starts to describe the system and the way its acquires and creates the image (characteristics space). Following, comes the identification of possible industrial applications where it can be used, as well as the analyses of different methods and different ways of managing the systems of patterns recognition, giving a particular emphases to the structural methods, as they will come to be the reference in the development of the algorithms. The next step is the creation of a simulation platform, underlining its role as an important tool in the understanding of the patterns in study. Lastly the experimental work is developed, implementing and testing the algorithms previously designed, and comparing closely the results obtained in the simulation field with those from the experimental work.

The presented study reinforces the valuable role of the simulation combined with a more limited group of experiences, allowing to realize the real influence of the interferences produced not only by the different testing surfaces but also from the process used for data acquisition.

Índice

Índice de tabelas

Índice de figuras

1. Introdução

Reconhecer imagens e formas, bem com a orientação e a localização de objectos, são necessidades sentidas na generalidade das indústrias produtivas e de serviços. Daí que, desde há longos anos, se faça investigação aplicada de técnicas de reconhecimento de padrões com fins industriais. O sucesso prático de tal investigação está bem patente nos inúmeros recursos tecnológicos actualmente disponíveis, permitindo assim que os resultados da pesquisa cheguem efectivamente a quem mais beneficia: os engenheiros de automação. Foi assim que o reconhecimento de padrões primeiro se introduziu e, depois, se generalizou, em grande parte das aplicações industriais. A maior evidência dessa evolução é a penetração a um ritmo exponencial das tecnologias de aquisição e tratamento de imagem nos sistemas flexíveis de automação industrial. De facto, são aparentemente cada vez mais raras as instalações automáticas que não incluem e não beneficiam de todo um sistema de aquisição de imagem.

Pese embora os computadores tenham a virtude de serem capazes de executar muitas tarefas complexas de uma forma muito mais rápida e eficaz do que os humanos, reconhecer objectos é, ironicamente, uma tarefa que os humanos realizam normalmente muito melhor do que os computadores. Daí que os sistemas industriais de reconhecimento de padrões alberguem tipicamente uma grande quantidade de segredos científicos e tecnológicos salvaguardados por diversas patentes e que, mesmo assim, não fosse a necessidade de libertar os operadores de tarefas rotineiras ou perigosas, teriam um interesse prático discutível em muitas aplicações.

Mas, seja com maior ou menor sucesso, e independentemente da aplicação, o facto é que os sistemas de reconhecimento de padrões são hoje actores fundamentais no teatro da automação industrial e dos serviços. E, neste contexto, os sistemas de aquisição de imagem baseados em elementos ópticos tais como câmaras fotográficas ou de vídeo, estão na base da generalidade das soluções. Todavia, tais soluções são normalmente muito fechadas para o utilizador final que apenas as pode configurar e integrar na aplicação de interesse e, ainda assim, normalmente com algum dificuldade. Adicionalmente, não raras vezes, a solução encontrada surge como excessiva

para o problema em causa, uma vez que acaba por ser utilizado um dispositivo relativamente sofisticado para resolver um problema aparentemente simples, mas para o qual não há aparentemente uma solução mais equilibrada. Resultado: o industrial acaba por adquirir um equipamento caro, e fica simultaneamente refém de uma tecnologia que não domina, arriscando-se assim, em caso de avaria, a ter a sua produção suspensa até que um técnico exterior lhe forneça o suporte que necessita.

A presente dissertação surge precisamente neste contexto: no sentido de procurar investigar até que ponto é possível satisfazer algumas necessidades industriais de reconhecimento de padrões com base em tecnologias e equipamentos simples, económicos, de utilização aberta e generalizada, e que qualquer engenheiro de automação domine e saiba integrar na aplicação de interesse. Assim, são eleitos os PLC's (Controladores Lógicos Programáveis) como elementos centrais de processamento de informação e os sensores industriais como meios de aquisição de imagem. A ideia é partir de um PLC que tem por missão o controlo de uma máquina e adicionar-lhe o software necessário ao reconhecimento em causa. Ao mesmo controlador terão também de ser necessariamente adicionados os sensores adequados à aplicação.

1.1. Conjectura e objectivos

O vasto leque de aplicações industriais de reconhecimento de padrões é claramente sinónimo de diferentes exigências técnicas. Por exemplo: o que é exigido a um sistema de reconhecimento de peças, distribuídas aleatoriamente num tapete, de movimento rápido e irregular, para que um sistema "*pick and place*" as possa recolher e seleccionar segundo determinados critérios (por exemplo, por ordem crescente de dimensões) é completamente diferente de projectar um sistema de inspecção que verifique se as peças, que são conduzidas de forma ordenada e a baixa velocidade para uma máquina, possuem ou não um furo no centro. É assim lícito conjecturar se algumas técnicas de reconhecimento de padrões ou de orientação de objectos podem ser conduzidas por PLC's, usando soluções de baixo custo e facilitando a integração com os processos de controlo das máquinas que correm nos mesmos equipamentos. O objectivo desta dissertação é precisamente investigar tal conjectura, conduzindo a investigação de duas formas: primeiro num ambiente de simulação onde os princípios teóricos aplicáveis ao reconhecimento de imagens são transportados para um PLC que trata sinais isentos de ruído, supostamente com origem em sensores comuns de baixo custo; depois por via experimental, verificando assim a viabilidade prática da investigação conduzida.

1.2. Organização da Dissertação

A dissertação está organizada em 6 capítulos e 4 anexos. No capítulo 2 começa-se por enquadrar e justificar a pertinência deste trabalho no contexto actual da automação industrial, abordando o conceito geral de reconhecimento de padrões e a sua inter-relação com diversas áreas científicas. Discute potenciais aplicações industriais concluindo, com a indicação de algumas hipóteses de resolução e linhas de investigação. O capítulo 3 dedica-se às técnicas de reconhecimento de padrões. Começa por dar uma visão geral do processo de reconhecimento de padrões e das diferentes abordagens existentes, focando-se depois nas técnicas estruturais, dado serem as que melhor se aplicam no presente caso. Retrata-se no capítulo 4, o estudo teórico, em particular o desenvolvimento de uma plataforma de simulação que permite o teste de diferentes algoritmos em condições ideais de ruído. No capítulo 5 expõem-se os resultados e conclusões dos algoritmos testados experimentalmente no sistema real. As principais conclusões e sumários da dissertação, assim como diversas sugestões para trabalhos futuros, são apresentados no capítulo 6. Em anexo apresenta-se informação adicional para a compreensão e avaliação do presente trabalho, nomeadamente: a programação de diversos algoritmos desenvolvidos.

2. Definição do problema

O reconhecimento de formas ou padrões é um requisito primário para um considerável número de aplicações de automação industrial. Por exemplo: para identificar se as peças que alimentam uma máquina têm uma determinada geometria ou estão devidamente orientadas. Um enorme leque de técnicas e tecnologias pode ser utilizado para esse fim, desde sistemas mecânicos elementares a sofisticados processo de visão artificial. Tudo depende da aplicação. É objecto desta dissertação identificar, testar e validar técnicas simples que sirvam o reconhecimento de formas e que possam ser facilmente implementáveis, a custos reduzidos, em equipamentos e ambientes industriais.

2.1. Justificação e formalização do problema

A existência de Controladores Lógicos Programáveis (PLC's), ver Hugh (2007), normalmente associados ao controlo e monitorização dos processos industriais, justifica a pertinência do estudo da sua utilização, e das tecnologias a si associadas, no reconhecimento de padrões. O sistema de varrimento proposto para a aquisição de dados para o PLC é constituído por um conjunto de sensores (neste caso: ópticos, do tipo difuso) dispostos em linha, perpendicularmente à direcção do movimento dos objectos.

Os sensores assim dispostos vão adquirindo faixas do perfil do referido objecto em instantes consecutivos (o número de pontos registados – pixel - é função do número de sensores instalados, da velocidade do movimento de varrimento e da frequência de registo pedida pelo operador) por forma a construir uma imagem a partir do agrupamento dessas amostras ("colunas") em memória pelo programa – figura 2.1.

A aquisição directa dos dados, no espaço e/ou no tempo, para efeitos de digitalização, pressupõe necessariamente uma distribuição espacial ou temporal descontínua. Por este motivo se designa por amostragem, a aquisição directa de dados nesse contexto.

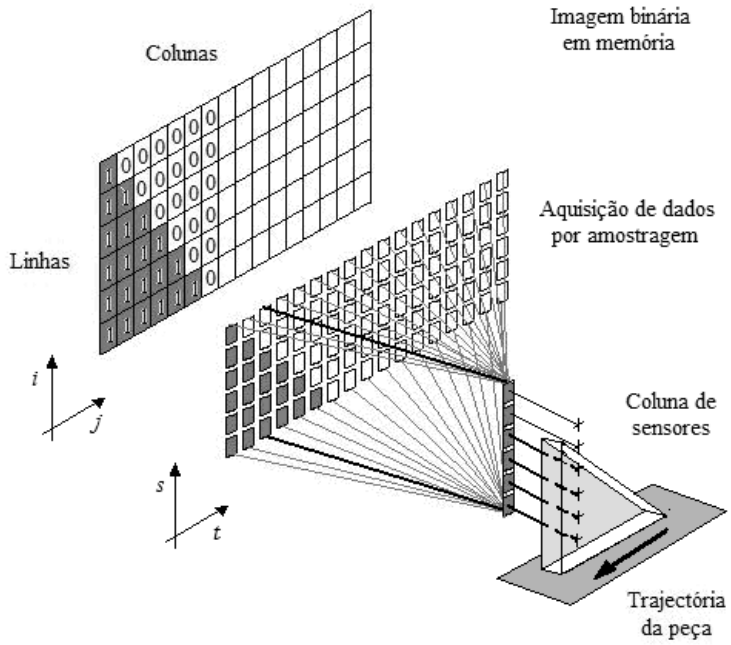


Figura 2.1. Construção da Imagem Binária (matriz binária bidimensional) pelo processo de amostragem por varrimento

A imagem digital, pode assim ser definida como um conjunto de pontos, onde cada ponto (pixel) corresponde a uma unidade de informação do objecto, formada através de uma função bidimensional *f(t,s)*, onde *t* e *s* são coordenadas espaciais (instante de aquisição e posição relativa do sensor) e o valor de *f* no ponto (*t,s*) representa o brilho ou radiância da área correspondente ao pixel. Tanto *t e s* (coluna e linha) quanto *f* só assumem valores inteiros, portanto, a imagem pode ser expressa numa forma matricial, onde a linha *i* e coluna *j* correspondem às coordenadas espaciais *t* e *s*, e o valor digital no ponto correspondente a *f*, é o nível de cinza do pixel daquele ponto (zero ou um – imagem binária).

O pixel (unidade indivisível) é a menor parcela considerada na imagem e a sua dimensão é denominada de resolução espacial, directamente relacionada com a distancia relativa entre os sensores, visível na figura 2.2. Por outro lado, relacionada com a faixa de valores numéricos associados aos pixeis, está a *resolução radiométrica*. Este valor numérico representa a intensidade da radiância proveniente da área correspondente ao pixel, também chamado de *nível de cinza*. No caso presente utilizamos apenas um bit para cada pixel, os sensores usados são sensíveis ao brilho, e são binários, em resumo: não há "cinzentos", só "pretos" e "brancos". Na figura 2.1. os pixéis a branco simbolizam bits a zero, e os pretos bits a um.

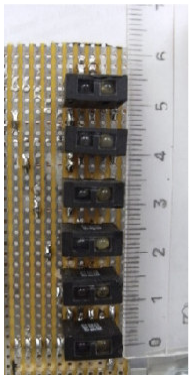


Figura 2.1. Coluna de sensores

A imagem assim obtida, tem uma resolução muito pequena, quando comparada com as imagens de uma câmara de um outro qualquer sistema de visão comercial. Contudo, e apesar de pouco explorada, esta solução pode ser muito interessante do ponto de vista custo/ benefício, numa gama específica de aplicações. A sua utilização poderia simplificar e reduzir os altos custos de implementação de um sistema de visão alternativo, tais como: 1) aquisição de componentes de hardware (câmaras, kits de iluminação, placas de aquisição de imagem, etc.); 2) software que acompanha o sistema e necessidade de compatibilização de protocolos; 3) exigência de mão-de-obra muito especializada; 4) custos de manutenção e atendimento.

Múltiplas são as aplicações industriais que carecem de sistemas de visão. Estas podem ser divididas em três sectores principais - ver Braggins (1996):

- Reconhecimento *(Recognition)*
- Orientação *(Robot and other guidance)*
- Inspecção *(Inspection)*

Estas três funções estão presentes em diversos sistemas onde a identificação de objectos e/ a sua orientação são necessários:

- Sistemas de controlo de qualidade
- Sistemas *pick and place*
- Sistemas de triagem (*Sortation Technologies and Solutions*)
- Sistemas de alimentação (*Feeding sytem, Feeder tooling*)
- Sistema de paletização

De imediato e intuitivamente, associamos alguns destes sistemas a aplicações de velocidades altas e variáveis, exigindo capacidades de processamento, em tempo real, muito elevadas, como são na maioria dos casos os sistemas pick and place ou os sistemas de alimentação e, por vezes, também os sistemas de triagem. Por outro lado os sistemas de

paletização e de controlo de qualidade funcionam em grande parte das vezes a velocidade baixa e constante. É principalmente nesta gama de aplicações que estará concentrada a atenção.

Sabendo-se, de antemão, que não será possível encontrar uma solução globalmente óptima, a investigação a desenvolver terá como principal propósito analisar a adequação das técnicas e métodos existentes às aplicações mais relevantes. Devido à simplicidade do sistema proposto (e sua pequena resolução) é fundamentalmente com base em características estruturais mais "evidentes" que será possível fazer-se a análise das imagens adquiridas.

Diferentes linhas de investigação serão exploradas: 1) serão investigados os diferentes métodos de reconhecimento de padrões, identificando aquele que mais se adecue ao sistema proposto; 2) será feito um levantamento das necessidades de reconhecimento de formas e padrões em aplicações industriais; 3) será desenvolvida uma plataforma de simulação com vista ao teste das técnicas a estudar em ambiente de investigação perfeito (sem ruídos ou com modelos de ruído definidos pelo utilizador); 4) proceder-se-à à investigação experimental com as contrariedades inerentes ao mundo real.

2.2. Estado da arte

A essência do reconhecimento de padrões é algo inerente à capacidade dos seres vivos em lidar com o meio envolvente, e é crucial para a sua sobrevivência - vantagem adaptativa – como o reconhecimento da presença de conceitos abstractos tais como: comida, perigo, acasalamento. De facto, este processo de "abstracção" ou "idealização" permite a um organismo lidar com novas situações de forma similar ao que foi visto como necessário em experiencias anteriores com situações do mesmo tipo. Deste modo, não é muito surpreendente a constatação de que os humanos são capazes de desempenhar muito fácil e eficazmente tarefas complexas de reconhecimento de padrões (até subconscientemente), mas que por outro lado possam ter muita dificuldade com, por exemplo, operações matemáticas simples de multiplicação.

O desenvolvimento de capacidades para "interpretar" o mundo que nos rodeia, dotando máquinas com "comportamentos inteligentes" é um dos grandes desafios neste inicio de século. Tal desafio resulta de os processos mentais serem obviamente complexos e mal conhecidos – ver figura 2.3. Neste contexto são múltiplas e importantes as relações do reconhecimento de padrões com várias disciplinas, em particular da área da Informática: processamento de sinal e imagem, inteligência artificial, aprendizagem automática *(machine learning)*, sistemas adaptativos, teoria dos autómatos, entre outras.

Múltiplas são também as abordagens ou metodologias que podem ser utilizadas para o efeito, tendo por objectivo "a busca da "estrutura" nos dados". A titulo de exemplo, e apesar de não existir ainda uma classificação unânime nesta matéria, pode referir-se, de entre os mais importantes, os métodos: estatísticos, estruturais e neuronais.

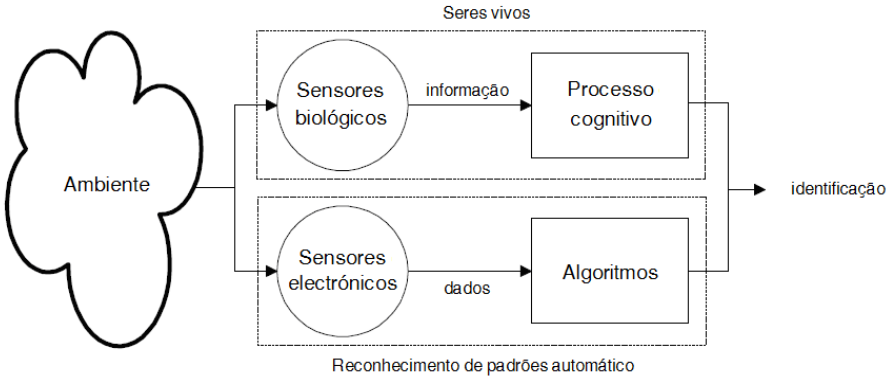


Figura 2.2. O problema de identificação resolvido por organismos vivos e sistemas automáticos de reconhecimento de padrões. Sistemas automáticos de reconhecimento de padrões utilizam sensores electrónicos e implementam algoritmos que aproximam o funcionamento de percepção e capacidades cognitivas dos organismos vivos, (Olszewski, 2001).

As principais áreas de aplicação industrial foram inicialmente o reconhecimento de caracteres, o controlo de processo, a análise de assinaturas e a análise de voz. Actualmente um vasto leque de aplicações pode ser identificado num grande número de áreas científicas e tecnológicas, nomeadamente as referidas na Tabela 2.1., ver Marques de Sá (2000).

Tabela 2.1. Diferentes áreas de aplicação de sistemas de reconhecimento de padrões

Científicas	○ Previsão sísmica, meteorológica, económica ○ Ciências da vida e do comportamento
Médicas	○ Estudos genéticos ○ Análise de electrocardiogramas, radiografias e tomografias
Agrícolas	○ Análise de solos e colheitas ○ Inspecção, ordenação e empacotamento do produto
Governamentais	○ Análise e controlo de tráfico ou crescimento urbano ○ Reconhecimento de pessoas
Militares	○ Análise de fotografia aérea ○ ATR – Reconhecimento automático de alvos

Numa perspectiva do interesse prático mais concreto da aplicação de sistemas de reconhecimento de formas e padrões do tipo preconizado na indústria "real", foi realizada uma breve pesquisa e identificadas algumas necessidades de sistemas deste tipo. Contudo, no decorrer da mesma, foi também possível reconhecer que na grande maioria das aplicações eram requeridos sistemas mais sofisticados de visão ou a combinação de um conjunto de diferentes tipos de sensores capazes de captar e medir diferentes características.

- Sistemas de controlo de qualidade

Na figura 2.4. um sistema de inspecção de embalagens de plástico, operado por um equipamento de visão. À passagem dos garrafões, uma câmara fotografa o gargalo do mesmo, utilizando a imagem assim obtida para a verificação da abertura. Dependendo do tipo de ocorrência, um sistema para este efeito mas baseado num varrimento de sensores do tipo capacitivo, justapostos poderá ser suficiente.

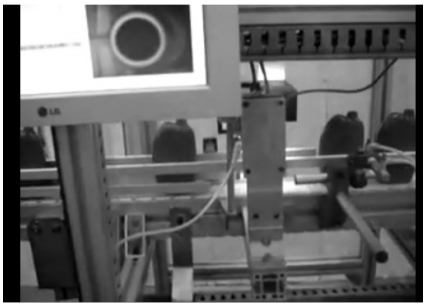


Figura 2.3. Sistema de Inspecção automática para garrafões de plástico. Verifica se a abertura do gargalo do garrafão está totalmente desimpedida

Na figura 2.5. é possível observar um sistema que assegura o transporte correcto do objecto, certifica que o mesmo não se encontra danificado. Em certos casos, onde a verificação de alguns requisitos de características mais "grosseiras" seja necessária, um sistema de varrimento por sensores, com um espaçamento e orientações bem definidos, pode ser capaz de executar a tarefa

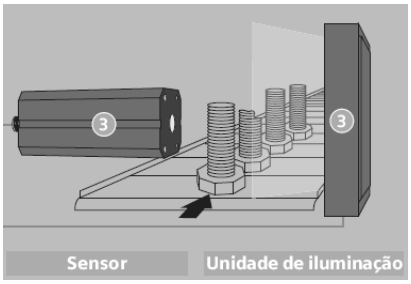


Figura 2.4. Sistema da Siemens constituído por uma unidade de controlo SIMATIC S7-200 com o analisador TP 170ª apoiados pelo SIMATIC VS 110 Vision Sensor. (http://www.siemens.com/microset)

- Sistemas de triagem (*Sortation Technologies and Solutions*)

Na figura 2.6., um sistema de transporte ou triagem, adquire a secção da forma (caixas no caso) pela colocação perpendicular de dois conjuntos de sensores relativamente ao movimento do tapete (duas das faces ficam na "sombra" – consideremos que a forma é simétrica). A existência de um terceiro conjunto de sensores disposto paralelamente à direcção do movimento permite adquirir o "comprimento" do objecto independentemente da velocidade do tapete. Este tipo de aplicação é perfeitamente enquadrável como o presente trabalho.

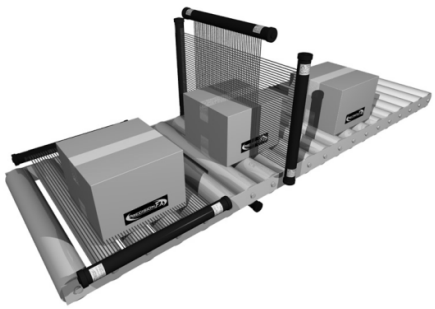


Figura 2.5. Sistema para separação de caixas de diferentes tamanhos e formas 3D utilizando um sistema Beam-Array (www.precisionfx.ca).

- Sistemas de alimentação (*Feeding sytem, Feeder tooling*)

Na figura 2.7. passa-se uma situação que embora pareça um pouco semelhante com a da figura 2.6. permite perspectivar algo bastante diferente. Dependendo do tipo de sensor será possível verificar apenas as dimensões segundo os seus eixos horizontal e vertical (sensores ópticos) ou as dimensões reais de ¾ da secção (sensores capacitivos).

Figura 2.6. Sistema de medição 2D do tamanho de um tronco de madeira (www.precisionfx.ca).

Na figura 2.8. é possível verificar que um sistema de varrimento acoplado ao sistema de alimentação de uma linha de produção permite verificar se determinada sequência de peças é cumprida, identificando para isso cada peça, através da aquisição da imagem do seu perfil à sua passagem.

Figura 2.8. Verificação de perfil de peças em cabine de pintura (www.bannerengineering.com).

- Sistema de paletização

Na figura 2.9. é utilizado um sistema de visão para verificar se uma palete de garrafas está completa. Poderá servir também para verificar se todas as garrafas da palete têm tampa. Este, parece ser um caso passível de ser solucionado pelo varrimento de um conjunto de quatro sensores capacitivos, que à medida da passagem da palete vão verificando se não existem "espaços vazios"

Figura 2.7. Sistema capaz de detectar a ausência de uma garrafa ou apenas a falta de rolha em alguma delas e, se possível, identificar a posição relativa da falha na embalagem.

2.3. Linhas de Investigação

As linhas de investigação e desenvolvimento a seguir, centram-se no desenvolvimento de uma plataforma de simulação, com vista ao teste das técnicas a estudar em ambiente de investigação perfeito (sem ruídos ou com modelos de ruído definidos pelo utilizador) e na investigação experimental com as contrariedades inerentes ao mundo real. Tendo por base os exemplos de aplicações referidas na secção anterior (da figura 2.4. à figura 2.9.) pode afirmar-se que é possível usar sistemas de "aquisição de formas" por PLC com sucesso em certas aplicações que envolvam a sensorização de formas simples, recorrendo a sensores ópticos, de proximidade, ou outros, para a obtenção da "imagem" da peça, pelo que este trabalho se centra na resolução de problemas de identificação e orientação de objectos desenvolvendo algoritmos de reconhecimento para um conjunto de perfis representativo.

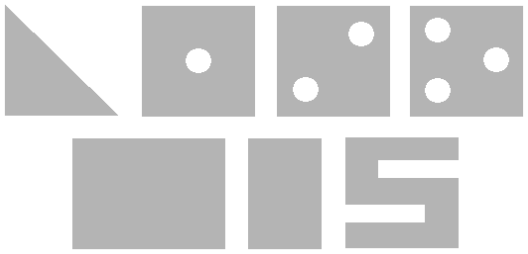


Figura 2.8. Perfis de peças adoptados para estudo

Com os casos de estudo referidas na figura 2.10. é possível analisar o comportamento do sistema na presença de: declives (derivadas), "ocos" internos, contornos externos varáveis e variações da forma (tamanho).

É neste contexto que surge o desenvolvimento da plataforma de simulação com o intuito de recriar o sistema real em condições ideais de funcionamento. O controlo do fenómeno ruído, que no sistema real é função do acabamento superficial e do processo de aquisição, permite desenvolver e testar algoritmos com desempenho óptimo para cada caso em estudo. A plataforma de simulação, tem por base um SoftPLC (da empresa 3s-software) compatível com a norma IEC 61131-3 (International Electrotechnical Commission (IEC), 2007), e servirá sobretudo de ferramenta de simulação de apoio à manipulação e análise dos padrões facilitando a sua compreensão e a extracção das suas características mais relevantes.

Por seu turno, a investigação experimental introduz as contingências do mundo real permitindo validar ou não os resultados obtidos na simulação. Assim será testada a robustez dos algoritmos, desenvolvidos na plataforma de simulação, para modelos de ruído real introduzidos por dois factores principais: diferentes tipos de acabamentos superficiais que serão testados e o modo de aquisição de sinal: *"event driven"* (aquisição de sinal inibida por um sinal de relógio) ou *"time driven"* (apenas conduzida pela ocorrência de eventos discretos).

2.4. Síntese

São inúmeras as aplicações industriais que necessitam de sistemas de visão. Apesar disso, o sistema que se pretende desenvolver apenas é viável para um conjunto reduzido mas não menos importante e necessário de aplicações mais simples com requisitos de velocidade baixa e não variáveis, onde não se justifique a utilização de outros meios mais sofisticados e necessariamente mais caros.

A obtenção da imagem por varrimento permite adquirir a informação do perfil da peça transferindo-a directamente para armazenamento sob a forma de uma imagem binária no PLC. Esta imagem "descontínua" fica, assim, directamente disponível para análise. Diferentes metodologias de desenvolvimento de algoritmos podem ser tidas em consideração. A sua escolha será em função das capacidades inerentes ao PLC bem como da aplicação em concreto.

O desenvolvimento de uma plataforma de simulação bem como a verificação da validade dos resultados aí obtidos, através de uma investigação experimental, são passos importantes a dar para a validação do modelo de sistema proposto.

3. Reconhecimento de Padrões

O termo "reconhecimento de padrões" referindo-se *ao "uso de computadores para realizar algo que os seres humanos fazem instintivamente: reconhecer uma "mensagem" num sinal"* foi introduzido no início da década de 60, ver Dunn (2008). Mais tarde, é sugerida por Duda e Hart (1973) uma definição, que continua a ser actualmente bem aceite, referindo-se ao reconhecimento de padrões como sendo o *"campo interessado no reconhecimento de regularidades significativas em ambientes ruidosos ou complexos, feito por máquinas"* ou, mais sinteticamente, como a *"procura de "estrutura" nos dados"*. Neste enquadramento dever-se-á, por isso, ter presentes os seguintes conceitos (Grivet, 2002):

- Dado - entendido como a saída de virtualmente qualquer processo físico, quer se origine na natureza ou outro ambiente específico;
- Estrutura – enquanto modo pelo qual a informação pode ser organizada de forma que as relações entre as variáveis do processo possam ser identificadas;
- Padrões – definidos como representações "físicas" de objectos. Normalmente sinais, imagens ou simples tabelas de valores. Podem frequentemente ser referidos como objectos, casos ou amostras.

De uma forma geral, pode dizer-se que o processo de reconhecimento de "regularidades" num padrão engloba três fases principais: a representação dos dados de entrada e sua mensuração; a extracção das características e, por último, a classificação do objecto em estudo.

- Na primeira fase - antes que o reconhecimento ocorra, o padrão deve ser "sentido" e "medido" pelos órgãos sensores (recolha de observações de "padrões característicos"), sendo processado e segmentado quando necessário.
- Na segunda fase - dá-se a extracção das características dos objectos a classificar, seguido da selecção de algumas das mais discriminantes, com o objectivo de reduzir o Espaço de Medida, sem perda de generalidade, a uma representação no Espaço das Características.

- A terceira e última fase, envolve a determinação de procedimentos que possibilitem a identificação e classificação do objecto numa classe de objectos - processo de construção de um Classificador.

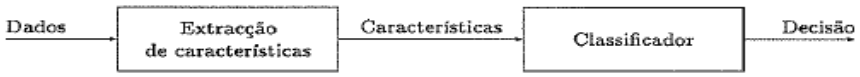


Figura 3.1. Estrutura clássica de um "reconhecedor" de padrões.

Destas três actividades, esquematizadas na figura 3.1., a extracção das características é a mais crítica pois a disponibilização de características discriminantes influenciam directamente a eficácia da tarefa de classificação. Desenvolvendo alguns conceitos acima referidos, temos que:

- Características (*Feature*) - são medidas, atributos ou primitivas, derivados dos padrões, que podem ser relevantes para a sua classificação. A escolha das características mais adequadas é mais uma arte que uma ciência.
- Espaço de Medida (*Measurement Space*) - é o espaço no qual o padrão é inicialmente representado. Este espaço é em geral de alta dimensionalidade.
- Espaço das Características (*Feature Space*) ou Espaço de Representação - retém as propriedades dos dados, de acordo com as medidas de semelhança definidas. É um espaço de dimensionalidade reduzida.

O classificador é um dispositivo que implementa uma função de decisão executando uma sequência de operações lógicas e computacionais sobre o padrão. Esta sequência é denominada de algoritmo de reconhecimento e o grupo de ideias subjacentes é denominado de método de reconhecimento

3.1. Metodologias

Uma das questões centrais do reconhecimento de padrões prende-se com a metodologia utilizada no reconhecimento das características e dos atributos de um objecto, ou seja, no sistema de classificação implementado. Iremos por isso, procurar apresentar de forma breve as abordagens mais representativas para um entendimento sobre o enquadramento geral do reconhecimento de padrões: métodos estatísticos, métodos estruturais, redes neuronais, conjuntos difusos.

Não existe um consenso geral quanto à classificação das diferentes abordagens e técnicas. Na figura 3.2. pode ver-se a relação das diferentes técnicas de reconhecimento de padrões com o

tipo de supervisão bem como com a solução de mapeamento no espaço proposto por Marques de Sá (2001).

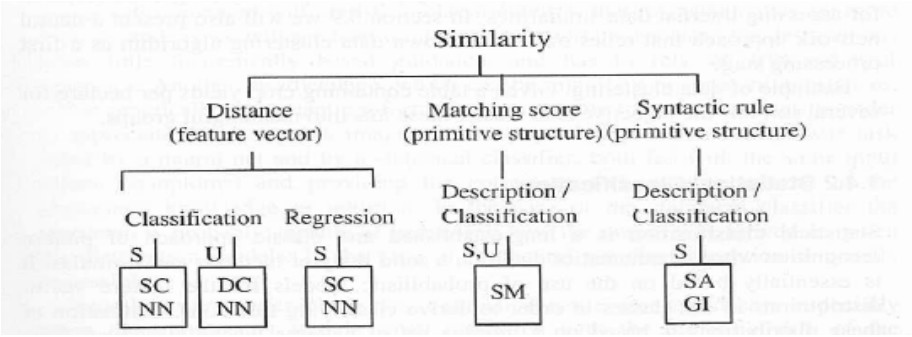


Figura 3.2. Abordagens RP: S - supervisionada; U – não supervisionada; SC – classificação estatística; NN – redes neurais; DC – data clustering; SM – correspondência estrutural; SA – análise sintáctica; GI – inferência gramatical

Para um melhor enquadramento do desenvolvimento destas diferentes abordagens poderão ser encontrados alguns marcos importantes na história do reconhecimento de padrões em Marques de Sá (2000).

3.1.1. Métodos Estatísticos

Historicamente mais antigos, os métodos **Estatísticos** foram dominantes entre os anos de 1965 – 1975 (Pavlidis, 2003). Tida como a "abordagem clássica", é também conhecida por "Teoria da Decisão", ver Howson e Urbach (1993). Assume que as características das classes se regem por determinados modelos probabilísticos.

Se o objecto foi mapeado num vector x (de dimensão d), então a estimação da probabilidade de um objecto pertencer a uma dada classe, depois de observada uma determinada característica, é muito interessante no contexto do reconhecimento de padrões. Esta probabilidade pode ser estimada pela fórmula de Bayes, se um conjunto de objectos com classificação conhecida estiver disponível:

$$P\left(\frac{w}{x}\right) = \frac{P\left(\frac{x}{w}\right) \bullet P(w)}{P(x)} \qquad (1)$$

onde $P(w)$ – probabilidade, à priori, da classe w; $P(x/w)$ - função densidade de probabilidade (condicionada) de ocorrência da característica x, dado a classe w; $P(x)$ - probabilidade de

ocorrência da característica *x*; *P*(*w/x*) - probabilidade à posteriori (probabilidade do objecto pertencer á classe *w*, dado a característica *x*).

Dentro dos métodos estatísticos, podem admitir-se dois grandes blocos de técnicas em função dos dados do objecto que se possuem: nos métodos paramétricos, assume-se que a função densidade de probabilidade ou uma família destas, têm um determinado tipo de distribuição, não se conhecendo, apenas, o valor dos seus parâmetros. A estimativa destes parâmetros pode ser feita recorrendo ao método de máxima verosimilhança, ver Mardia (1979), ao método do algoritmo EM, Dempster (1977), ou recorrendo à estimação baseada na teoria da informação e da estimação baysiana, ver Fukunaga (1990).

Já nos métodos não paramétricos, não é definida especificamente a distribuição da densidade de probabilidade, o que, por um lado, os torna métodos mais gerais mas com necessidade de maior número de dados e com resultados piores comparativamente com os métodos paramétricos. Exemplos de métodos não paramétricos são: o método Parzen, Fukunaga (1993), o método k vizinhos mais próximos (KNN), McQueen (1967), ou a curva ROC – *Receiver Operating Characteristic*, ver Everson e Fieldsend (2006).

No final dos anos 70 tornava-se evidente que o reconhecimento de padrões estatísticos não devia ser "levado ao limite". A natureza quantitativa de reconhecimento de padrões estatísticos, torna difícil a discriminação entre os grupos com base na morfologia (shapebased ou estrutural). Esta limitação impulsiona o desenvolvimento de uma abordagem estrutural para reconhecimento de padrões.

3.1.2. Redes Neuronais

Uma rede neuronal é um modelo de processamento de informação inspirado no comportamento do cérebro humano. Por oposição à arquitectura de Von Neumann, o cérebro realiza operações de forma paralela e distribuída. As redes neuronais artificiais são assim constituídas por elementos de processamento paralelo simples, interligados, ver Rojas (1996). O elemento chave deste paradigma é então o "neurónio", constituído por três elementos básicos (figura 3.3.):

- um conjunto de sinapses, que são conexões onde um sinal x_j na entrada da sinapse j conectada ao neurónio k é multiplicado pelo peso sináptico w_{kj};
- um somador para somar os sinais de entrada, ponderados pelas respectivas sinapses do neurónio;
- uma função de activação para restringir a amplitude da saída do neurónio (função de limiar).

O modelo neuronal inclui também um *bias* que tem por função aumentar ou diminuir a entrada da função de activação (dependendo se é positivo ou negativo), ver Haykin (1999).

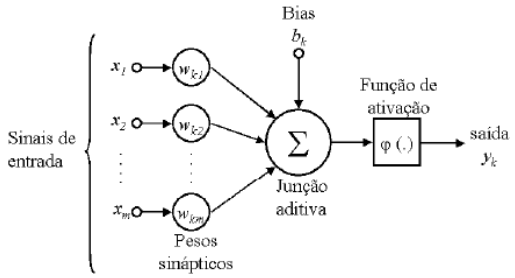


Figura 3.2 Modelo de um neurónio de uma rede neuronal

As principais vantagens do uso de redes neurais sobre muitas técnicas tradicionais de reconhecimento de padrões são a sua:

- Adaptabilidade: aptidão para se ajustar a novas informações;
- Velocidade: devido ao paralelismo de processamento;
- Tolerância a falhas: capacidade de oferecer boas respostas mesmo com falta, confusão ou dados ruidosos;

3.1.3. Conjuntos Difusos

É uma abordagem recente, que tem em consideração o grau de incerteza por vezes inerente a características e a classificações, usando para isso a teoria dos conjuntos difusos. O conceito de conjunto difuso foi introduzido, por Lotfi (1965) e uma das suas principais potencialidades, quando comparada com outros esquemas que tratam com dados imprecisos como redes neurais, é que as suas bases de conhecimento, as quais estão no formato de regras de produção, são fáceis de examinar e entender, tornando assim fácil a sua manutenção e a actualização.

3.2. Metodologias Estruturais

Tipicamente, esta abordagem procura descrever como um padrão mais complexo pode ser interpretado enquanto uma organização de padrões mais simples. Formula uma descrição da estrutura (morfologia) dos padrões usando as inter-relações, tipicamente hierárquicas, entre as características descritoras básicas presentes nos dados, denominadas primitivas.

As primitivas são elementos simples e bem definidos, cuja escolha depende da aplicação, podendo ser de diferente natureza, nomeadamente: um sinal (decomposto em segmentos de linha ou outras curvas simples, seguindo-se a sua rotulagem segundo uma determinada regra), uma imagem (como no presente caso onde as primitivas podem ser obtidas por um conjunto largo de técnicas de análise de imagens: segmentação da imagem, detecção de arcos, seguimento de linhas de contorno, etc), entre outras representações possíveis.

Para além de não existir uma abordagem geral que diga quais as primitivas que devem ser extraídas, a tarefa de classificação de um sistema de reconhecimento de padrões estruturais também é difícil de implementar, uma vez que as gramáticas sintácticas incorporam critérios precisos que discriminam entre classes e, portanto, são pela sua natureza e domínio, específicos para cada aplicação. Por exemplo, Friedman, (1999, p. 243) referindo-se à questão, diz: *"The selection of primitives by which the patterns of interest are going to be described depends upon the type of data and the associated application."* Nadler, (1993, p. 152) parece apoiar esta posição ao afirmar, *"...features are generally designed by hand, using the experience, intuition, and/or cleverness of the designer."*

A representação e qualificação estrutural é usualmente feita recorrendo ao uso de gramáticas formais e descrições relacionais. Neste contexto abordaremos particularmente o recurso a strings e à teoria dos grafos. Alguns autores, como J.P. Marques de Sá, subdividem as técnicas estruturais em dois tipos principais:

- A análise sintáctica - baseada no uso da teoria da linguagem formal;
- Métodos de correspondência estrutural (*"strutural matching"* - baseados em relações entre as primitivas – onde se destacam o *"string matching", "relaxation matching"* e o *"graph matching"*). Na verdade, este método de correspondência estrutural é mais baseado na observação de senso comum em que os problemas devem ser primeiro entendido antes que eles possam ser resolvidos. Não fornece um quadro analítico sistemático como o oferecido pelas abordagens estatística e de análise sintáctica.

A utilidade de sistemas de reconhecimento estrutural de padrões, no entanto, é limitada como consequência da complexidade das tarefas de descrição e classificação. Estas são difíceis de implementar, dado que as gramáticas sintácticas precisam de ter definidos critérios precisos relativos aos grupos discriminantes e, portanto, são por sua própria natureza do domínio específico de cada aplicação.

3.2.1. Linguagens e Palavras

Começaremos por abordar em maior detalhe uma das representações acima referidas: as "strings". Considere-se um conjunto *E,* não vazio, o "alfabeto", e chamem-se "letras" (símbolos ou "eventos") aos elementos que o constituem.

$$E = \{a, b, c\} \tag{2}$$

Defina-se uma *"string"* (ou "palavra") *w,* no ou sobre o conjunto *E,* definido em (2), como uma sequência finita ordenada de letras de *E (*ex.: $w = accbaaa = ac^2ba^3$*)*, onde cada letra representa uma primitiva. O comprimento de uma palavra *w*, escrito $|w|$, corresponde ao número de elementos da sua sequência de letras. Para o exemplo dado, temos $|w| = 7$. A palavra de *E* que não possui letras, é designada de palavra vazia, e é representada por λ, o comprimento $|\lambda| = 0$. Ao conjunto de todas as palavras de *E* é atribuída a designação *E** (onde * se denomina por Fecho de Kleene, ver Teorema de Kleene, (2010)).

Neste contexto, a concatenação é a operação (associativa) fundamental para a representação de estruturas por palavras. Considerem-se duas palavras *u* e *v* no alfabeto *E*. A concatenação de *u* e *v*, escrito *uv*, é a palavra obtida escrevendo as letras de *u* seguidas das letras de *v*. A palavra vazia λ é um elemento neutro para a operação. Qualquer palavra contígua que seja parte de outra palavra é chamada de factor (ou sub-palavra, ou segmento) dessa palavra. Em certos casos, torna-se necessária a utilização de palavras com um vector associado de atributos (duração, comprimento, cor, etc).

3.2.1.1. Método da correspondência estrutural (strutural matching)

A sua utilização em strings é baseada na determinação/medição da similaridade entre duas palavras, e é definida em termos do número de operações elementares necessárias para transformar uma palavra *u* numa palavra *v*. Importantes problemas de reconhecimento de padrões que envolvem cálculos em palavras incluem:

- "String matching" (a correspondência de palavras): dada a palavra *u* e um texto (palavra longa constituída por sub palavras), determinar se *u* é um factor do texto e, em caso afirmativo, onde ele aparece. O problema consiste em descobrir se existe um shift *s* válido onde exista uma correspondência perfeita entre cada letra da palavra *u* e a correspondente no texto;

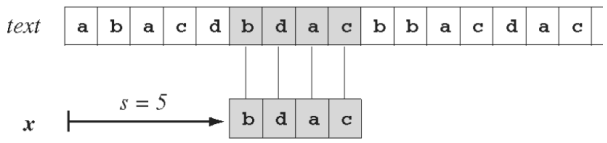


Figura 3.3. String matching (Aksoy, 2008).

- "String matching with errors" (a correspondência de palavras com erros): Dada a palavra *u* e texto, encontrar as posições no texto onde a "distância" de *u* para qualquer factor do texto é mínima.
- A correspondência de palavras com o símbolo "don't care", "String matching with the "don't care" symbol" é formalmente o mesmo algoritmo da "strig matching", mas onde o Ø tanto em *u* como no texto é dito corresponder a qualquer outra letra.
- "String edit distance" (edição da distância de palavras): Dadas duas palavras *u* e *v*, calcular o número mínimo de operações básicas: substituir, inserir ou eliminar (medida da distancia entre palavras) necessário para transformar *u* em *v*.

3.2.1.2.Análise sintáctica

A análise sintáctica é a abordagem dos métodos estruturais baseada na teoria das linguagens formais. Esta abordagem possui um conhecimento já bem desenvolvido, mas também algumas desvantagens inerentes, nomeadamente a insuficiente capacidade de representação para padrões complexos, e a necessidade de definir à priori uma gramática para cada problema em particular.

Nesta abordagem a gramática é o factor determinante e constitui-se como o modelo detalhado que está por detrás da geração da sequência de letras nas palavras. Um exemplo representativo é a estrutura "gramatical" restritiva que as palavras representando números de telefone possuem. Da mesma forma, sistemas de reconhecimento de orientações e formas de objectos, podem reconhecer e interpretar regras que condicionam o posicionamento dos símbolos.

No reconhecimento de padrões, dada uma frase (uma "string" gerada por um conjunto de regras) e uma gramática (o conjunto de regras), procura-se determinar se a frase foi gerada por essa gramática.

Uma Gramática é composta por quatro partes (Lipson e Lipschutz, 2007):

(1) Os símbolos - conjunto finito de letras (primitivas) de um alfabeto *E*. Uma linguagem é um subconjunto de $E \cup \{\lambda\}$ constituído por palavras obedecendo a certas regras.

(2) As variáveis - subconjunto *T* de *E*, cujos elementos são chamados de "terminais", e os elementos de *V* também chamados de "não-terminais" ou "variáveis".

(3) O símbolo de começo - símbolo não-terminal *S*.

(4) Um conjunto finito *P* de regras sintácticas, "regras de produção", usadas para gerar palavras (especifica como transformar um conjunto de variáveis e símbolos em outras varáveis e símbolos). A produção é um par ordenado (α, β), geralmente por escrito $\alpha \rightarrow \beta$, onde α e β são palavras em *E*, e a produção deve conter pelo menos um não-terminal α no seu lado esquerdo. O conjunto de regras P determina a linguagem *L(G)* (ou simplesmente *L*) - o subconjunto de palavras de *E* admitidas pelas regras.

Essa gramática *G* é denotada por *G (E, T, S, P),* quando queremos indicar as suas quatro partes. A classificação gramatical que se segue é devida a Noam Chomsky. As gramáticas são classificadas de acordo com os tipos de produção que são permitidos. A gramática tipo 0 não tem restrições nas suas regras de produção, e como tal não fornecem restrições ou estrutura às palavras que podem produzir. Os tipos 1, 2 e 3 são definidos como segue:

1. Uma gramática G é dita sensível ao contexto (de tipo 1) se as regras de produção são da forma $\alpha \rightarrow \beta$, onde $|\alpha| \leq |\beta|$ ou da forma $\alpha \rightarrow \lambda$.
2. Uma gramática G é dita ser de tipo 2 ("context-free), se toda a regra de produção é da forma $E \rightarrow \beta$, onde o lado esquerdo *E* é um não-terminal e o lado direito pode ser um não-terminal ou um terminal.
3. Uma gramática G é dita ser do tipo 3 (ou regular) se as regras de produção são da forma $E \rightarrow a$ ou $E \rightarrow$ aB, ou seja, onde o lado esquerdo *E* é um não terminal único e o lado direito é um terminal simples ou um terminal seguido de um não-terminal, ou da forma $S \rightarrow \lambda$.

As gramáticas de tipo 3 são também chamadas de gramáticas regulares.

Define-se uma linguagem *L* sobre um alfabeto *E* como um qualquer conjunto de palavras em *E*. Dado que *E** denota o conjunto de todas as palavras de *E*, pode dizer-se que uma linguagem *L* é simplesmente um subconjunto de *E**. E como vimos anteriormente, a linguagem *L(G)* gerada pela gramática *G* é o conjunto de todas as palavras (de *E*, possivelmente infinito em número) que pode ser gerado por G. Por outro lado, uma Linguagem diz-se Regular se for constituída por expressões regulares *r*. Cada uma das seguintes, é uma expressão regular sobre um alfabeto *E (não vazio)*:

1. O símbolo *"λ"* (palavra vazia) e o par "( )" (expressão vazia) são expressões regulares de comprimento *1*;

2. Cada letra de E é uma expressão regular;
3. Se r é uma expressão regular, então r^* é uma expressão regular;
4. Se r_1 e r_2 são expressões regulares, então $(r_1 \vee r_2)$ é uma expressão regular;
5. Se r_1 e r_2 são expressões regulares, então $(r_1\ r_2)$ é uma expressão regular;

Todas as expressões regulares são formadas desta forma. É importante reter que uma expressão regular r é um tipo especial de palavra, que utiliza as letras de E e os cinco símbolos: ( ) * $\vee$ λ. Nenhum outro símbolo é usado em expressões regulares.

Seja L uma linguagem sobre E. Então L é chamada de linguagem regular sobre E, se existe uma expressão regular r sobre E tal que $L = L(r)$. Um Autómato, é o dispositivo capaz de gerar uma determinada linguagem de acordo com determinadas regras gramaticais. A relação fundamental entre linguagens regulares, autómatos finitos e gramáticas regulares, pode ser compreendida como se segue:

- Como visto anteriormente uma linguagem L é regular se e só se $L=L(r)$ onde r é uma expressão regular,
- Por outro lado, pelo Teorema de Kleene, tem-se que: uma linguagem L sobre um alfabeto E é regular se e somente se existe um autómato finito M tal que $L=L(M)$.
- Por ultimo, M é um autómato finito se e só se $L=L(G)$ onde G é uma gramática regular (Teorema: uma linguagem L pode ser gerada por uma gramática regular (tipo 3), se e somente se existir um autómato finito M que aceite L.)

Como tal, podemos dizer que um Autómato é um dispositivo que observa palavras de um alfabeto E e reúne um conjunto (finito) X de estados, onde $x_0 \in X$ denota o estado original. Um autómato é capaz de gerar uma determinada linguagem, de acordo com a seguinte regra: $x'= f(x,e)$, onde: x - estado actual; e – evento observado; x' – estado seguinte. Uma palavra w é reconhecida (falada) por um autómato $M = (E, X, f, x_0, F)$ se $f(x_0,w) = x$ com $x \in F$, onde:

1. Um conjunto finito E (alfabeto) de entradas.
2. Um conjunto finito de X (interno) estados.
3. Um subconjunto F de X (estados finais).
4. O estado inicial x_0 em X.
5. A função de transição de estados $f : X \times E \rightarrow X$

Diagrama de transição de um Estado M: um autómato *M* é geralmente definido por meio do seu estado diagrama *D = D(M)* em vez de listar as suas cinco partes. O diagrama de estado *D = D(M)* é um grafo caracterizado da seguinte forma:

1. Os vértices de *D(M)* são os estados em *X* e um estado de aceitação é indicado por meio de um círculo duplo.
2. Há uma seta (aresta direccionada) em *D(M)* do estado x_j para o estado x_k marcada por uma transição por $f(x_j,a)= x_k$ ou, equivalentemente, se $fa(x_j) = x_k$.
3. O estado inicial x_0 é indicado por meio de uma flecha especial que termina em s0, mas não tem vértice inicial.

Diz-se que *M* reconhece a palavra *w* se o estado final x_m é um estado de aceitação em *F*. A linguagem *L(M)* de *M* é o conjunto de todas as palavras de *E* que são aceites por *M*. Ou seja, a Linguagem L(M) é determinada pelo Autómato M.

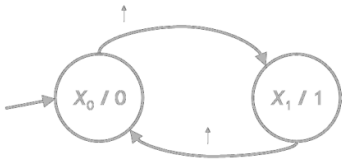


Figura 3.4. Diagrama de Estados de um Autómato

O reconhecimento utilizando gramáticas pode ser feito supondo que nos é dada uma palavra teste *u* gerada por uma de c gramáticas diferentes G_1, G_2, …, G_c, que podem ser consideradas como modelos ou classes diferentes. A palavra teste *u* é classificada de acordo com a gramática que lhe poderá ter dado origem, ou equivalentemente, de acordo com a linguagem $L(G_i)$ da qual *u* é membro.

Exemplos de gramáticas (Aksoy, 2008):

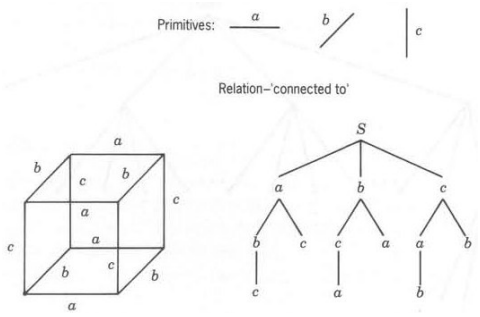


Figura 3.5. Árvore de representação de um cubo baseada numa gramática.

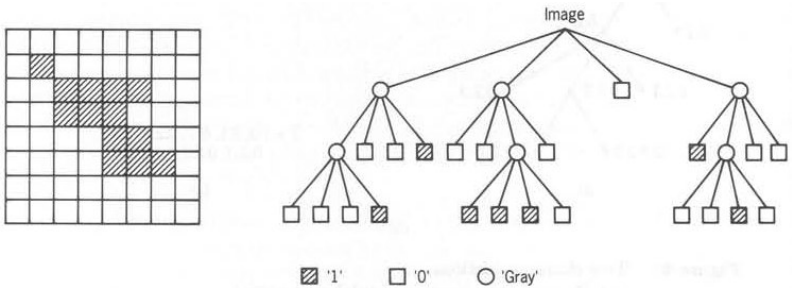


Figura 3.6. Árvore de representação de uma imagem binária 8 × 8.

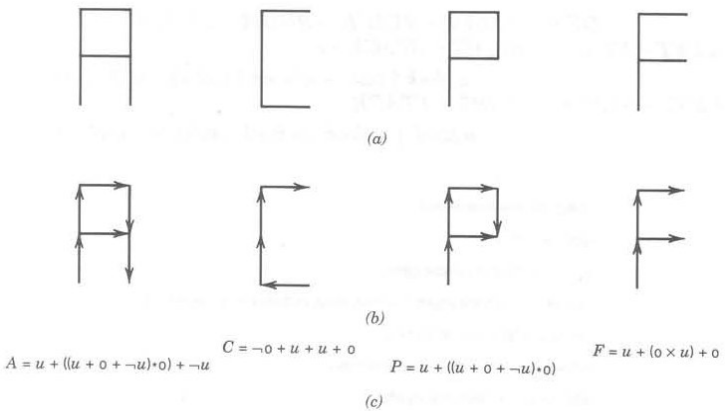


Figura 3.7. Representação de quatro caracteres através da gramática que descreve a figura pelo desenho de linhas de contorno (a) Padrão dado. (b) Representação das primitivas e sua interconexão (c) descrição.

Como vimos nos exemplos acima referidos, uma gramática pode ser representada de várias formas. Seguidamente iremos abordar os grafos com um pouco mais de detalhe.

3.2.2. Grafos

A Teoria dos Grafos é um ramo da matemática que estuda as relações entre os objectos de um determinado conjunto. Um Grafo G ={N, R} é um par ordenado representado por:

- Um conjunto de nós (vértices), *N*,
- Um conjunto de arestas (arcos), $R \subseteq N \times N$.

Dependendo da aplicação, as arestas podem ou não ter direcção (i.e. a ordem pela qual a entidade surge no par é significante), pode ser ou não permitido as arestas ligarem um nó a ele próprio e os nós e/ou arestas podem ter um peso (numérico) associado.

Um grafo orientado (também chamado de grafo direccionado, dígrafo ou quiver) é dado por uma função do tipo $s, t : R \to N$, onde $s(e)$ é a *fonte* e $t(e)$ é o *alvo* da aresta direccionada e. Por outro lado um grafo não orientado (ou simplesmente grafo) é dado por uma função $w : R \to P(N)$ que associa a cada aresta um subconjunto de dois ou de um elemento de N, interpretado como os pontos terminais da aresta.

3.3. Hipóteses de aplicação no presente caso

Como foi referido as características são determinantes no reconhecimento de padrões, e a obtenção de um espaço de características contendo as mais discriminantes influencia directamente a eficácia da tarefa de classificação. Diferentes abordagens podem ser utilizadas no reconhecimento das características, nomeadamente métodos estatísticos, métodos estruturais, redes neuronais, conjuntos difusos.

Se por um lado a natureza quantitativa de reconhecimento de padrões estatísticos, torna difícil a discriminação entre os grupos com base na sua morfologia (shapebased ou estrutural), como no presente caso, já os pontos fortes das redes neuronais não suplantam a complexidade e necessidade de capacidade de cálculo paralelo inerentes. Da mesma ordem de grandeza são os constrangimentos relativos à utilização de abordagens baseadas em conjuntos difusos com recurso a um PLC. Os métodos Estruturais parecem por isso ser os mais apropriados, sendo por isso os utilizados ao longo deste trabalho.

A forma que se pertence identificar determina, para cada caso específico, qual a técnica a utilizar e que tipo de algoritmos desenvolver. Por vezes diferentes técnicas podem ser utilizadas com resultados semelhantes, porém casos existem em que a diferença de resultados é significativa e a complexidade e tamanho do algoritmo não o justificam. Neste contexto iremos desenvolver algoritmos baseados em regiões e gramáticas, representados por strings, diagramas de estado, grafos e árvores.

3.4. Síntese

O reconhecimento de padrões é um assunto relativamente recente, contudo, devido ao seu grande interesse, dotar as máquinas de comportamentos inteligentes, capazes de percepcionarem e agirem sobre o meio envolvente, um grande esforço tem vindo a ser realizado no sentido de desenvolver novos métodos ou abordagens mais capazes.

Apesar de não haver uma classificação bem definida, podemos dizer que as metodologias mais importantes serão a estatística, a estrutural, a neuronal e os conjuntos difusos. Neste trabalho iremos utilizar a abordagem estrutural dada a natureza morfológica do objecto de reconhecimento (relações entre características espaciais de um determinado padrão).

4. Simulação do processo

Existem inúmeras definições para simulação. De entre elas é pertinente citar-se a de Pegden, (1990) segundo o qual "a simulação é um processo de projectar um modelo computacional de um sistema real e conduzir experiências com este modelo, com o propósito de entender o seu comportamento e/ou avaliar estratégias para a sua operação". É neste contexto que surge o desenvolvimento da plataforma de simulação, com o intuito de recriar o sistema real em condições ideais de funcionamento. O controlo da variável ou fenómeno ruído (que é função do acabamento superficial da peça e do processo de aquisição) permitirá analisar modelos de ruído intencionalmente introduzidos, com o objectivo de desenvolver e testar algoritmos com desempenho elevado ou óptimo para cada caso particular em estudo.

O procedimento completo necessário começa com a definição do cenário em que se deseja realizar a simulação, seguindo-se esta, de acordo com as características definidas para os elementos envolvidos na simulação e, por último, a necessária análise dos resultados obtidos.

4.1. Objectivos, cenários de interesse e suporte tecnológico

A simulação é entendida, aqui, como um processo que engloba a construção do modelo em si, mas também todo o método experimental que se segue. Isto leva a que a plataforma desenvolvida tenha por objectivos:

- Descrever o comportamento ideal do sistema;
- Construir teorias e hipóteses, considerando as observações efectuadas, através da manipulação interactiva dos padrões;
- Usar o modelo para "prever" comportamentos do sistema, isto é, os efeitos produzidos por alterações no sistema ou nos métodos e algoritmos empregados;
- Avaliar o desempenho dos métodos e algoritmos, assim desenvolvidos para cada caso particular.

Este processo permitirá, paralelamente, a formação do utilizador no domínio do uso de softwares de simulação para PLC's (SoftPLC's). Uma possível simulação do funcionamento do conjunto PLC/HMI, com base numa linguagem de programação compatível com a do sistema real, permite que os algoritmos aí criados possam ser facilmente transpostos, não havendo uma completa duplicação de trabalho mas antes uma forma mais flexível e prática para a sua execução (não se ficando fisicamente "preso" ao sistema real, mas com uma interface muito semelhante a este, com maior capacidade de calculo e maior memória).

Quanto à dimensão de ferramenta, dotada de um conjunto de funcionalidades, a sua utilidade é evidente: por um lado, porque permite aproximar-se do sistema real, simulando os diferentes processos de aquisição e por causa da introdução manual do ruído, em zonas consideradas mais críticas; por outro lado, é dotada da capacidade de manipular os padrões por forma a auxiliar o utilizador na "visualização/percepção" de características mais importantes, por comparação de regiões e/ou das primitivas da forma ou padrão em determinada posição.

O CoDeSys (Controller Development System) foi o software utilizado neste trabalho. É uma ferramenta de software direccionada para tecnologias de automação industrial, desenvolvido pela empresa 3S-software, e disponível gratuitamente no seu site (na versão demonstração). É constituído basicamente por duas partes distintas: o sistema de programação CoDeSys e o sistema de implementação em tempo real CoDeSys Control. É um dos sistemas de programação, mais importantes a nível mundial, que respeita a norma IEC 61131-3 (International Electrotechnical Commission (IEC), 2007), tendo-se inclusivamente estabelecido como standard no controlo e programação de PLC's. No contexto deste trabalho é apenas utilizado enquanto ferramenta de programação, utilizando o texto estruturado para implementar os algoritmos, apesar de permitir programar em outras linguagens: Instruction List (IL), Sequential Function Chart (SFC), Function Block Diagram (FBD), Ladder Diagram LD e Continuous Function Chart (CFC).

Como alternativa ao CoDeSys, e na mesma linha de desenvolvimento, existem outros softplc's disponíveis como, por exemplo, o Zelio Soft (Software de Programação do MicroPLC Zélio da Schneider Electric.), pese embora menos interessantes. Já numa outra linha distinta, seria ainda possível recorrer a linguagens de programação de alto nível como o Visual Basic ou o C++ que, como principal atractivo, têm a possibilidade de poderem introduzir e testar de forma automática, determinado modelo de ruído, por poderem gerar automaticamente e de forma aleatoria novos ensaios, capacidade que geralmente justifica o recurso aos processos de simulação. Contudo, neste trabalho, é dada maior importância ao tipo de ruído (em que zona este ocorre – no contorno, no interior ou exterior da peça,…) e em que medida os diferentes algoritmos desenvolvidos serão ou não suficientemente robustos para os conseguir ignorar, não

se tornando assim, tão necessário, conhecer a probabilidade desse tipo de ruído ocorrer. Contudo num caso limite, um algoritmo teoricamente "pior" pode, na prática, revelar-se o mais "eficiente" e "melhor" por conseguir reconhecer a forma no ruído que mais vezes se repete. Será necessário por isso, procurar introduzir manual e intuitivamente modelos de ruído relevantes, ainda que baseados na sensibilidade e espírito crítico do operador.

A opção pelo CoDeSys deve-se principalmente à sua compatibilidade com a norma IEC-61131-3 que torna possível uma fácil transposição dos programas e algoritmos desenvolvidos em contexto de simulação para o sistema real. O facto de ter um ambiente extremamente profissional, fazendo com que muitos fabricantes (FESTO, Schneider, ABB, etc.) o utilizem, chegando mesmo a distribui-lo em versões personalizadas juntamente com os seus equipamentos, torna também a sua aprendizagem interessante, do ponto de vista da formação do próprio utilizador/estudante.

4.2. Desenvolvimento realizado

O ambiente de programação do CoDeSys é bastante intuitivo e amigável. Na figura 4.1 pode ver-se o ambiente de trabalho que permite, entre outras coisas, criar os programas, fazer o debug, acompanhar em tempo real o valor das diversas variáveis. Pode ser encontrado no anexo 3, o código de alguns dos programas desenvolvidos.

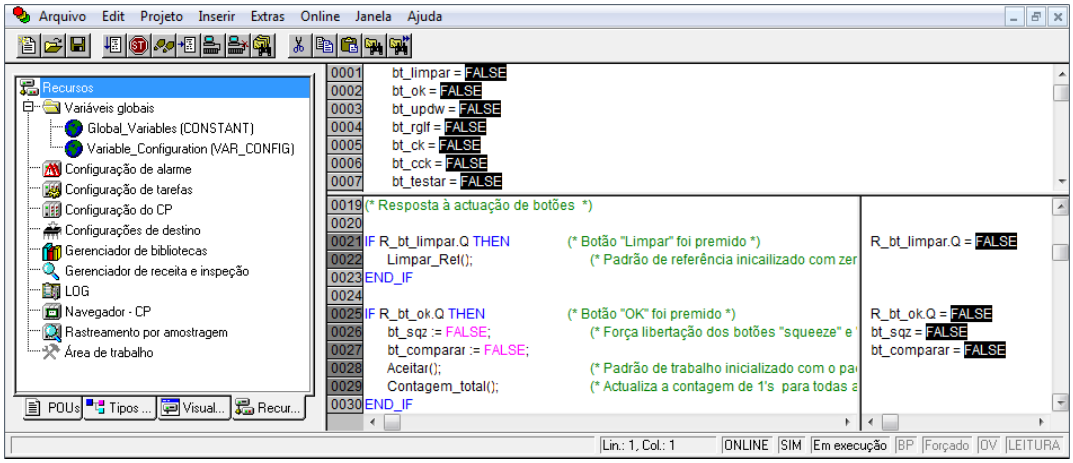


Figura 4.1. Ambiente de programação do CoDeSys

Com o objectivo de reproduzir o sistema real, foi desenvolvida uma interface gráfica simulando a HMI real, mais uma das capacidades do CoDeSys que ajudam à visualização de todo processo. Algumas destas funcionalidades estão visíveis na figura 4.2.

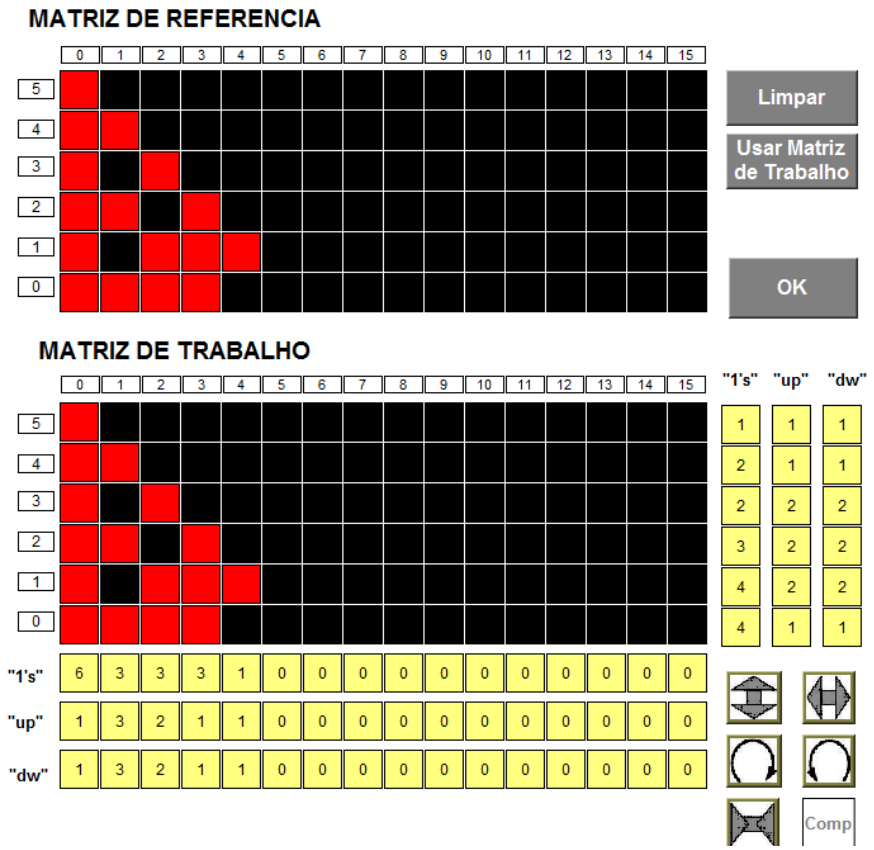


Figura 4.2. Plataforma de simulação da HMI

A "Matriz de Referência", de uma forma geral, é o local onde o utilizador pode introduzir manualmente o perfil da peça desejado, assim como o modelo de ruído pretendido (simula o espaço de amostragem). Neste contexto, existem dois sistemas de aquisição de dados que influenciam directamente a imagem da peça que é obtida, pelo que é necessário serem tidos em consideração na simulação:

- Em sitemas *"time driven"*, a aquisição de sinal é regida por um sinal de relógio. Leitura síncrona aplicável para velocidade constante de passagem da peça e frequência de amostragem *f* que permita a representação da peça dentro do espaço de amostragem.
- Em sistema em *"event driven"* ou "*data driven*" é apenas a observação de mudança de dados que conduz a aquisição. Processo de aquisição assíncrono.

A Matriz de Referência tem associado a si três funcionalidades principais:

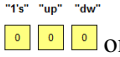
- O botão "Limpar"  – que, como o próprio nome indica, limpa (coloca a "0s" todos os valores) a Matriz de referência
- O botão "Ok"  - faz a submissão da forma desejada, transferindo-a para o espaço de trabalho, mas activando também um conjunto de funções a si associadas tais

como a determinação das dimensões da forma ou do número de "1s" ou transições ascendentes e descendentes em determinada posição

- O botão "Usar matriz de trabalho" 

– que permite transferir um padrão manipulado na matriz de trabalho para a matriz de referência.

Por seu lado, a "Matriz de trabalho" é o espaço onde vai ser processada a imagem. Aqui estão disponíveis um conjunto de imagens distintas:

- Existe um campo 

onde é possível visualizar o número de transições, bem como o número de "1s" ao longo de cada linha e coluna do espaço de amostragem
- Um conjunto de botões que permite espelhar e rodar a imagem: os botões de Flip horizontal  e Flip vertical  fazem com que a forma da peça seja invertida, tendo por base a mediatriz correspondente, os botões de rotação permitem rodar a forma no sentido horário  e no sentido contrário ao dos ponteiros do relógio  .
- O botão "Comparar" 

permite comparar formas: a presente na matriz de referência, com a da matriz de trabalho, evidenciando as regiões de diferenciação
- O botão "Squeeze"  aproxima a simulação da aquisição de sinal em data-driven, através da compressão da imagem sempre que é detectada uma coluna (amostragem) repetida
- O Botão testar 

submete a forma presente na "Matriz de trabalho" ao conjunto de algoritmos de reconhecimento desenvolvidos, activando uma "tabela de resultados".

Figura 4.3. Tabela de resultados relativa à identificação da forma representada na Figura 4.2.

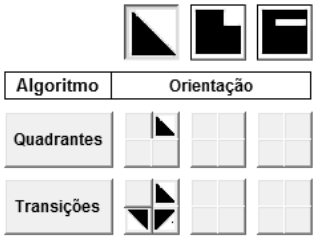
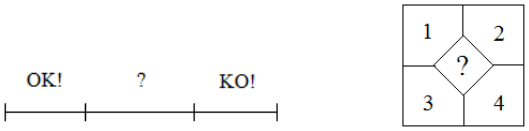


Figura 4.4. Tabela de resultados relativa à orientação da forma representada na Figura 4.2.

A tabela de resultados é o espaço que torna possível visualizar e comparar todos os algoritmos relativos a um mesmo caso. Funcionam do seguinte modo: depois de introduzida a forma desejada na matriz de referência, e eventualmente manipuladas na matriz de trabalho, é possível seleccionar, na tabela de resultados, o caso concreto que se pretende testar, premindo-se o botão com a forma correspondente. Quando actuado o botão testar, irão aparecer activos os resultados dos algoritmos existentes para a forma especificada.

4.3. Cenários simulados

Serão abordados dois problemas distintos, de identificação da correcta orientação de determinada forma, nomeadamente: 1) com dois resultados possíveis (mais estado de indeterminação), 2) com mais de dois resultados possíveis – ver figura 4.5.



a) b)

Figura 4.5. Critérios de identificação da correcta orientação de uma forma com: a) dois resultados possíveis, b) mais de dois resultados possíveis

Para os problemas de tipo 1), foram definidos três critérios de diferenciação: o critério "OK!" determina que a imagem obtida é aproximadamente igual ao elemento que se pretende verificar; o critério "KO!" determina que a imagem obtida é aproximadamente igual ao elemento de rejeição; por último, o critério "?" determina que a imagem obtida fica fora dos intervalos de confiança estabelecido nos critérios anteriores, para o caso concreto.

Já nos problemas do tipo 2) são considerados os quatro critérios 1,2,3 e 4 que determinam a orientação da forma segundo o sentido da posição que o número ocupa no espaço – ver figura 4.5. b). O critério "?", tal como nos casos do tipo 1, determina que a imagem obtida fica fora dos intervalos de confiança estabelecido para os critérios 1, 2, 3 e 4.

Tabela 4.1. Enumeração dos casos simulados para a identificação da correcta orientação de determinada forma segundo os critérios "OK!" e "KO!" definidos

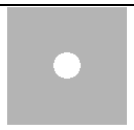
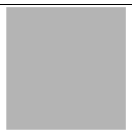
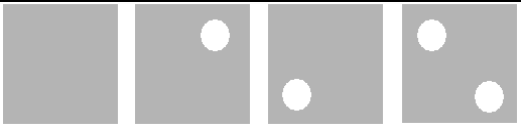
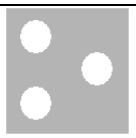
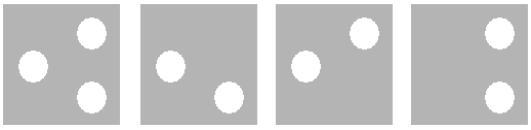
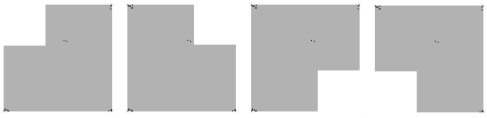
	Critério OK!	**Critério KO!**
Caso 1		
Caso 2		
Caso 3		
Caso 4		
Caso 5		
Caso 6		
Caso 7		

Tabela 4.2. Enumeração dos casos simulados de orientação de determinada marca

	1)	**2)**	**3)**	**4)**
Caso 8				
Caso 9				
Caso 10				

Com o propósito de testar e comparar o desempenho dos algoritmos em desenvolvimento, procurando prever a sua eficácia e robustez, serão definidos diferentes modelos de ruído, para o efeito. Admitir-se-á ainda que o ruído resultante do acabamento superficial seja constante, ao longo da peça, ou seja: a probabilidade de existir ruído é igual para qualquer ponto da peça. O mesmo é dizer que se considera existir uma rugosidade, capacidade reflectiva e ortogonalidade constantes ao longo da peça (imaginando uma superfície espelhada perfeita).

Dos casos definidos nas Tabelas 4.1. e 4.2., serão seguidamente expostos para os casos 1, 2 e 4 os modelos de ruído considerados, os diferentes algoritmos desenvolvidos, baseados em diferentes metodologias e técnicas e os resultados por estes obtidos. Os restantes casos 3, 5, 6, 7, 8, 9 e 10 serão do mesmo modo tratados no anexo 2.

4.3.1. Caso 1 – Identificação da correcta orientação de uma peça triangular

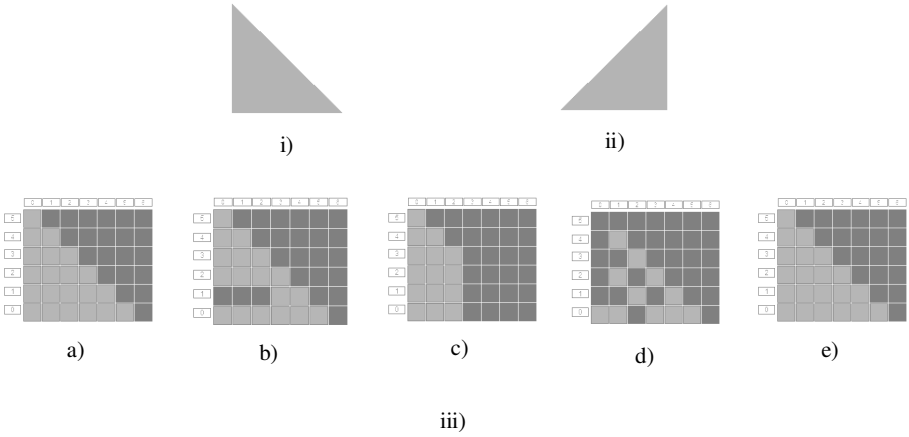


Figura 4.6. Identificação da correcta orientação de uma peça triangular: i) forma pretendida, ii) forma de rejeição, iii) modelos de ruído testados.

Na figura 4.6. pode observar-se a forma, cuja orientação se pretende identificar, assim como os modelos de ruído que serão considerados para o desenvolvimento e teste dos algoritmos. Assim, idealmente teríamos uma imagem com a matriz da forma pretendida, obtida por time driven, representada no caso a), mas considerando o caso de uma avaria temporária de um dos sensores podemos ter algo semelhante ao caso exposto em b), por seu lado, o caso c) procura simular a falha temporária e de curta duração do conjunto de todos os sensores. Já no caso d) procura-se testar a sensibilidade e robustez dos algoritmos para o aparecimento de ruído em zonas mais sensíveis da peça, nomeadamente nas zona de contorno. Por último procura-se reproduzir, no caso e, a resposta esperada da aquisição em *event driven*.

Método 1: Análise dos quadrantes

Este Algoritmo pretende analisar a imagem binária, a partir da partição ortogonal em quatro regiões iguais, que designaremos de quadrantes. As regiões comparadas são: o quadrante inferior esquerdo (*ie*) com o quadrante superior direito (*sd*) e o quadrante inferior direito (*id*) com o quadrante superior esquerdo (*se*).

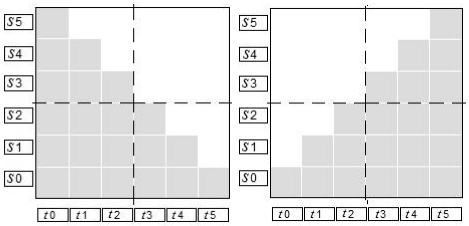


Figura 4.7. Identificação por regiões (quadrantes)

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" **-** é o triângulo pretendido, se existir um número consideravelmente maior de "1s" no quadrante *ie* do que no *sd* e, ao mesmo tempo, o numero de "1s" nos quadrantes *se* e *id* for aproximadamente igual, tal que:

$$\frac{ie+1}{sd+1} \geq 3 \quad \wedge \quad \frac{id+1}{se+1} \in [0.75,1.5] \tag{3}$$

- Critério "KO!" – é o triângulo de rejeição, se existir um número consideravelmente maior de uns no quadrante *id* do que no *se* e, ao mesmo tempo, o número de uns nos quadrantes *sd* e *ie* for aproximadamente igual, tal que:

$$\frac{id+1}{se+1} \geq 3 \quad \wedge \quad \frac{ie+1}{sd+1} \in [0.75,1.5] \tag{4}$$

- Critério "?" – quando não é possível assegurar qual o triângulo em presença. O valor de ruído admitido é superado, o que coloca a peça numa zona de incerteza. Para o primeiro e segundo casos, respectivamente:

$$\frac{ie+1}{sd+1} < 3 \quad \vee \quad \frac{id+1}{se+1} < 3 \tag{5}$$

Neste algoritmo, é possível a aquisição de dados: em Time driven - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça, dentro do espaço de amostragem, e em Data driven – aplicável dada a forma variável da peça que, em conjunto com o movimento de passagem da peça, perante os sensores de varrimento, provoca a detecção da variação e consequente aquisição para o espaço de amostragem.

Por fim, pode-se considerar que os principais pontos fortes deste algoritmo são: ter, por um lado, um critério de dupla verificação e, por outro lado, ser pouco extenso (rápido). Já o seu ponto fraco, é poder ser muito prejudicado, se ocorrer uma "descentragem" da peça;

Método 2: Análise por comparação de regiões sobrepostas

Analisa a imagem binária a partir da sobreposição dos dois objectos que se pretendem identificar e, utilizando um OU Exclusivo, extrair dessa forma as regiões que nessa comparação os mais descriminam (zona especifica e exclusiva de cada objecto), criando um "espaço de características":

- A característica do triângulo pretendido é definida pelo somatório de uns dessa região mais à esquerda do espaço de amostragem (*re*).
- A característica do triângulo de rejeição é definida pelo somatório de uns dessa região mais à direita do espaço de amostragem (*rd*).

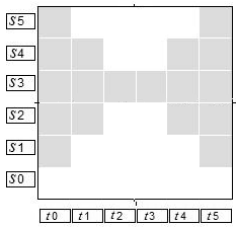


Figura 4.8. Regiões comparadas – resultado do OU exclusivo das imagens a) e b) da figura 4.6.

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – é o triângulo pretendido se existir um número consideravelmente maior de "1s" na região *re* que na *rd*. Considerando-se um modelo de distribuição constante do ruído e definindo-se um ruído máximo de mais ou menos dois pontos (~25%) em cada região, vem que:

$$\frac{re+1}{rd+1} > 2{,}5 \qquad (6)$$

- Critério "KO!" – **é** o triângulo de rejeição se existir um número consideravelmente maior de uns na região *rd* que na *re*, Nas mesmas condições consideradas no critério anterior, tal que:

$$\frac{rd+1}{re+1} > 2{,}5 \qquad (7)$$

- Critério "?" – quando o valor de ruído admitido é superado, colocando a peça numa zona de incerteza.

$$\frac{re+1}{rd+1} \leq 2{,}5 \quad \vee \quad \frac{rd+1}{re+1} \leq 2{,}5 \tag{8}$$

As duas formas de aquisição de dados são aplicáveis: em Time driven - com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem e, em Data driven – dada a forma variável da peça que em conjunto com o movimento de passagem da peça perante os sensores de varrimento, provoca a detecção da variação e consequente aquisição para o espaço de amostragem.

Este algoritmo tem por pontos fortes: ser um programa curto (mais rápido) e o facto de ser insensível ao ruído fora das zonas comparadas. O seu ponto fraco é não conseguir identificar a forma triângulo em si, mas apenas a diferença entre duas regiões escolhidas.

Método 3: Análise por String Matching

De uma forma geral este algoritmo identifica a forma da peça por comparação da sequência de leituras adquiridas pelo varrimento de sensores (colocada na matriz/espaço de amostragem), com a imagem binária pretendida e previamente armazenada em memória. Começa por procurar a palavra inicial t_0' no instante $t = t_0$, e sempre que tal acontece, verifica se a sequência pretendida é conseguida. Se sim, então a forma foi identificada, se não repete o processo para $t = t+1$.

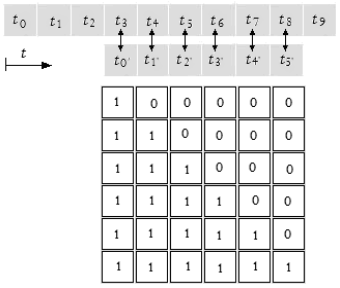


Figura 4.9. String Matching

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – A sequência de palavras que descrevem a forma desejada é encontrada no espaço de amostragem e dentro da tolerância definida (uma letra por palavra):
  Matriz_d = [63, 31, 15, 7, 3, 1] ou [111111.011111.001111.000111.000011.000001]

- Critério "KO!" – a sequência de palavras que descrevem forma de rejeição é encontrada no espaço de amostragem e dentro da tolerância definida (uma letra por palavra)
  Matriz_e[1, 3, 7, 15, 31,63] ou [000001.000011.000111.001111.011111.111111]

- Critério "?" – Nenhuma das sequências é encontrada, ou é "encontrada" mas dentro de uma tolerância superior.

É passível de ser aplicado em Time driven, com um tempo de amostragem que permita captar a imagem real, uma vez que a peça é previamente "desenha" em memória, admitindo a imagem real esperada. Mas também em Data driven dada a sua forma continuamente variável.

Tem por pontos forte ser um programa pequeno (por isso rápido) e como ponto fraco a possível ocorrência de erros na introdução dos dados iniciais.

Método 4: Ordem da sequência de transições

Este algoritmo procura verificar a presença do triângulo em função da ordem por que ocorrem as transições positivas ou negativas.

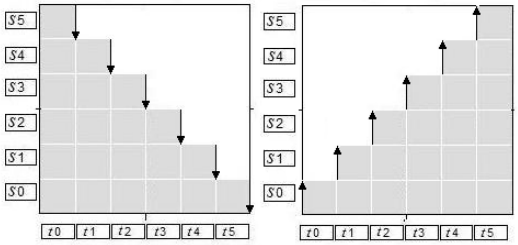


Figura 4.10. Ordem de sequência

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" - As transições descendentes ocorrem de s5 para s0

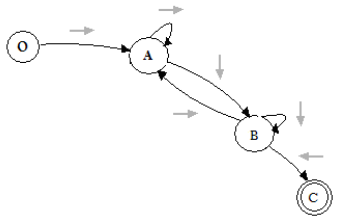


Figura 4.11. Diagrama de Estados

- Critério "KO!" - As transições ascendentes ocorrem de s0 para s5

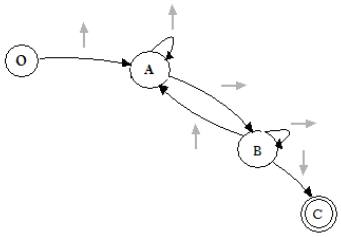


Figura 4.12. diagrama de Estados

- Critério "?" - Verificação de alternância de ascendentes e descendentes

É possível a aquisição de dados em Time driven - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem e em Data driven – aplicável dada a forma variável da peça que em conjunto com o movimento de passagem da peça perante os sensores de varrimento, provoca a detecção da variação e consequente aquisição para o espaço de amostragem.

Tem como ponto forte ser um programa pequeno (mais rápido), insensível ao ruído na zona central e contornos que não o inclinado. O seu principal ponto fraco é a sensibilidade ao ruído nas transições (zona de contorno característica.

Método 5: Contorno integral

Este algoritmo consiste em seguir, procurando fechar, a linha de contorno do triângulo, recorrendo para isso a certas regras gramaticais que articulam um conjunto de primitivas (linhas orientadas - constituído pelas transições positivas e negativas dos sensores) por forma a obter um triângulo.

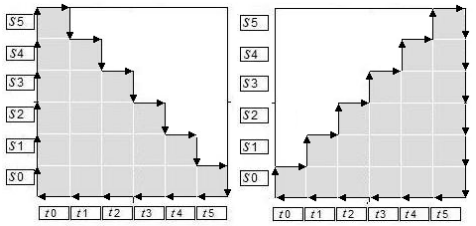


Figura 4.13. Contorno integral

Os critérios considerados são:

- Critério "OK!" - consegue concluir o ciclo fechando o contorno, descendo três ou mais degraus.

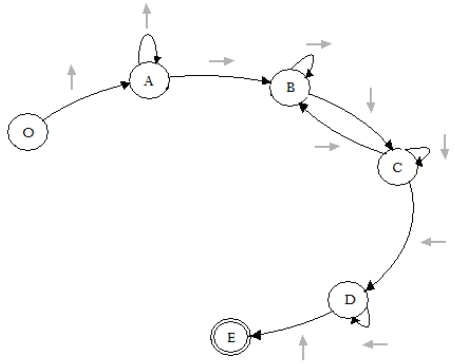


Figura 4.14. Diagrama de estados – contorno da forma pretendida

- Critério "KO!" - consegue concluir o ciclo fechando o contorno, subindo três ou mais degraus.

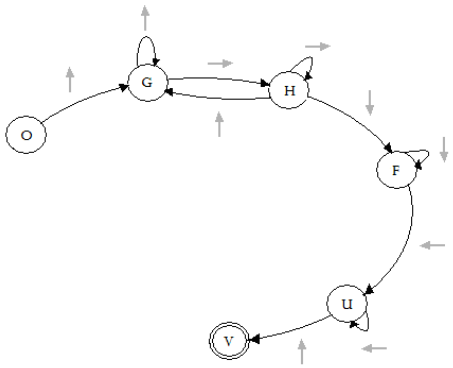


Figura 4.15. Diagrama de Estados

- Critério "?" - encontra uma quebra no contorno e não consegue fechar o ciclo ou consegue concluir o ciclo fechando o contorno, descendo ou subindo menos de três degraus.

É possível a aquisição de dados em *Time driven* - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem e em Data driven – aplicável dada a forma variável da peça que em conjunto com o movimento de passagem da peça perante os sensores de varrimento, provoca a detecção da variação e consequente aquisição para o espaço de amostragem.

Como ponto forte este algoritmo tem a sua insensíbilidade ao ruído na zona central e o facto de permitir identificar triângulos independentemente do seu comprimento (se este não ultrapassar o espaço de amostragem). Os pontos fracos são ser um programa relativamente extenso (mais lento de correr), sensível a ruído nas zonas de contorno, sendo particularmente sensível ao contorno ascendente.

Método 6: Análise por regressão

Podemos ver os dados do espaço de amostragem como um conjunto de amostras bivaridas, constituída pelo par ordenado de dados quantitativos, sendo o primeiro atributo o número de "1s" na linha i e o segundo atributo o número de "1s" da coluna j, e assim procurar a relação entre eles.

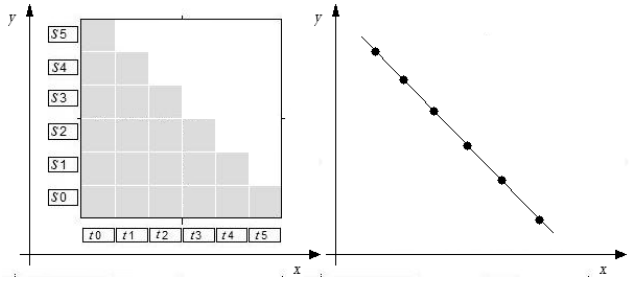


Figura 4.16. Regressão

Considerando a equação da recta no plano (*x,y*), tem-se a equação: $y = a' + b \cdot x$; onde *a*' é a ordenada na origem e b o declive. Calculo do declive *b* da regressão linear entre o número de uns das linhas (*x*) e o das colunas (*y*).

$$b = \frac{\sum_{n=1}^{N} (x - \bar{x})^2}{\sum_{n=1}^{N} (x - \bar{x})(y - \bar{y})} \tag{9}$$

A linha de tendência assim obtida é uma aproximação da linha de contorno do triângulo. Alternativamente poder-se-ia obter a tendência contando o número de uns de cada coluna e dizer que vai decrescendo ou crescendo).

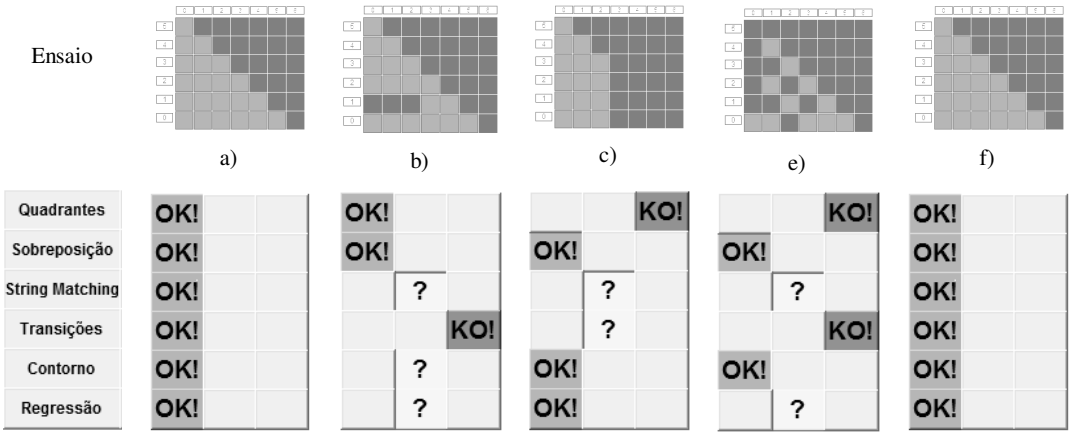
Os critérios considerados na análise do algoritmo são:

- Critério "OK!" - Na situação em que o coeficiente angular da recta é positivo, o somatório S_{xy} é também positivo uma vez que, para a maioria dos dados, a cada diferença $(x_n - \bar{x})$ corresponde uma diferença $(y_n - \bar{y})$ com o mesmo sinal, sendo portanto predominantemente positivas as parcelas do somatório. Declive no intervalo [0.75, 1.5].
- Critério "KO!" - Quando a relação linear tiver um coeficiente angular negativo, as diferenças de $(x_n - \bar{x})$ e $(y_n - \bar{y})$ serão normalmente de sinal contrário, e o somatório será negativo. Declive no intervalo [-0.75, -1.5]
- Critério "?" - Quando não existir qualquer relação (linear) entre os valores de x_n e y_n, os sinais das diferenças $(x_n - \bar{x})$ e $(y_n - \bar{y})$ serão alternadamente iguais ou opostos, tendo o somatório S_{xy} um valor próximo de zero. Declive no intervalo]-0.75, 0.75[.

As duas formas de aquisição de dados (Time driven & Data driven) são aplicáveis considerando as mesmas condições do algoritmo 1. O seu ponto forte é ser pouco sensível ao ruído independentemente da sua localização.

Resultados simulados obtidos

Tabela 4.3. Ensaios simulados - Caso 1 peça triangular



Avaliando os algoritmos pelos modelos de ruído introduzidos verificamos que neste caso, o algoritmo que melhor identifica a forma é o algoritmo "sobreposição" e aquele que pior resultados obtém o "string matching". Com resultados interessantes e passíveis de serem melhorados, surge o algoritmo quadrantes (que denota uma zona de incerteza estreita chegando a "mentir" no caso c).

4.3.2. Caso 2 – Identificação da correcta orientação de um orifício numa peça quadrangular

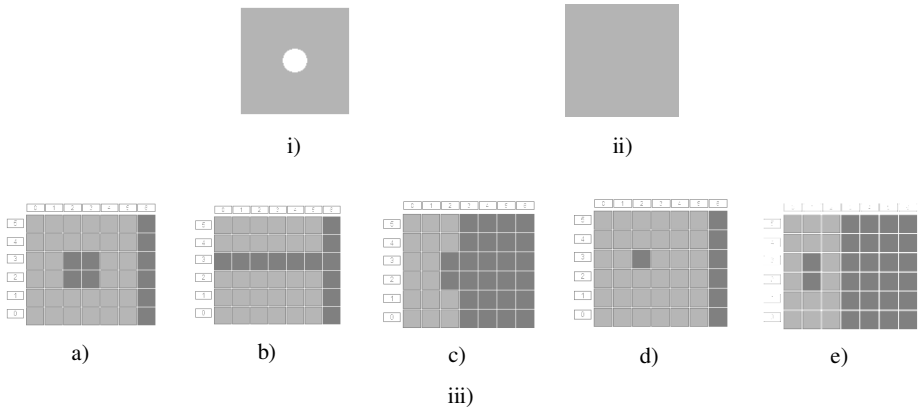


Figura 4.17. Orifício numa peça quadrangular: i) forma pretendida, i) forma de rejeição, iii) modelos de ruído testados.

Na figura 4.17. podem observar-se as formas pretendida e de rejeição em i) e ii) respectivamente. Os modelos de ruído, considerados para o desenvolvimento e teste dos algoritmos em iii), começam por representar a imagem ideal da forma pretendida no caso a), mas considerando o caso de uma avaria permanente de um dos sensores podemos ter algo semelhante ao caso exposto em b), por outro lado, no caso c) procura-se simular a falha temporária e de curta duração do conjunto de todos os sensores. Já no caso d) procura-se testar a sensibilidade e robustez dos algoritmos para o aparecimento de ruído na zona mais sensíveis da peça, mais concretamente a região onde se pode encontrar o orifício da peça. Por último procura-se reproduzir, no caso e), a resposta esperada da aquisição em *event driven*.

Método 1: Análise por sobreposição (regiões comparadas)

Analisa a imagem binária a partir da sobreposição dos dois objectos que se pretendem identificar e, utilizando um OU Exclusivo, extrai a região que nessa comparação os mais descriminam (zona especifica e exclusiva de cada objecto), criando um "espaço de características":

- A zona que maior peso tem na definição comparativa de ambas as imagens é a região central, caracterizada pelo número de "1s" aí existente (*rc*).

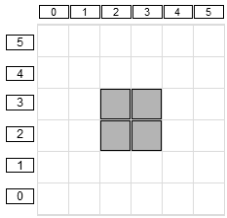


Figura 4.18. Identificação por sobreposição

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – é o quadrado pretendido se existir um número consideravelmente menos de "1s" na região central rc. Considerando-se o ruído e definindo-se um ruído máximo de + ou – um ponto (~25%) na região, vem que:

$$rc \leq 2 \tag{10}$$

- Critério "KO!" – **é** o quadrado de rejeição se existir um número consideravelmente maior de "0s" na região central rc, tal que:

$$rc > 3 \tag{11}$$

- Critério "?" – quando o valor de ruído admitido é superado, colocando a peça numa zona de incerteza.

$$rc = 3 \tag{12}$$

Apenas aplicável em Time driven, dada a dimensão esperada e consequentes valores estabelecidos nos critérios (aplicável com velocidade de tapete constante e frequência de amostragem adequada para a representação da peça dentro do espaço de amostragem).

Este algoritmo tem com pontos forte o facto de ser um programa curto (e por essa razão mais rápido) O seu ponto fraco é como apenas faz a análise na região considerada, não consegue garantir que efectivamente o corpo envolvente seja um quadrado (o que se poderia conseguir combinando um algoritmo de contorno).

Método 2: Análise por String Matching

De uma forma geral este algoritmo identifica a forma da peça por comparação da sequência de leituras adquiridas pelo varrimento de sensores (colocada na matriz/espaço de amostragem), com a imagem binária pretendida e previamente armazenada em memória. Começa por procurar a palavra inicial t_0' no instante $t = t_0$, e sempre que tal acontece, verifica se a sequência pretendida é conseguida. Se sim, então a forma foi identificada, se não repete o processo para t = t+1.

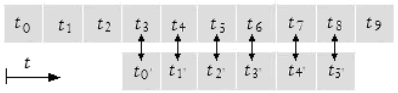


Figura 4.19. Identificação por String Matching

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – a sequência de palavras que descrevem o quadrado desejado encontrada dentro da tolerância definida (uma letra diferente por palavra).
  Matriz_d[63, 63, 51, 51, 63, 63] ou [111111.111111.110011.110011.111111.111111]

- Critério "KO!" - sequência de palavras que descrevem o quadrado de rejeição encontrada dentro da tolerância definida (uma letra diferente por palavra).
  Matriz_e[63, 63, 63, 63, 63, 63] ou [111111.111111.111111.111111.111111.111111]

- Critério "?" - Sequencia encontrada com tolerância de duas ou mais letras por palavra.

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real.

Tem por pontos forte ser um programa pequeno (por isso rápido) e como pontos fracos a possível ocorrência de erros na introdução dos dados iniciais e a sua sensibilidade a zonas de ruído concentrado.

Método 3: Análise por regressão

Podemos ver os dados do espaço de amostragem como um conjunto de amostras bivaridas, constituída pelo par ordenado de dados quantitativos, sendo o primeiro atributo o número de "1s" na linha i e o segundo atributo o número de "1s" da coluna j, e assim procurar a relação entre eles.

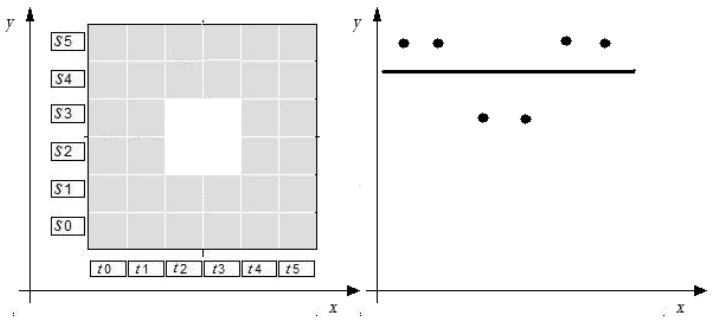


Figura 4.20. Identificação por regressão

Considerando a equação da recta no plano (*x,y*), tem-se a equação: $y = a' + b \cdot x$; onde *a*' é a ordenada na origem e b o declive. A linha de tendência assim obtida é uma aproximação da linha de contorno do triângulo. Caso a distribuição do ruído seja constante, a linha de tendência

tanto no caso da forma desejado como na forma de rejeição, será sempre horizontal, pelo que será utilizado a média como critério de diferenciação.

Os critérios considerados no algoritmo são:

- Critério "OK!" - Na situação em que a média $\bar{y}$ pertence ao intervalo [4, 5].
- Critério "KO!" - Na situação em que a média $\bar{y}$ pertence ao intervalo [5.5, 6].
- Critério "?" - Na situação em que a média $\bar{y}$ pertence aos intervalos [0, 4[ou]5, 5.5].

As duas formas de aquisição de dados (Time driven & Data driven) são aplicáveis. O seu ponto forte é ser pouco sensível ao ruído independentemente da sua localização.

Método 5: Análise por sequência de transições

De uma forma geral este algoritmo identifica a forma da peça pela presença e sequência de um conjunto de transições esperadas em determinada zona da peça. Começa por verificar a existência de uma transição positiva e uma negativa nas duas linhas do centro da peça e repete o processo para as colunas do centro da peça. O somatório do número de transições assim detectadas determinará a forma em presença.

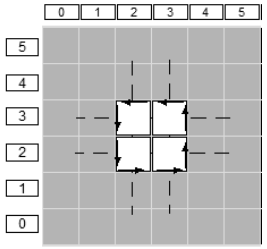


Figura 4.21. Sequência de transições

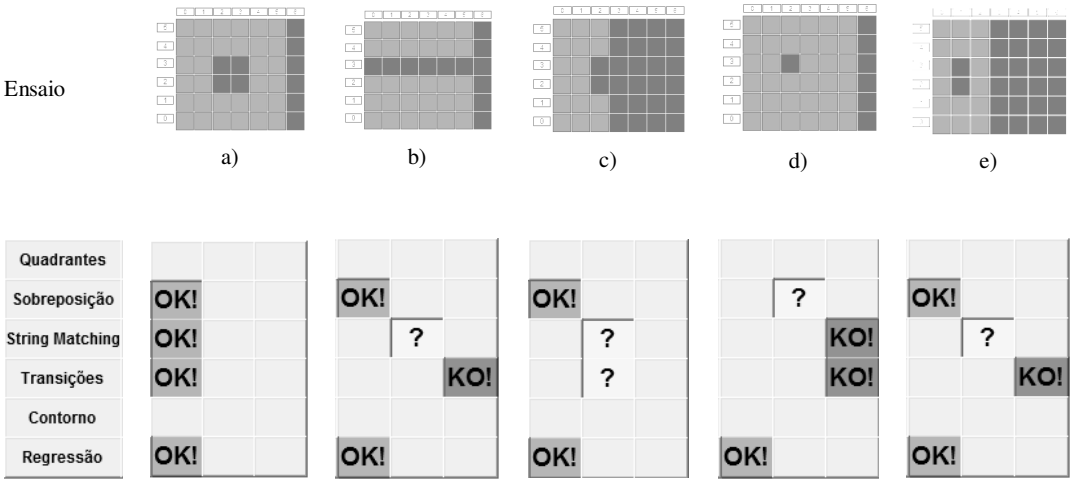
Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – Mais que 4 transições positivas ou outras tantas negativas encontradas.
- Critério "KO!" – Detectadas 2 ou menos transições negativas.
- Critério "?" – Quando detectadas 3 transições positivas.

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real. Tem por pontos forte ser um programa insensível ao ruído fora destas duas linhas e duas colunas, e como pontos fracos o facto de se a zona de furo estiver descentrada o algoritmo terá um mau desempenho e a sua sensibilidade ao ruído nas linhas e colunas analisadas.

Resultados obtidos

Tabela 4.4. Ensaios simulados - Caso 2 peça quadrangular com um orifício



Neste caso, os algoritmos quadrantes e contorno não são aplicados. Para os ensaios realizados com diferentes modelos de ruído verifica-se que o algoritmo "sobreposição" é aquele que melhor identifica a forma. O algoritmos string matching apresenta mais uma vez resultados muito desanimadores, assim como os resultados do algoritmo transições, acabam por se muito maus, uma vez que o algoritmo, erradamente indica a presença da forma de rejeição em *b*, *d* e *e*.

4.3.3. Caso 4 – Identificação da correcta orientação de um orifício numa peça triangular

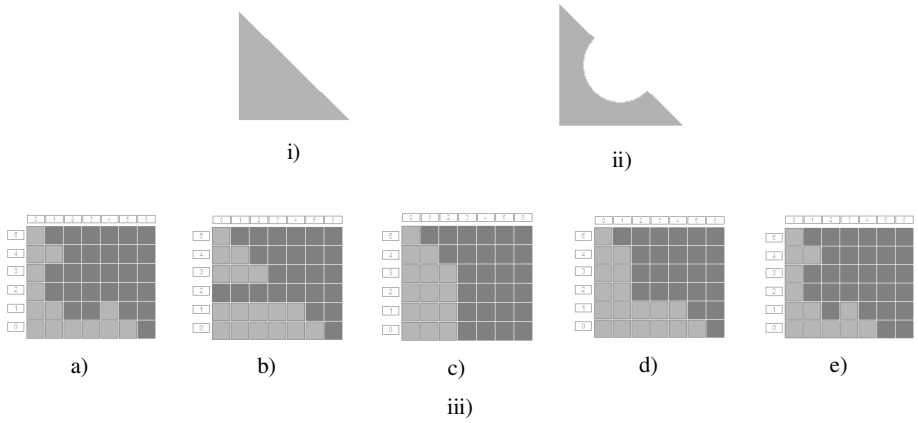


Figura 4.22. Peça triangular com um orifício: i) forma pretendida, ii) forma de rejeição iii) modelos de ruído testados.

Na figura 4.22. podem observar-se as formas pretendida e de rejeição em i) e ii) respectivamente e os modelos de ruído, considerados para o desenvolvimento e teste dos algoritmos em iii). Estes começam por representar a imagem ideal da forma de rejeição no caso a), mas considerando o caso de uma avaria permanente de um dos sensores podemos ter algo semelhante ao caso exposto em b), por outro lado, no caso c) procura-se simular a falha temporária e de curta duração do conjunto de todos os sensores. Já no caso d) procura-se testar a sensibilidade e robustez dos algoritmos para o aparecimento de ruído na zona mais sensíveis da peça, mais concretamente na segunda linha e coluna, região de contorno do orifício da peça. Por último procura-se reproduzir, no caso e), a resposta esperada da aquisição em *event driven*.

Método 1: Análise por regiões comparadas

O algoritmo analisa a imagem binária a partir da sobreposição dos dois objectos que se pretendem identificar e, utilizando um OU Exclusivo, extrai a região que nessa comparação os mais descriminam (zona especifica e exclusiva de cada objecto), criando um "espaço de características": a zona que maior peso tem na definição comparativa de ambas as imagens é a região central, caracterizada pelo número de "1s" aí existentes (*rc*).

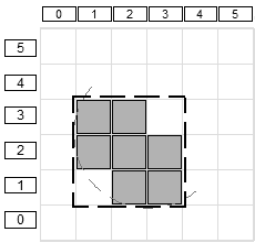


Figura 4.23. Identificação por regiões comparadas

Os critérios tomados em conta são os seguintes:

- Critério "OK!" – é o triângulo pretendido se existir um número consideravelmente menos de "1s" na região do semi circulo rc. Considerando-se o ruído e definindo-se um ruído máximo de + ou – dois pontos (~25%) na região, vem que:

$$rc \geq 5 \tag{13}$$

- Critério "KO!" – é o quadrado de rejeição se existir um número consideravelmente maior de "0s" na região central rc, tal que:

$$rc \leq 2 \tag{14}$$

- Critério "?" – quando o valor de ruído admitido é superado, colocando a peça numa zona de incerteza.

$$rc \in [3,4] \tag{15}$$

Aplicável em: Time driven - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem, E em Data driven – aplicável dada a forma variável da peça que em conjunto com o movimento de passagem da peça perante os sensores de varrimento, provoca a detecção da variação e consequente aquisição para o espaço de amostragem.

Este algoritmo tem com pontos forte o facto de ser um programa curto (e por essa razão mais rápido) O seu ponto fraco é como apenas faz a análise na região considerada, não consegue garantir que efectivamente o corpo envolvente seja um triangulo (o que se poderia conseguir combinando um algoritmo de contorno).

Método 2: Análise por regiões sextantes

Este Algoritmo pretende analisar a imagem binária a partir da partição ortogonal em nove regiões iguais, que designaremos de sextantes: *se* – superior esquerdo, *me* – médio esquerdo, *ie* – inferior esquerdo, *ms* – médio superios, *mm* – médio médio, *mi* – médio inferior, *sd* – superior direito, *md* – médio direito, *id* – inferior direito. As regiões comparadas são: o sextante *mm* é o que mais informações contem sobre as duas formas.

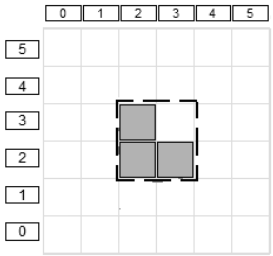


Figura 4.24. Identificação por regiões (sextantes)

O algoritmo apresenta os seguintes critérios:

- Critério "OK!" – é o quadrado pretendido se existir um número consideravelmente menos de "1s" na região central *mm*. Considerando-se o ruído e definindo-se um ruído máximo de + ou – um ponto (~25%) na região, vem que:

$$mm \geq 2 \tag{16}$$

- Critério "KO!" – é o quadrado de rejeição se existir um número consideravelmente maior de "0s" na região central rc, tal que:

$$mm \leq 1 \tag{17}$$

- Critério "?" – quando o valor de ruído admitido é superado, colocando a peça numa zona de incerteza.

$$rc = 2 \tag{18}$$

Tipo de aquisição de dados aplicável: Time driven - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem. Data driven – aplicável dada a forma variável da peça que em conjunto com o movimento de passagem da peça perante os sensores de varrimento, provoca a detecção da variação e consequente aquisição para o espaço de amostragem.

Os principais pontos fortes deste algoritmo é ser insensível ao ruído for a da região central. Já o seu ponto fraco é poder ser muito prejudicado se ocorrer uma "descentragem" da peça e por outro lado se basear numa pequena zona o que o torna sensível a ruídos concentrados.

Método 3: Análise por sequência de transições

De uma forma geral este algoritmo identifica a forma da peça pela presença e sequência de um conjunto de transições esperadas em determinada zona da peça. As zonas determinantes são mais facilmente visualizadas se se recorrer à função comparar, desta forma pode dizer-se que integramos os algoritmos transições com o sobreposição. Este algoritmo verifica o momento em que se dão as transições positivas na linha 1 e coluna 4, assim como as transições negativas na linha 4 e coluna 2.

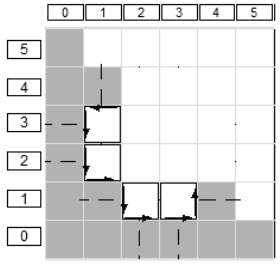


Figura 4.25. Identificação por sequência de transições

Os critérios tomados em conta são os seguintes:

- Critério "OK!" – detectadas transições negativas nas posições 4 e 3 nas linhas 2 e 3 e nas colunas 2 e 3 respectivamente.
- Critério "KO!" – detectadas transições negativas na posição 1 nas linhas 2 e 3 e nas coluna s 2 e 3.
- Critério "?" – detecta mais ou menos transições que as estabelecidas nos critérios anteriores e por ordens diferentes.

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real. Tem por pontos forte ser um programa insensível ao ruído fora destas duas linhas e duas colunas, e com ponto fraco a sua sensibilidade ao ruído nas linhas e colunas analisadas.

Método 3: Análise por String Matching

De uma forma geral este algoritmo identifica a forma da peça por comparação da sequência de leituras adquiridas pelo varrimento de sensores (colocada na matriz/espaço de amostragem), com a imagem binária pretendida e previamente armazenada em memória. Começa por procurar a palavra inicial t_0' no instante $t = t_0$, e sempre que tal acontece, verifica se a sequência pretendida é conseguida. Se sim, então a forma foi identificada, se não repete o processo para $t = t+1$.

Figura 4.26. Identificação por string matching

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – a sequência de palavras que descreve a forma desejada é encontrada dentro da tolerância definida (ruído equivalente a uma letra diferente por palavra).
  Matriz_d[63, 31, 15, 7, 3, 1] ou [111111.011111.001111.000111.000011.000001]

- Critério "KO!" - sequência da forma de rejeição é encontrada dentro da tolerância definida (ruído equivalente a uma letra diferente por palavra).
  Matriz_e[63, 19, 1, 1, 3, 1] ou [111111.010001.000001.000001.000011.000001]

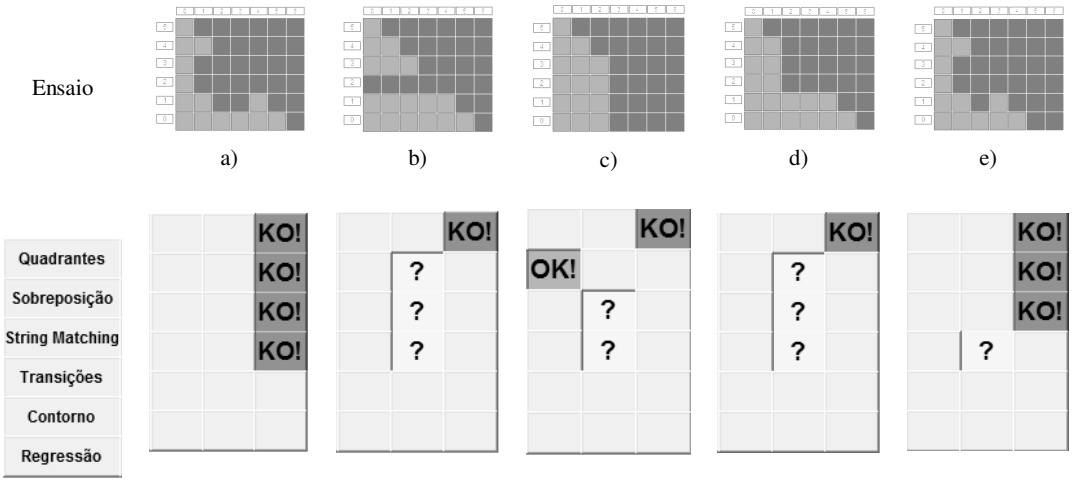
- Critério "?" - Sequencia encontrada com tolerância de duas ou mais letras por palavra.

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real (será possível em data driven se a imagem introduzida para comparação, for introduzida já com esse fim em vista e tenha por isso, a forma esperada para uma aquisição data driven da peça).

Tem por pontos forte ser um programa pequeno (por isso rápido) e como pontos fracos a possível ocorrência de erros na introdução dos dados iniciais e a sua sensibilidade a zonas de ruído concentrado.

Resultados obtidos

Tabela 4.5. Ensaios simulados - Caso 4 peça triangular com um orifício



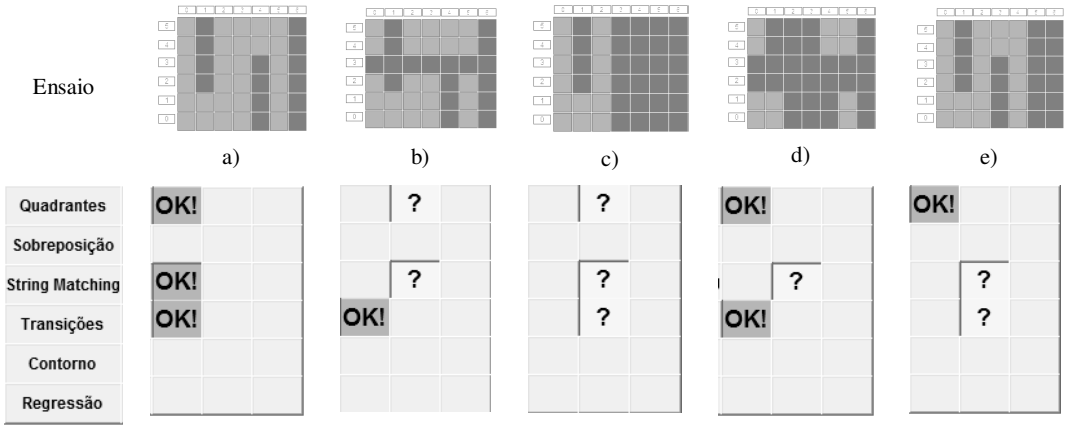
Nestes ensaios são testadas tanto a forma pretendida com a forma de rejeição, o que torna os resultados muito esclarecedores. O algoritmo que melhor resultados obtém é o quadrantes, acertando em todos os modelos de ruído testados. Há ainda a evidenciar que o algoritmo sobreposição é o único a "mentir", mais precisamente em c. Já o algoritmo transições denota uma zona de incerteza muito larga. O algoritmo string matching continua a desapontar.

4.4. Outros resultados obtidos (resumo)

Como é possível verificar alguns algoritmos são mais sensíveis a ruídos junto ao contorno da forma ao passo que outros são-no à densidade de "1s" em determinadas regiões da forma. Seguidamente serão apresentados os resultados dos diferentes algoritmos aplicados aos casos definidos nas tabelas 4.1. e 4.2. Os algoritmos correspondentes podem ser consultados no anexo 1, assim como o código dos programas está disponível no anexo 3.

4.4.1. Caso 3 – Identificação da correcta orientação das reentrâncias de uma peça quadrangular

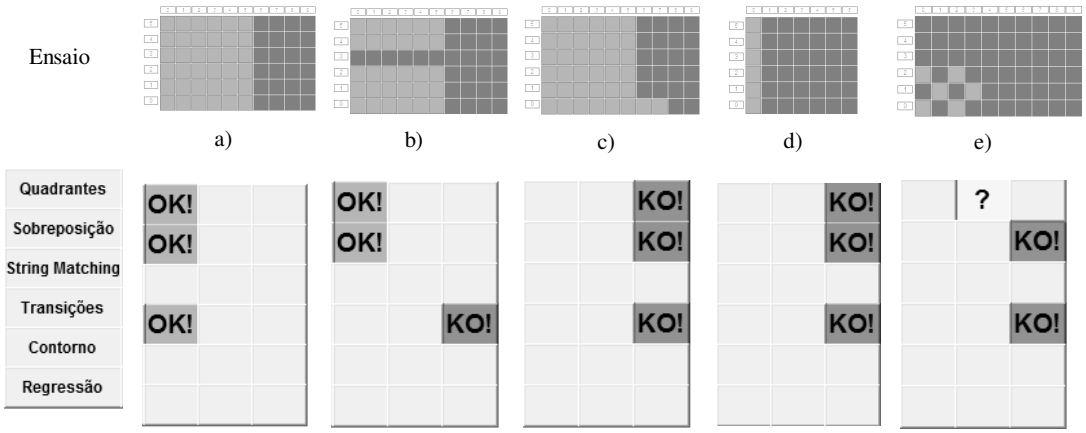
Tabela 4.6. Ensaios simulados - Caso 3 peça quadrangular com reentrâncias



Neste caso são testados apenas três algoritmos. O algoritmo dos quadrantes é aquele que apresenta os melhores resultados, sendo o único que identifica positivamente a forma em data driven. O mesmo acontece para o algoritmo transições desta feita, na presença de uma falha de um sensor. O algoritmo string matching continua a apresentar resultados desanimadores.

4.4.2. Caso 5 – Identificação da correcta orientação de uma peça rectangular

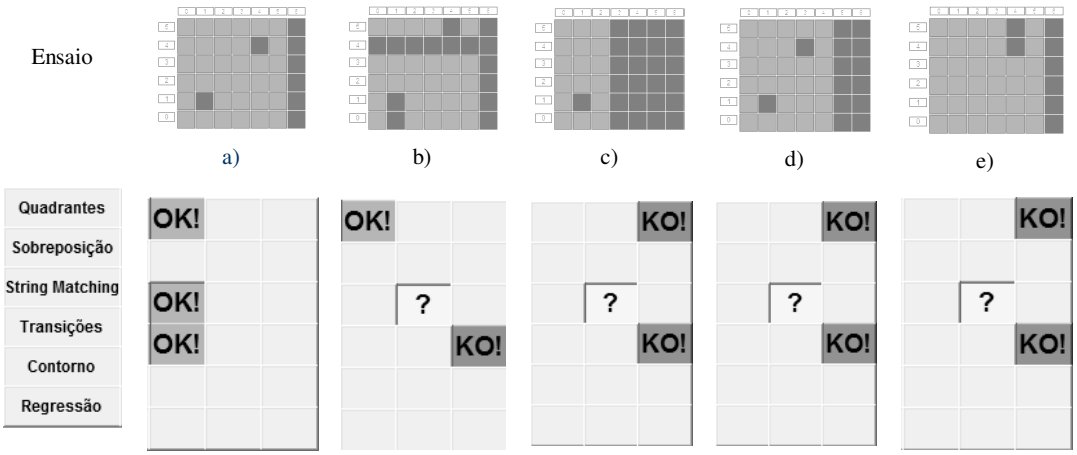
Tabela 4.7. Ensaios simulados - Caso 5 peça rectangular



Avaliando os algoritmos pelos modelos de ruído introduzidos verificamos que neste caso, o algoritmo que melhor identifica a forma é o algoritmo quadrantes. O algoritmo sobreposição parece muito sensível a ruído que possa surgir em zonas externas à peça, por seu turno o algoritmo transições demonstra uma fragilidade no facto de parecer não "suportar" a falha de um sensor.

4.4.3. Caso 6 – Identificação da correcta orientação de dois orifícios de uma peça quadrangular

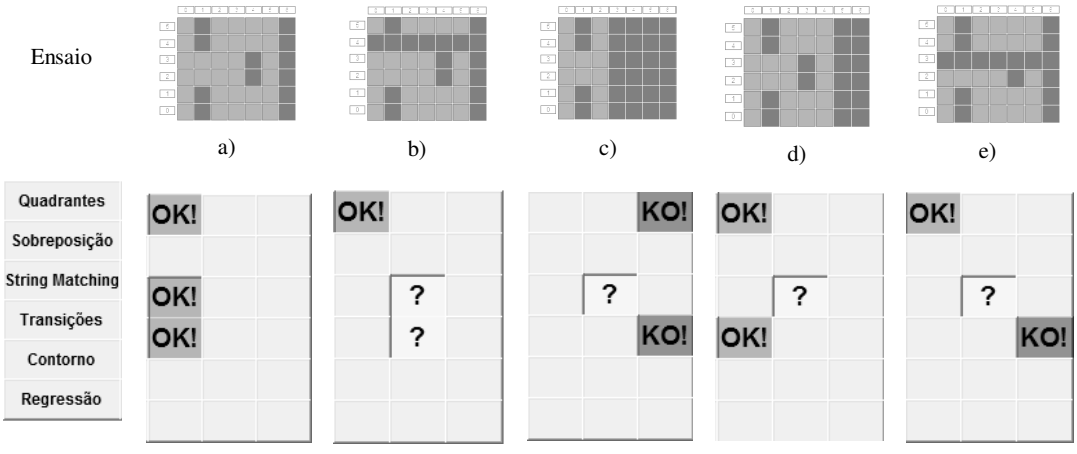
Tabela 4.8. Ensaios simulados - Caso 6 peça quadrangular com dois orifícios



Avaliando os algoritmos pelos modelos de ruído introduzidos verifica-se que o algoritmo que mais vezes identifica a forma pretendia é o "quandrantes". Nenhum dos algoritmos consegue detectar a forma correcta quando adquirida por data driven. Já o algoritmo transições "mente" quando ocorre uma falha dum sensor numa posição coincidente com um dos orifícios.

4.4.4. Caso 7 – Identificação da correcta orientação de três orifícios de uma peça quadrangular

Tabela 4.9. Ensaios simulados - Caso 7 peça quadrangular com três orifícios

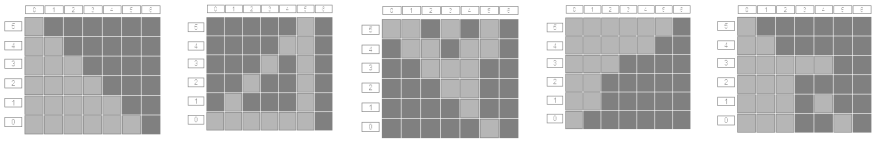


Para os ensaios realizados com diferentes modelos de ruído verifica-se que o algoritmo "quadrantes" continua a ser aquele que melhor identifica a forma. O algoritmos string matching apresenta mais uma vez resultados muito desanimadores, assim como os resultados do algoritmo transições, neste caso demonstram ser sensíveis a falha de sensores na zona central.

4.4.5. Caso 8 – Orientação possível de uma marca triangular

Tabela 4.10. Ensaios simulados - Caso 8 orientação de marca triangular

Ensaio



Quadrantes

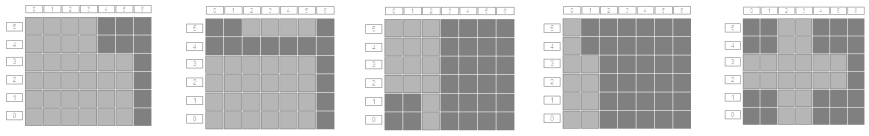
Transições

De uma forma geral, ambos os algoritmos são capazes de detectar a orientação da forma, contudo reagem de forma diferente na zona de incerteza: o primeiro aquando da falta de exclusividade não activa nenhuma resposta, o segundo activa todas – mas ambos os comportamentos denotam incerteza.

4.4.6. Caso 9 – Orientação possível de uma marca quadrangular

Tabela 4.11. Ensaios simulados - Caso 9 orientação de marca quadrangular

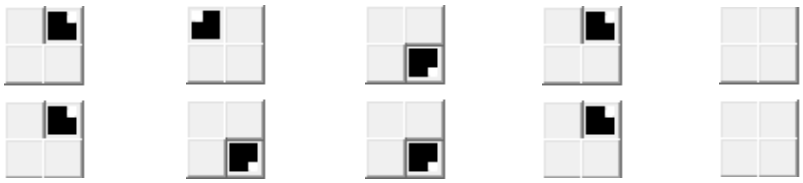
Ensaio



a) b) c) d) e)

Quadrantes

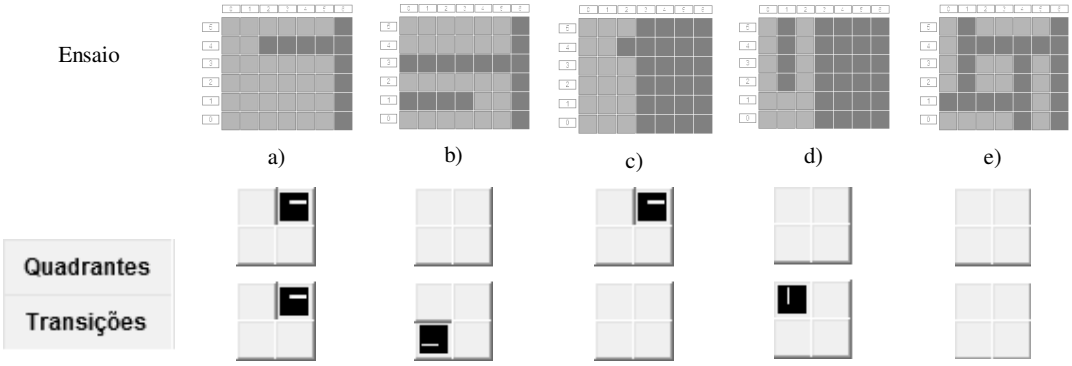
Transições



Pelos modelos de ruído introduzidos verifica-se que o algoritmo que mais vezes identifica a forma pretendia é o "quadrantes". Ambos reagem da mesma forma em caso de incerteza, não activando nenhuma resposta. Nenhum consegue identificar a correcta orientação na presença de apenas metade da peça em c.

4.4.7. Caso 10 – Orientação possível de uma marca quadrangular

Tabela 4.12. Ensaios simulados - Caso 10 orientação de marca rectangular



Pelos modelos de ruído introduzidos verifica-se que ambos os algoritmos identificam a forma pretendia o mesmo número de vezes, sendo que o "transições"consegue detectar a forma mesmo na presença de uma falha de um sensor (ver ensaio b), ou contrário do quadrantes, que por seu lado consegue identificar a forma na presença de apenas meia peça (ver ensaio c). Ambos reagem da mesma forma em caso de incerteza, não activando nenhuma resposta.

4.5. Conclusões e necessidades adicionais de investigação

No início deste capítulo, após um breve enquadramento das razões que levaram ao desenvolvimento de uma plataforma de simulação, bem como à escolha do software a utilizar, foi feita uma apresentação das diversas funcionalidades da plataforma/ferramenta assim como da interface de apresentação de resultados. A Plataforma mostrou ser uma ferramenta amigável, de grande utilidade para a manipulação e apoio à compreensão dos diferentes padrões. A sua semelhança ao software de programação do PLC também se mostrou muito vantajoso.

.

Os algoritmos desenvolvidos, tendo por base as metodologias estruturais, tiveram por objectivo: identificar formas ou orientações de determinado objecto. Para cada um dos cenários estabelecidos, critérios especificamente criados para o efeito, procuraram responder de forma o mais robusta possível aos modelos de ruído simulados, dado nada se saber do ruído real introduzido aquando o processo de aquisição de dados e devidos a diferentes acabamentos superficiais. A principal dificuldade esteve directamente relacionada com a necessidade de formalização dos métodos estruturais, necessária ao desenvolvimento dos algoritmos, por ser

algo complexo. Com mais tempo seria possível desenvolver algoritmo mais refinados e inter-relacionados, misturando as melhores características dos diferentes métodos.

Dos resultados obtidos das Tabelas 4.3 – 4.12 pode concluir-se que o algoritmo "sobreposição" é aquele que melhor desempenho apresenta no caso da identificação dos casos 1 e 2, já nos casos 3, 4, 5, 6 e 7 o que melhor identifica a forma pretendida é o algoritmo "quadrantes". Aquele que de uma forma geral piores resultados obtém é o "string matching". Pelo que, de entre todos os algoritmos analisados, suscita particular interesse o aprofundamento do estudo (experimental) destes que parecem obter os melhores resultados, nomeadamente: o "quadrantes" e o "sobreposição", baseados nas regiões, mas também, por outro lado, e baseada nas primitivas: o "transições". Será também interessante procurar-se investigar-se aquele que de longe obteve os piores resultados durante a simulação: o "string matching", que, apesar de aparentemente mau, poder vir a funcionar muito bem em determinadas condições de ruído.

Pode concluir-se assim que a simulação foi muito importante para a compreensão do padrão, não sendo contudo por si só suficiente, dado o desconhecimento das condições reais de funcionamento e as contingências ("imprevisibilidades") assim introduzidas.

5. Investigação experimental

A experimentação tem um valor fundamental na aplicação a um sistema real. A implementação e análise experimental dos algoritmos desenvolvidos permitem um feedback importante para o ajuste dos parâmetros dos mesmos. Permite ganhar a noção exacta da influência que o sistema sofre devido à aquisição de sinal e às diferentes superfícies reflectoras.

5.1. Objectivos e cenários de interesse

A Investigação experimental é o processo que engloba a implementação dos algoritmos desenvolvidos no modelo real. O objectivo principal é testar a robustez dos algoritmos e seu comportamento na presença dos modelos de ruído reais, introduzidos pelas diferentes superfícies reflectoras testadas mas também pela qualidade" do sistema de aquisição de dados. Paralelamente, a investigação experimental permite o domínio do uso do software de programação do PLC's e da HMI,

5.2. Suporte tecnológico

O sistema responsável pela aquisição de sinal, é composto por 6 sensores fotoeléctricos do tipo reflectivo (Long Distance Reflective Switch, OPB732) (Figura 2.2.) pré-montados na vertical, central e perpendicularmente ao tapete de transporte da peça, ver Lopes, Alcino e Ribeiro, Fernando, (2009):

- A alimentação dos sensores é feita em série o que permite identificar se um dos sensores avariou ou está apenas desligado. A alimentação dos sensores é comum à alimentação do PLC.

- A distância entre cada sensor é de aproximadamente 1 cm, sendo que o primeiro sensor dista aproximadamente 2,5 cm do plano de movimentação do tapete.
- Os sensores foram numerados de forma a que ao sensor mais próximo do tapete correspondesse o menor índice.
- Foi necessário proceder a uma estruturação da aplicação de modo a que a face reflectora da peça passe sempre na perpendicular e a uma distância constante da coluna de sensores.

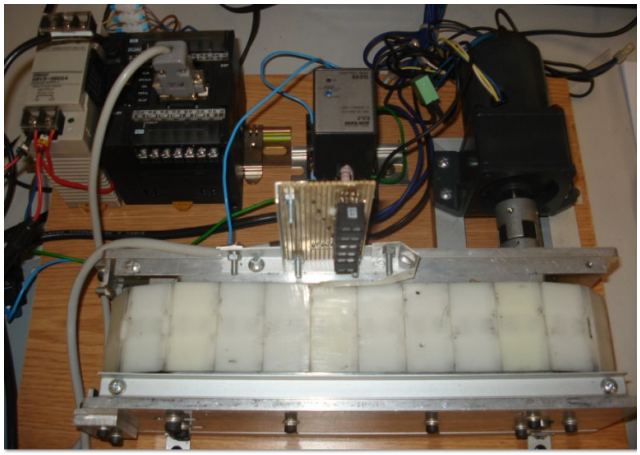


Figura 5.1. Imagem de conjunto do equipamento utilizado

O tapete é accionado por um motor AC 100 V e um variador de velocidade (Figura 1). A ligação motor-tapete é feita através de um redutor e um acoplamento elástico. A energia que alimenta o motor provem de uma fonte AC de 100V. Um selector bi-estável liga/desliga o motor, esse selector tem ainda associado um contacto normalmente aberto ligado a uma entrada do PLC. Como o accionamento do motor funciona em lógica inversa, quando o motor estiver ligado, a entrada associada no PLC estará desligada e vice-versa.

O PLC utilizado é o CP1L da OMRON (Figura 5.2. a)). Este PLC requer alimentação a 24V DC que provém de uma fonte S8VS-06024 também da OMRON. O PLC está dotado de 8 entradas e 6 saídas todas a 24V.

a) b)

Figura 5.2. Equipamentos utilizados: a) PLC CPIL, b) HMI NS5-MQ00B-V2

O PLC está ligado a uma unidade HMI NS5-MQ00B-V2 da OMRON (Figura 5.2. b)). Essa ligação é feita em série através de um cabo com ficha de 9 pinos utilizando um protocolo proprietário da OMRON. Nesta ligação a HMI assume o papel de Master e o PLC é o Slave.

Tanto o PLC com a HMI podem ser programados por um computador que comunica com estas através de portas USB.

O software utilizado é o CX-Programmer. Este Software, para o caso do PLC CPIL apenas permite desenvolver programas em ladder, contudo possui uma funcionalidade, a Funtion Block, que torna possível a escrita de "rotinas" em linguagem de texto. Esta funcionalidade permite transpor os programas desenvolvidos na plataforma de simulação recorrendo a apenas pequenas alterações como substituir as variáveis globais por variáveis internas alocadas a determinada posição de memória ou a forma de se chamarem subrotinas entre outros pormenores. Como principal limitação, tem o facto se ser um método bastante mais ineficiente pelo que rapidamente ocupa a memória do PLC, não sendo por isso possível ter todos os programas a correr em simultâneo. A alternativa é converte-los para linguagem ladder.

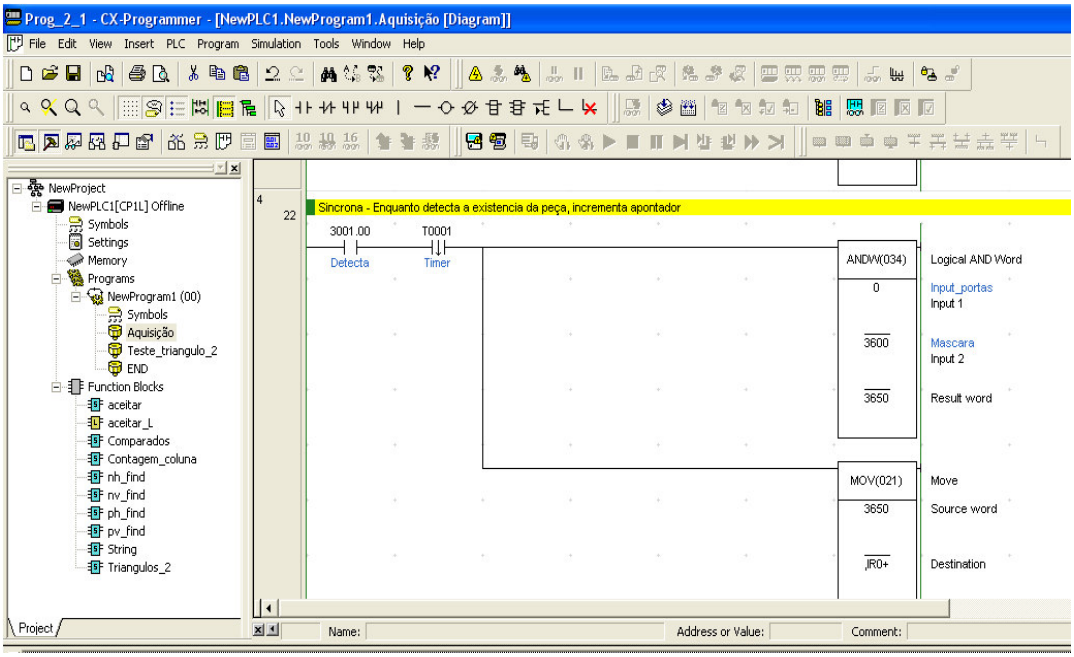


Figura 5.3. Ambiente de trabalho do programa CX-Programmer

O software de programação da HMI, o CX-Designer, também é bastante amigável – ver figura 5.4.

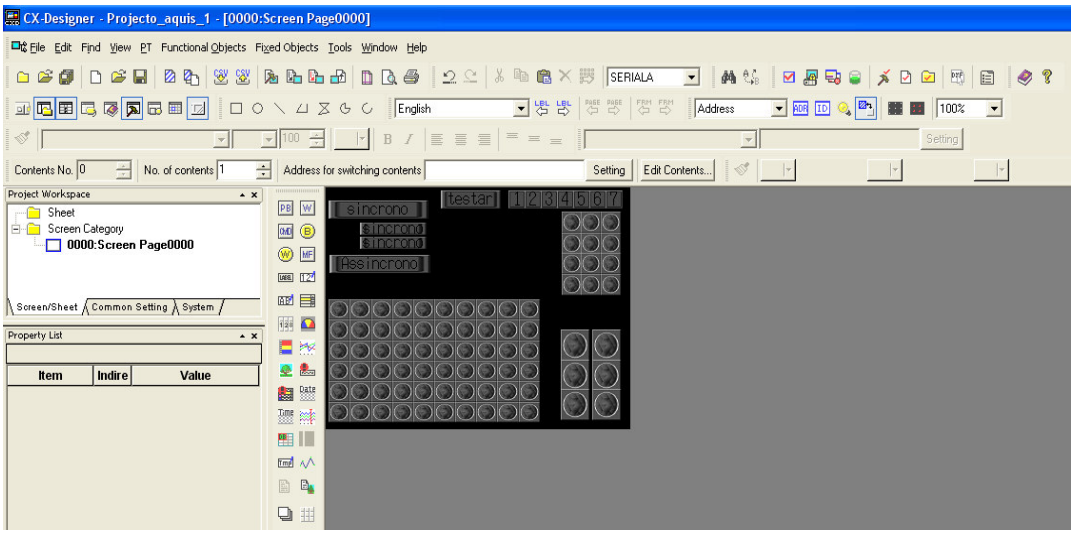


Figura 5.4. Ambiente de trabalho do programa CX-Designer

5.3. Cenários experimentados

Serão submetidos à experimentação os mesmos casos definidos para a simulação presentes na Tabela 4.1. Enumeração dos casos simulados para a identificação de formas segundo os critérios "ok" e "ko" definidos, e da Tabela 4.2. Enumeração dos casos de orientação simulados. Já os algoritmos implementados serão aqueles que no capítulo 4 se entendeu que seriam mais interessantes, nomeadamente: sobreposição, transições , string matching e quadrantes.

A aquisição de dados será efectuada em Time Driven (com a possibilidade de seleccionar duas frequências de aquisição diferentes, que seleccionadas adequadamente permitam captar o tamanho real da peça) e em Data Driven. Serão ainda testadas duas superfícies: uma de referência – superfície espelhada teoricamente perfeita, e uma segunda superfície prateada. Ambas podem ser vistas na figura 5.

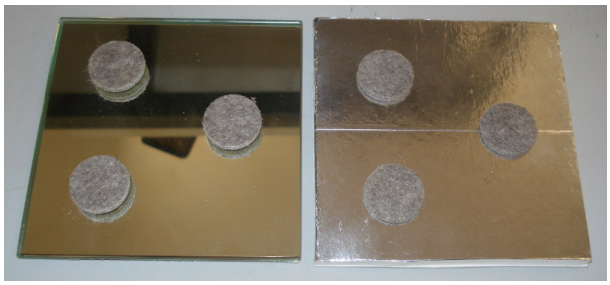


Figura 5.5. Superfícies reflectoras: a) superfície espelhada, b) superfície revestida com filme de prata

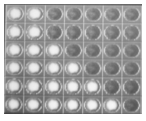
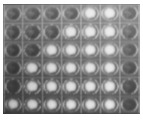
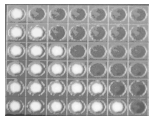
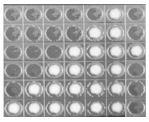
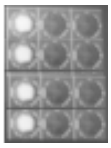
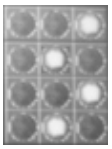
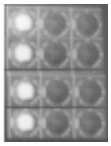
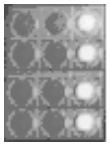
Como estrutura de suporte foram utilizados blocos de madeira pré existentes, aos quais foram acopladas as diferentes superfícies reflectoras (Figura 5.5.). Para simular os furos nas peças foram utilizados pedaços autocolantes de velcro cinzento. Este material é de fácil implementação e a sua cor não é detectada pelos sensores.

5.4. Resumo dos resultados obtidos

Nesta secção são apresentados os resultados dos ensaios realizados para os diferentes casos. Permite comparar os dados da aquisição com os resultados dos algoritmos. As duas colunas da esquerda dizem respeito a imagens obtidas por Time Driven, as duas da direita são obtidas por Data Driven. A primeira e terceira colunas dizem respeito a ensaios realizados com a superfície espelhada, as restantes.

5.4.1. Caso 1 – Identificação da correcta orientação de uma peça triangular

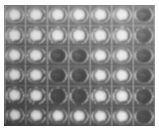
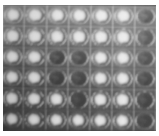
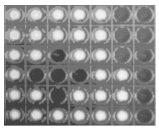
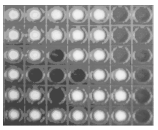
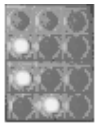
Tabela 5.1. Ensaio experimental - Caso 1 peça triangular

Ensaio				
	(70% dos casos)	(90% dos casos)	(100% dos casos)	(100% dos casos)
Quadrantes				
Sobreposição				
String Matching				
Transições				

Neste caso pode-se constatar que tanto os resultados obtidos por experimentação se aproximam dos da simulação, uma vez que também o ruído que surge na experimentação é semelhante ao modelo de ruído simulado. Apesar destes bons resultados os algoritmos que seguem a técnica quadrante e sobreposições apresentam resultados mais fiáveis principalmente na peça com a forma de rejeição.

5.4.2. Caso 2 – Identificação da correcta orientação de um orifício de uma peça quadrangular

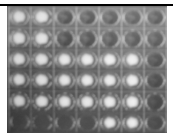
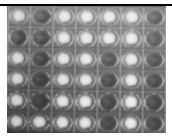
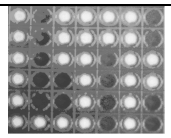
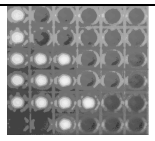
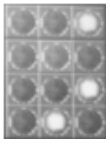
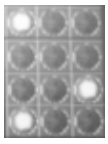
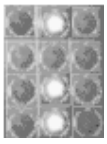
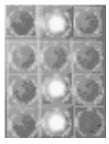
Tabela 5.2. Ensaio experimental - Caso 2 peça quadrangular com um orifício

Ensaio	 (100% dos casos)	 (70% dos casos)	 (50% dos casos)	 (60% dos casos)
Quadrantes				
Sobreposição				
String Matching				
Transições				

O caso 2 através da aquisição de dados por Time Driven os resultados obtidos foram bastante satisfatórios já que em todos os algoritmos identificaram positivamente a peça pretendida. Em data driven o algoritmo transições apresenta dificuldades em identificar a forma.

5.4.3. Caso 3 – Identificação da correcta orientação das reentrâncias de uma peça quadrangular

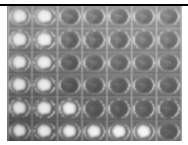
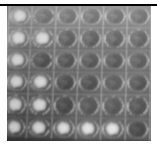
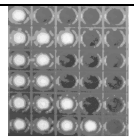
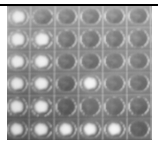
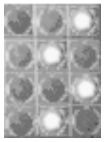
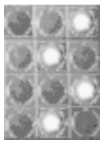
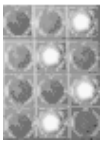
Tabela 5.3. Ensaio experimental - Caso 3 peça quadrado com reentrâncias

Ensaio	 (100% dos casos)	 (60% dos casos)	 (40% dos casos)	 (70% dos casos)
Quadrantes				
Sobreposição				
String Matching				
Transições				

O algoritmo apresentou uma boa eficiência na identificação das peças, em time driven, após ter sido realizado alguns ajustes nos seus parâmetros de forma a ser compatível com o modelo de ruído real. Em data driven os resultados não foram bons, caindo na zona de incerteza.

5.4.4. Caso 4 – Identificação da correcta orientação de um orifício numa peça triangular

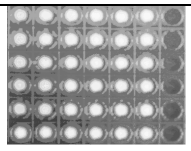
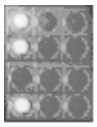
Tabela 5.4. Ensaio experimental - Caso 4 peça triangular com um orifício

Ensaio (superfície)	 (60% dos casos)	 (80% dos casos)	 (100% dos casos)	 (40% dos casos)
Quadrantes				
Sobreposição				
String Matching				
Transições				

O algoritmo que melhor resultados apresenta é o quadrantes. De uma forma generalista pode-se concluir que este algoritmo serve o propósito para qual foi concebido. Idealmente seria interessante ajustar os parâmetros no string matching e transições.

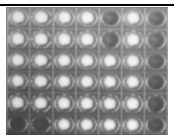
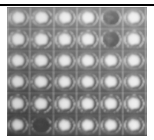
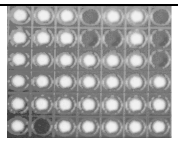
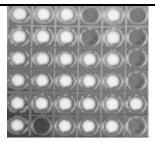
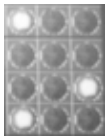
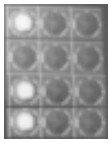
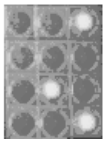
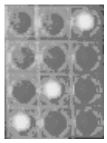
5.4.5. Caso 5 – Identificação da correcta orientação de uma peça rectangular

Tabela 5.5. Ensaio experimental - Caso 5 peça rectangular

Ensaio (superfície)	 (100% dos casos)	
Quadrantes		
Sobreposição		
String Matching		
Transições		

5.4.6. Caso 6 – Identificação da correcta orientação de dois orifícios de uma peça quadrangular

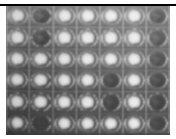
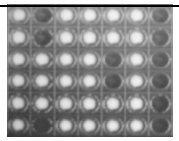
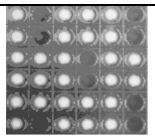
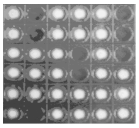
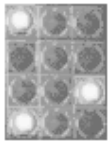
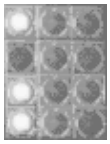
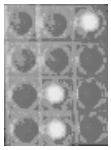
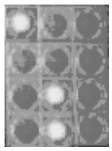
Tabela 5.6. Ensaio experimental - Caso 6 peça quadrangular com dois orifícios

Ensaio				
	(100% dos casos)	(70% dos casos)	(60% dos casos)	(60% dos casos)
Quadrantes Sobreposição String Matching Transições				

Neste algoritmo foi necessário proceder a uma alteração de um dos parâmetros. O facto de um dos orifícios encontrar-se na zona inferior da peça, zona sensível dado a sua relativa pequena dimensão e que adicionado e também o facto do primeiro sensor estar numa posição elevada, traduz na obtenção dum resultado diferente do esperado. Em data driven o algoritmo quadrantes funciona mal.

5.4.7. Caso 7 – Identificação da correcta orientação de três orifícios de uma peça quadrangular

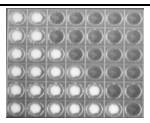
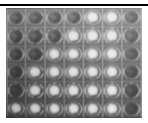
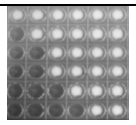
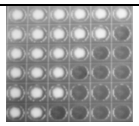
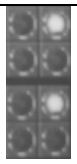
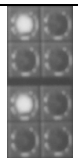
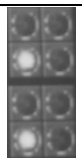
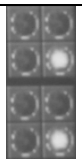
Tabela 5.7. Ensaio experimental - Caso 7 peça quadrangular com três orifícios

Ensaio				
	(100% dos casos)	(80% dos casos)	(50% dos casos)	(60% dos casos)
Quadrantes Sobreposição String Matching Transições				

Neste algoritmo também foi necessário proceder a uma alteração de um dos parâmetros pelos mesmos motivos apresentados para o caso 6. O algoritmo que melhor desenpenho apresenta é o quadrantes. Os algoritmos transições e string matching funcionam particularmente mal no dados adquiridos por data driven.

5.4.8. Caso 8 – Orientação possível de uma marca triangular

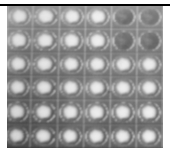
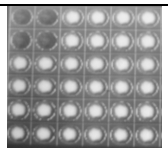
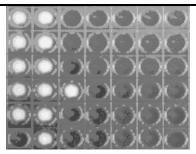
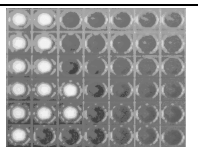
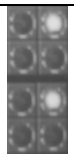
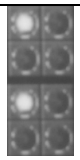
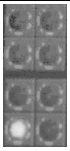
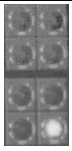
Tabela 5.8. Ensaio experimental - Caso 8 orientação de marca triangular

Ensaio				
	(70% dos casos)	(90% dos casos)	(80% dos casos)	(60% dos casos)
Quadrantes				
Transições				

Pode dizer-se que os algoritmo obtém óptimos resultados para os modelos de ruído reais obtidos.

5.4.9. Caso 9 – Orientação possível de uma marca quadrangular

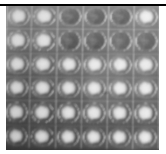
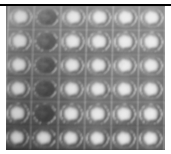
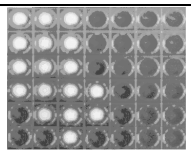
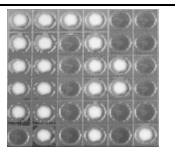
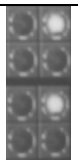
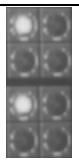
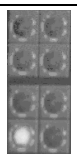
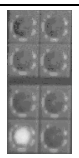
Tabela 5.9. Ensaio experimental - Caso 9 orientação de marca quadrangular

Ensaio				
	(100% dos casos)	(70% dos casos)	(60% dos casos)	(80% dos casos)
Quadrantes				
Transições				

Os algoritmos funcionam bem para dados obtidos em time driven. Já na presença de dados obtidos em data driven apenas o algoritmo transições responde correctamente. O quadrantes fica em zona de incerteza.

5.4.10. Caso 10 – Orientação possível de uma marca quadrangular

Tabela 5.10. Ensaio experimental - Caso 10 orientação de marca rectangular

Ensaio				
	(90% dos casos)	(80% dos casos)	(60% dos casos)	(70% dos casos)
Quadrantes				
Transições				

Tal como no caso anterior, os algoritmos funcionam bem para dados obtidos em time driven. Já na presença de dados obtidos em data driven o algoritmo quadrantes fica em zona de incerteza e o transições responde correctamente em certos casos, mentindo nos restantes.

5.5. Conclusões

De uma forma geral pode se concluir que a parte experimental obteve bons resultados uma vez que os algoritmos foram capazes de identificar as formas pretendidas, assim como sinalizar a orientação de determinadas marcas na peça. Também se verificou que o ruído não interferiu de forma significativa nos resultados, apesar de ter obrigado a alguns ajustes e flexibilização nos parâmetros dos algoritmos. Um outro aspecto, mais estrutural, e que independente do acabamento superficial das peças ou do processo de aquisição, é o facto do primeiro sensor se encontra a uma cota elevada interferindo assim com a imagem captada.

Desta experimentação pode-se concluir-se que as superfícies utilizadas (superfície espelhada e filme de prata) não tiveram influência significativa nos dados adquiridos, pelo que poderá ser interessantes que em futuras investigações se testem outras superfícies reflectoras de “pior” qualidade.

Relativamente ao processo de aquisição de dados, pode concluir-se que o modo Time-driven foi relativamente bem conseguido. Já o módulo Data-driven carece de aperfeiçoamentos. Neste contexto será necessário também proceder-se ao estudo ainda mais aprofundado de algoritmos específicos para aquisição de dados em Data Driven.

6. Conclusões

O presente relatório teve como objectivo apresentar o estudo elaborado na dissertação - Investigação Experimental de Técnicas de Reconhecimento de Padrões em Sistemas Automáticos. Neste contexto são actualmente inúmeras as aplicações industriais que necessitam de sistemas de visão, muitas vezes integrados em sistemas de reconhecimento de padrões. Este facto deve-se à cada vez maior necessidade (competitiva) de dotarem as máquinas de comportamentos inteligentes, capazes de percepcionarem e agirem sobre o meio envolvente.

O sistema aqui desenvolvido teve por objectivo ser capaz de reconhecer determinadas formas simples e orientações de determinado objecto, através da aquisição/sensorização do mesmo por varrimento de um conjunto de sensores e em determinadas condições de funcionamento (requisitos de velocidade baixa e não variáveis). Os resultados obtidos são animadores e permitem afirmar que para certos casos pode evitar-se a utilização de outros meios mais sofisticados e necessariamente mais caros.

A imagem binária adquirida é armazenada em memória do PLC ficando aí disponível para ser submetida análise por intermédio de diversos algoritmos, desenvolvidos com base os métodos estruturais (aqueles que para a aplicação em concreto mostraram ser os mais adequados). Aqui a principal dificuldade prende-se com a correcta formalização dos algoritmos.

O desenvolvimento de uma plataforma de simulação em CoDeSys (SoftPLC) enquanto ferramenta para manipulação e teste dos dados, permitiu apoiar a compreensão dos diferentes padrões e estabelecer os critérios de identificação das formas. A sua semelhança com o software de programação do PLC também se mostrou muito vantajoso, pois permitiu transportar diversos programas apenas realizando pequenas alterações no código. Já a experimentação permitiu a verificação da validade dos resultados obtidos na simulação, dado que a simulação, por si só, seria insuficiente devido ao desconhecimento das condições reais de funcionamento. O ruído, proveniente das superfícies utilizadas (superfície espelhada e superfície de filme de prata) assim

como do processo de aquisição de dados, não interferiu de forma significativa nos resultados, apesar de ter obrigado a alguns ajustes e flexibilização nos parâmetros dos algoritmos.

De uma forma geral pode dizer-se que a parte experimental obteve bons resultados uma vez que os algoritmos foram capazes de identificar e sinalizar a orientação de determinadas formas. Poderá ser interessantes em futuras investigações se testarem outras superfícies reflectoras de "pior" qualidade, assim como estudar em maior pormenor a aquisição de dados em Data Driven.

Bibliografia

Aksoy, Selim (2008), *Structural and Syntactic Pattern Recognition*, Department of Computer Engineering Bilkent University

Braggins, (1996), *Machine Vision Systems* (http://homepages.inf.ed.ac.uk/rbf/ CVonline/LOCAL_COPIES/ECVNET/Industrial.Vision.html)

CoDeSys (2010), Controller Development System, Consultado em Agosto de 2010, de http//.www.3s-software.com.

Dempster, A., Laird, N., Rubin, D. (1977), *Maximum Likelihood from Incomplete Data via the E-M Algorithm*, Journal of the Royal Statistical Society, série B, 39, pp. 1-38

Duda, R.O., Hart, P.E. (1973), *Pattern Classification and Scene Analysis*, Wiley -Interscience, New York.

Dunn, Rosalie A. (2008), Celebrating 40 years of Pattern Recognition – Reflections. Editorial / Pattern Recognition 41 (2008) 2139 – 2144, Elsevier Science Inc..

Everson, R. M. e Fieldsend, J. E. (2006). Multi-class roc analysis from a multi-objective optimisation perspective. Pattern Recognition Letters, 27:918_927.

Friedman, Menahem e Kandel, Abraham (1999), *Introduction to Pattern Recognition: Statistical, Structural, Neural, and Fuzzy Logic Approaches*. World Scientific, Singapore,

Fukunaga, K. (1990), *Introduction to Statistical Pattern Recognition*, Academic Press

Fukunaga, K. (1993), *Statistical Pattern Recognition*, Handbook of Pattern Recognition & Computer Vision, World Scientific

Grivet, Marco (2002), Centro de Estudos em Telecomunicações CETUC-PUC/Rio, http://www.lncc.br/~biologia/english/downloads/ReconhecimentoPadroes.pdf

Haykin, S. (1999), "Neural Networks: A Comprehensive Foundation", 2nd edition, Prentice-Hall.

Howson, Colin e Urbach, Peter (1993). *Scientific Reasoning: The Bayesian Approach*. Open Court. ISBN 9780812692341.

Hugh Jack, (2007), *Automating Manufacturing Systems with PLCs,* Version 5.0, May 4, - http://claymore.engineer.gvsu.edu/~jackh/books.html

International Electrotechnical Commission (IEC). (2007). IEC 61131 Standard, *Part 3 Textual Languages (IEC 61131-3).*

Lipschutz, Seymour e Lipson, Marc Lars (2007), Schaum's Outline of Discrete Mathematics, Theory and Problems of Discrete Mathematics, third edition, McGRAW-HILL.

Lopes, Alcino e Ribeiro, Fernando, (2009), Trabalho prático de Computação Industrial: *Sistema de identificação da orientação de peças*

Lotfi A. Zadeh, (1965), Fuzzy sets, Inf. Control 8, 338-353

Mardia, K., Kent, J., Bibby, J. (1979), *Multivariate Analysis*, Academic Press

Marques de Sá, J. P. (2000), Reconhecimento de Padrões, *http://paginas.fe.up.pt/~jmsa/recpad/index.htm*

Marques de Sá, Joaquim Pontes (2001),*Pattern recognition : concepts, methods and applications*, Berlin, Springer, cop.

Marques, J.S., (1999), *Reconhecimento de Padrões – métodos estatísticos e neuronais*, IST Press, pp.3-15

McQueen, J. (1967), *Some methods for Classification and Analysis of Multivariate Observations*, Proceedings of 5th Berkeley Symposium on Mathematics Statistics and Probabilities, vol. I, California University Press, pp. 281-286

Nadler, Morton e Smith, Eric P.. (1993), *Pattern Recognition Engineering*. Wiley, New York,

Olszewski, Robert T. (2001), *Generalized Feature Extraction for Structural Pattern Recognition in TimeSeries Data, (Doctoral dissertation)* (CMU-CS-01-108)

Pavlidis, Theo (2003), Pattern Recognition Letters, 36 years on the pattern recognition front; Lecture given at ICPR'2000 in Barcelona, Spain on the occasion of receiving the K.S. Fu prize, Elsevier Science Inc..

Rojas, R. (1996), *Neural Networks: a systematic introduction*, Springer-Verlag, Berlin

Teorema de Kleene. (2010). –Na Wikipédia. Consultado em Agosto de 2010, de http://pt.wikipedia.org/wiki/Stephen_Kleene

Manuais Técnicos da OMRON - PLC:

CP1L_Datasheet (P20E-EN-01)

CP1L_Operation_Manual (W462-E1-02)

CP1L_Programming_Manual (W451-E1-03)

Manuais técnicos da OMRON - HMI

V076-E1-02+NS+Tutorial_Manual

7. Anexos

Anexo 1

Programação da Plataforma de simulação

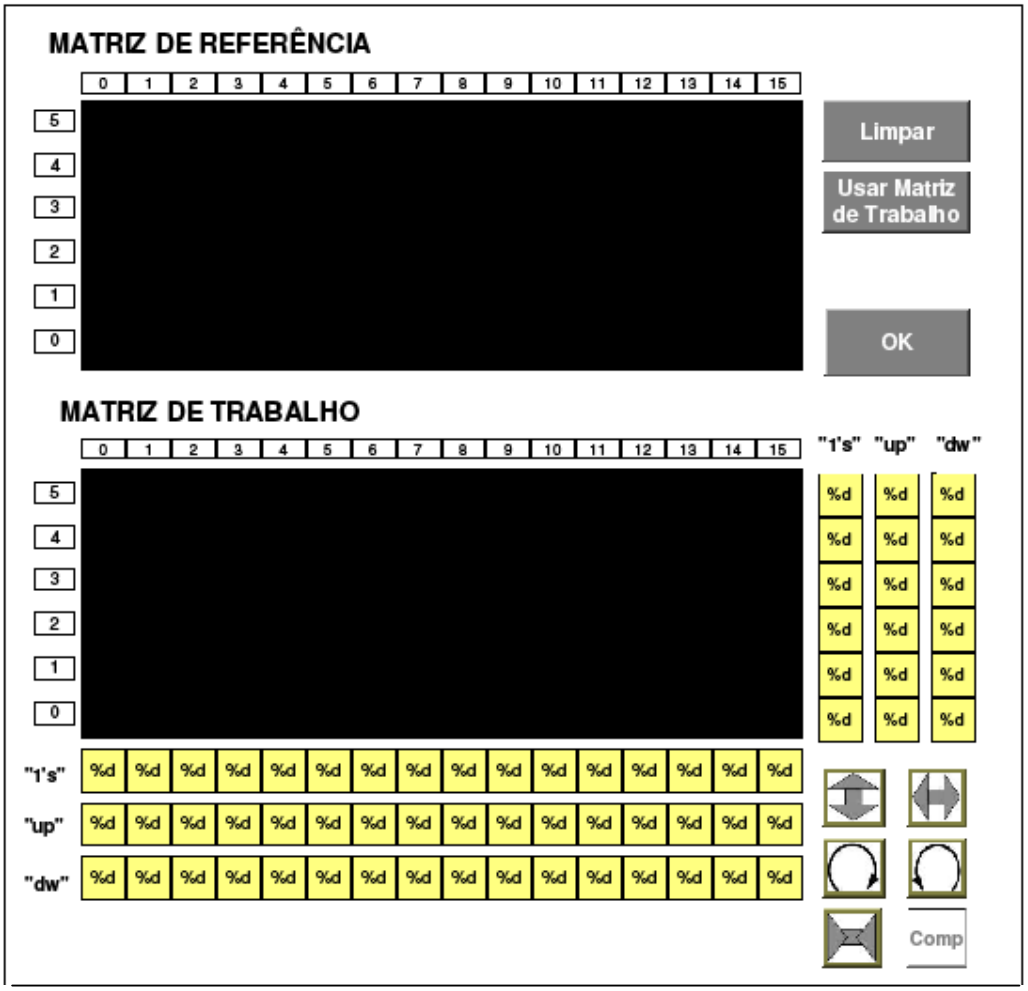
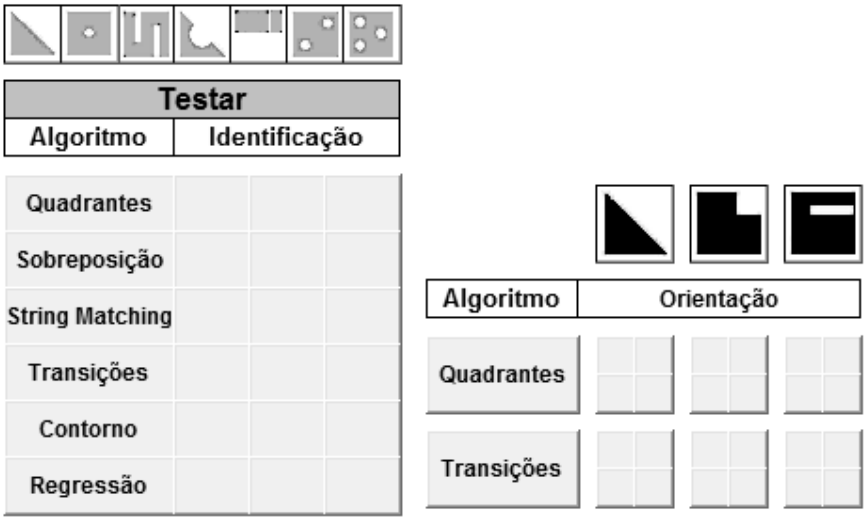


Figura A.1. HMI de simulação



a) b)

Figura A.2. a) resultados da identificação, b) resultados da orientação

```
0001 PROGRAM PLC_PRG
0002
0003 VAR
0004
0005 (* Estado dos botões na consola *)
0006     bt_limpar: BOOL;        (* Estado do botão "Clear all" *)
0007     bt_ok: BOOL;            (* Estado do botão "OK" *)
0008     bt_updw: BOOL;          (* Estado do botão "Up down" *)
0009     bt_rglf: BOOL;              (* Estado do botão "Right Left" *)
0010     bt_ck: BOOL;            (* Estado do botão "CK" *)
0011     bt_cck: BOOL;               (* Estado do botão "CCK" *)
0012     bt_testar: BOOL;        (* Estado do botão "Testar" *)
0013     bt_comparar: BOOL;          (* Estado do botão "Comparar" *)
0014     bt_sqz: BOOL;               (* Estado do botão "Squeezer" *)
0015     bt_novo : BOOL;         (* Estado do botão "NOVO" *)
0016
0017 (* Detecção da actuação dos botões da consola -> Transições "rising" ou "following"*)
0018     R_bt_limpar: R_TRIG;    (* Função usada na detecção da actuação do botão "Clear all" *)
0019     R_bt_ok: R_TRIG;            (* Função usada na detecção da actuação do botão "OK" *)
0020     R_bt_updw: R_TRIG;          (* Função usada na detecção da actuação do botão "Up down" *)
0021     R_bt_rglf: R_TRIG;      (* Função usada na detecção da actuação do botão "Right Left" *)
0022     R_bt_ck: R_TRIG;            (* Função usada na detecção da actuação do botão "CK" *)
0023     R_bt_cck: R_TRIG;       (* Função usada na detecção da actuação do botão "CCK" *)
0024     R_bt_testar: R_TRIG;        (* Função usada na detecção da actuação do botão "Testa" *)
0025     R_bt_comparar: R_TRIG;(* Função usada na detecção da actuação do botão "Comparar" *)
0026     R_bt_sqz: R_TRIG;       (* Função usada na detecção da actuação do botão "Squeeze" *)
0027     R_bt_novo: R_TRIG;      (* Função usada na detecção da actuação do botão "NOVO" *)
0028     F_bt_comparar: F_TRIG;      (* Função usada na detecção da libertação do botão "Comparar" *)
0029     F_bt_sqz: F_TRIG;       (* Função usada na detecção da libertação do botão "Squeeze" *)
0030
0031 (* Detecção da actuação dos botões da consola -> Transições "rising" ou "following" - selecção do caso a testar *)
0032     R_bt_caso1: R_TRIG; R_bt_caso2: R_TRIG; R_bt_caso3: R_TRIG; R_bt_caso4: R_TRIG; R_bt_caso5: R_TRIG;
0033     R_bt_caso6: R_TRIG; R_bt_caso7: R_TRIG; R_bt_caso8: R_TRIG; R_bt_caso9: R_TRIG; R_bt_caso10: R_TRIG;
0034     esconder_menus : BOOL;      (* Limitar menus disponiveis  *)
0035     colunas: INT;               (* Valor temporario, para salvaguarda de "amostras_coluna" em "squeeze" e "comparar" *)
0036     linhas: INT;                (* Valor temporario, para salvaguarda de "amostras_linha" em "squeeze" e "comparar" *)
0037
0038 (* coisas provisórias - problemas de orientação*)
0039     OK: BOOL;
0040     tri: BOOL;
0041     qua: BOOL;
0042     ko: BOOL;
0043
0044 (* variáveis auxiliares - para visualização dos resultados dos algoritmos - Critérios: "OK!" , "KO!" e "?" respectivamente *)
0045     alg_1: BOOL; alg_1_ok: BOOL; alg_1_ko: BOOL; alg_1_d: BOOL;
0046     alg_2: BOOL; alg_2_ok: BOOL; alg_2_ko: BOOL; alg_2_d: BOOL;
0047     alg_3: BOOL; alg_3_ok: BOOL; alg_3_ko: BOOL; alg_3_d: BOOL;
0048     alg_4: BOOL; alg_4_ok: BOOL; alg_4_ko: BOOL; alg_4_d: BOOL;
0049     alg_5: BOOL; alg_5_ok: BOOL; alg_5_ko: BOOL; alg_5_d: BOOL;
0050     alg_6: BOOL; alg_6_ok: BOOL; alg_6_ko: BOOL; alg_6_d: BOOL;
0051
0052 END_VAR
```

```
0001
0002 (* Detecção da actuação dos botões da consola *)
0003     R_bt_limpar (clk := bt_limpar);         (* Função de detecção da actuação do botão "Clear all" *)
0004     R_bt_ok (clk := bt_ok);                 (* Função de detecção da actuação do botão "OK" *)
0005     R_bt_updw (clk := bt_updw);                 (* Função de detecção da actuação do botão "Up down" *)
0006     R_bt_rglf (clk := bt_rglf);             (* Função de detecção da actuação do botão "Right Left" *)
0007     R_bt_ck (clk := bt_ck);                     (* Função de detecção da actuação do botão "CK" *)
0008     R_bt_cck (clk := bt_cck);               (* Função de detecção da actuação do botão "CCK" *)
0009     R_bt_testar (clk := bt_testar);             (* Função de detecção da actuação do botão "Contar «1's»" *)
0010     R_bt_comparar (clk := bt_comparar);     (* Função de detecção da actuação do botão "Comparar" *)
0011     R_bt_sqz (clk := bt_sqz);                   (* Função de detecção da actuação do botão "Squeeze" *)
0012     R_bt_novo (clk := bt_novo);             (* Função de detecção da actuação do botão "NOVO" *)
```

Figura A.3. Algoritmo geral do programa – parte 1

```
0013
0014 (*detecção da actuação dos botões da consola - de detecção do caso pretendido *)
0015     R_bt_caso1 (clk := bt_caso1); R_bt_caso2 (clk := bt_caso2); R_bt_caso3 (clk := bt_caso3); R_bt_caso4 (clk := bt_caso4);
0016     R_bt_caso5 (clk := bt_caso5); R_bt_caso6 (clk := bt_caso6); R_bt_caso7 (clk := bt_caso7); R_bt_caso8 (clk := bt_caso8);
0017     R_bt_caso9 (clk := bt_caso9); R_bt_caso10 (clk := bt_caso10);
0018
0019 (* Selecção exclusiva dos botões de escolha de caso da consola *)
0020     IF R_bt_caso1.Q THEN       (*actuação do botão "caso1")
0021 bt_caso2 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso9 := 0; bt_caso10 := 0;
0022     END_IF
0023     IF R_bt_caso2.Q THEN       (*actuação do botão "caso2")
0024 bt_caso1 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso9 := 0; bt_caso10 := 0;
0025     END_IF
0026     IF R_bt_caso3.Q THEN       (*actuação do botão "caso3")
0027 bt_caso1 := 0; bt_caso2 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso9 := 0; bt_caso10 := 0;
0028     END_IF
0029     IF R_bt_caso4.Q THEN       (*actuação do botão "caso4")
0030 bt_caso1 := 0; bt_caso2 := 0; bt_caso3 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso9 := 0; bt_caso10 := 0;
0031     END_IF
0032     IF R_bt_caso5.Q THEN       (*actuação do botão "caso5")
0033 bt_caso1 := 0; bt_caso2 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso9 := 0; bt_caso10 := 0;
0034     END_IF
0035     IF R_bt_caso6.Q THEN       (*actuação do botão "caso6")
0036 bt_caso1 := 0; bt_caso2 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso9 := 0; bt_caso10 := 0;
0037     END_IF
0038     IF R_bt_caso7.Q THEN       (*actuação do botão "caso7")
0039 bt_caso1 := 0; bt_caso2 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso8 := 0; bt_caso9 := 0; bt_caso10 := 0;
0040     END_IF
0041     IF R_bt_caso8.Q THEN       (*actuação do botão "caso7")
0042 bt_caso1 := 0; bt_caso2 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso9 := 0; bt_caso10 := 0;
0043     END_IF
0044     IF R_bt_caso9.Q THEN       (*actuação do botão "caso7")
0045 bt_caso1 := 0; bt_caso2 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso10 := 0;
0046     END_IF
0047     IF R_bt_caso10.Q THEN        (*actuação do botão "caso7")
0048 bt_caso1 := 0; bt_caso2 := 0; bt_caso3 := 0; bt_caso4 := 0; bt_caso5 := 0; bt_caso6 := 0; bt_caso7 := 0; bt_caso8 := 0; bt_caso9 := 0;
0049     END_IF
0050
0051 (* Define se deve haver ou não menus escondidos na consola *)
0052     esconder_menus := bt_comparar OR bt_sqz;  (* Limita menus  acessiveis quando "squeeze" ou "comparar" estão activos *)
0053
0054
0055 (* Resposta à actuação de botões  *)
0056
0057 IF R_bt_limpar.Q THEN          (* Botão "Limpar" foi premido *)
0058     Limpar_Ref();                  (* Padrão de referência inicailizado com zeros *)
0059 END_IF
0060
0061 IF R_bt_ok.Q THEN              (* Botão "OK" foi premido *)
0062     bt_sqz := FALSE;               (* Força libertação dos botões "squeeze" e "Comparar" *)
0063     bt_comparar := FALSE;
0064 alg_1_aux := FALSE; algor_2_aux := FALSE; alg_3_aux := FALSE; alg_4_aux := FALSE; alg_5_aux := FALSE; alg_6_aux := FALSE;
0065     Aceitar();                     (* Padrão de trabalho inicializado com o padrão de referência *)
0066     Contagem_total();                  (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0067 END_IF
0068
0069 IF R_bt_novo.Q THEN            (* Botão "Usar Matriz de Trabalho" foi premido *)
0070     Novo();                        (* O padrão de trabalho é adoptado como referência *)
0071     Contagem_total();              (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0072     bt_sqz := FALSE;               (* Força libertação dos botões "squeeze" e "Comparar" *)
0073     bt_comparar := FALSE;
0074 END_IF
0075
0076 IF R_bt_rglf.Q THEN            (* Botão "Right Left" foi premido *)
```

Figura A.4. Algoritmo geral do programa – parte 2

```
0077 alg_1_aux := FALSE; algor_2_aux := FALSE; alg_3_aux := FALSE; alg_4_aux := FALSE; alg_5_aux := FALSE; alg_6_aux := FALSE;
0078     Flip_vertical();                    (* Padrão de trabalho rodado segundo o eixo vertical *)
0079     Contagem_total();                   (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0080 END_IF;
0081
0082 IF R_bt_updw.Q THEN             (* Botão "Up down" foi premido *)
0083 alg_1_aux := FALSE; algor_2_aux := FALSE; alg_3_aux := FALSE; alg_4_aux := FALSE; alg_5_aux := FALSE; alg_6_aux := FALSE;
0084     Flip_horizontal();                  (* Padrão de trabalho rodado segundo o eixo horizontal *)
0085     Contagem_total();                   (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0086 END_IF
0087
0088 IF R_bt_ck.Q THEN               (* Botão "Ck" foi premido *)
0089 alg_1_aux := FALSE; algor_2_aux := FALSE; alg_3_aux := FALSE; alg_4_aux := FALSE; alg_5_aux := FALSE; alg_6_aux := FALSE;
0090     Roda_ck();                          (* Padrão de trabalho rodado 90º no sentido horário *)
0091     Contagem_total();                   (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0092 END_IF
0093
0094 IF R_bt_cck.Q THEN              (* Botão "CCk" foi premido *)
0095 alg_1_aux := FALSE; algor_2_aux := FALSE; alg_3_aux := FALSE; alg_4_aux := FALSE; alg_5_aux := FALSE; alg_6_aux := FALSE;
0096     Roda_cck();                         (* Padrão de trabalho rodado 90º no sentido anti-horário *)
0097     Contagem_total();                   (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0098 END_IF
0099
0100 IF R_bt_comparar.Q THEN        (* Botão "Comparar" foi premido *)
0101     Comparador();                       (* Matriz de trabalho mostar diferenças entre o padrão de referência e o de trabalho *)
0102     Contagem_total();                   (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0103 END_IF
0104
0105 IF R_bt_sqz.Q THEN              (* Botão "Squeeze" foi premido *)
0106     colunas := amostras_coluna;     (* Guarda número de amostras em colunas, para posterior reposição *)
0107     linhas := amostras_linha;       (* Guarda número de amostras em linhas, para posterior reposição *)
0108     squeeze();                      (* Constroi padrão assíncrono -> padraão de trabalho fico no padrão temporário *)
0109     Contagem_total();                   (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0110 END_IF
0111
0112 (* Resposta à reposição dos botões "Comparar" e "Squeeze" *)
0113     F_bt_comparar (clk := bt_comparar);    (* Função de detecção da actuação do botão "Comparar" *)
0114     F_bt_sqz (clk := bt_sqz);              (* Função de detecção da actuação do botão "Squeeze" *)
0115
0116     IF F_bt_comparar.Q AND NOT R_bt_novo.Q THEN  (* Botão "Comparar" volta a zero (não forçado)  *)
0117         repor();                            (* Padrão de trabalho é reposto *)
0118         Contagem_total();
0119     END_IF
0120
0121     IF F_bt_sqz.Q AND NOT R_bt_novo.Q THEN (* Botão "Squeezer" volta a zero (não forçado)  *)
0122         repor();                        (* Padrão de trabalho é reposto *)
0123         Contagem_total();                   (* Actualiza a contagem de 1's  para todas as linhas e colunas do padrão de trabalho *)
0124         amostras_coluna := colunas;     (* Repõe número de amostras em colunas *)
0125         amostras_linha := linhas;       (* Repõe número de amostras em linhas *)
0126     END_IF
0127
0128 (* ----- Execução do teste ------ *)
0129
0130 IF R_bt_testar.Q THEN           (* Botão "Testar" foi premido *)
0131
0132 (* ------ Testar algoritmos de identificação ------ *)
0133
0134 (* Algoritmo 1 - Quadrantes *)
0135 IF bt_caso1 OR  bt_caso3 OR bt_caso5 OR bt_caso6  THEN                    (* casos em que o algoritmo é aplicável*)
0136     alg_1 := INT_TO_BOOL (Quadrantes());
0137 END_IF
0138 IF bt_caso4 OR bt_caso7 THEN
0139     alg_1 := INT_TO_BOOL (sextante());
0140 END_IF
```

Figura A.5. Algoritmo geral do programa – parte 3

```
0141    IF alg_1 THEN
0142        alg_1_ok := TRUE; alg_1_ko := FALSE; alg_1_d :=FALSE;
0143    ELSE
0144        alg_1_ok := FALSE; alg_1_ko := TRUE; alg_1_d :=FALSE;
0145        IF alg_1_aux THEN
0146            alg_1_ko := FALSE; alg_1_d :=TRUE;
0147        END_IF
0148    END_IF
0149
0150    IF bt_caso2 OR bt_caso8 OR bt_caso9 OR bt_caso10 THEN                    (* casos em que o algoritmo não é aplicável*)
0151            alg_1_ok := FALSE; alg_1_ko := FALSE; alg_1_d := FALSE;
0152    END_IF
0153
0154 (* Algoritmo 2 - Regiões Comparadas*)
0155 IF bt_caso1 OR bt_caso2 OR bt_caso4 OR bt_caso5 THEN                          (* casos em que o algoritmo é aplicável*)
0156    alg_2 := INT_TO_BOOL (Comparados());
0157    IF alg_2 THEN
0158        alg_2_ok := TRUE; alg_2_ko := FALSE; alg_2_d :=FALSE;
0159    ELSE
0160        alg_2_ok := FALSE; alg_2_ko := TRUE; alg_2_d :=FALSE;
0161    END_IF
0162    IF algor_2_aux THEN
0163        alg_2_ok := FALSE; alg_2_ko := FALSE; alg_2_d :=TRUE;
0164    END_IF
0165 ELSE
0166    IF bt_caso3 OR bt_caso6 OR bt_caso7 OR bt_caso8 OR bt_caso9 OR bt_caso10 THEN(*casos em que o algoritmo não é aplicável*)
0167            alg_2_ok := FALSE; alg_2_ko := FALSE; alg_2_d := FALSE;
0168    END_IF
0169 END_IF
0170
0171 (* Algoritmo 3 - String Matching *)
0172 IF bt_caso1 OR bt_caso2 OR bt_caso3 OR bt_caso4 OR bt_caso6 OR bt_caso7 THEN
0173    alg_3 := INT_TO_BOOL (string_matching());
0174    IF alg_3 THEN
0175        alg_3_ok := TRUE; alg_3_ko := FALSE; alg_3_d :=FALSE;
0176    ELSE
0177        alg_3_ok := FALSE; alg_3_ko := TRUE; alg_3_d :=FALSE;
0178        IF  alg_3_aux THEN
0179            alg_3_ko := FALSE; alg_3_d :=TRUE;
0180        END_IF
0181    END_IF
0182 ELSE
0183    IF bt_caso5 OR bt_caso8 OR bt_caso9 OR bt_caso10 THEN                    (* casos em que o algoritmo não é aplicável*)
0184            alg_3_ok := FALSE; alg_3_ko := FALSE; alg_3_d := FALSE;
0185    END_IF
0186 END_IF
0187
0188 (* Algoritmo 4 - Transições *)
0189 IF bt_caso1 OR bt_caso2 OR bt_caso3 OR bt_caso4 OR bt_caso5 OR bt_caso6 OR bt_caso7 THEN
0190    alg_4 := INT_TO_BOOL (triangulos_2());
0191    IF alg_4 THEN
0192        alg_4_ok := TRUE; alg_4_ko := FALSE; alg_4_d :=FALSE;
0193    ELSE
0194        alg_4_ok := FALSE; alg_4_ko := TRUE; alg_4_d :=FALSE;
0195        IF  alg_4_aux THEN
0196            alg_4_ko := FALSE; alg_4_d :=TRUE;
0197        END_IF
0198    END_IF
0199 ELSE
0200    IF bt_caso8 OR bt_caso9 OR bt_caso10 THEN
0201        alg_4_ok := FALSE; alg_4_ko := FALSE; alg_4_d := FALSE;
0202    END_IF
0203 END_IF
0204
```

Figura A.6. Algoritmo geral do programa – parte 4

```
0205 (* Algoritmo 5 - Contorno Integral *)
0206
0207 IF bt_caso1 THEN
0208     alg_5 := INT_TO_BOOL (triangulo_contorno());
0209     IF alg_5 THEN
0210         alg_5_ok := TRUE; alg_5_ko := FALSE; alg_5_d :=FALSE;
0211     ELSE
0212         alg_5_ok := FALSE; alg_5_ko := TRUE; alg_5_d :=FALSE;
0213         IF  alg_5_aux THEN
0214             alg_5_ko := FALSE; alg_5_d :=TRUE;
0215         END_IF
0216     END_IF
0217 ELSE
0218     IF bt_caso2 OR bt_caso3 OR bt_caso4 OR bt_caso5 OR bt_caso6 OR bt_caso7 OR bt_caso8 OR bt_caso9 OR bt_caso10 THEN
0219             alg_5_ok := FALSE; alg_5_ko := FALSE; alg_5_d := FALSE;
0220     END_IF
0221 END_IF
0222
0223 (* Algoritmo 6 - Regressão *)
0224 IF bt_caso1 OR bt_caso2 THEN
0225     alg_6 := INT_TO_BOOL (Regressao());
0226     IF alg_6 THEN
0227         alg_6_ok := TRUE; alg_6_ko := FALSE; alg_6_d :=FALSE;
0228     ELSE
0229         alg_6_ok := FALSE; alg_6_ko := TRUE; alg_6_d :=FALSE;
0230         IF  alg_6_aux THEN
0231             alg_6_ko := FALSE; alg_6_d :=TRUE;
0232         END_IF
0233     END_IF
0234 ELSE
0235     IF bt_caso3 OR bt_caso4 OR bt_caso5 OR bt_caso6 OR bt_caso7 OR bt_caso8 OR bt_caso9 OR bt_caso10 THEN
0236             alg_6_ok := FALSE; alg_6_ko := FALSE; alg_6_d := FALSE;
0237     END_IF
0238 END_IF
0239
0240 IF bt_caso8 OR bt_caso9 OR bt_caso10 THEN
0241     Quadrantes();
0242     triangulos_2_orientacao();
0243     alg_1_ok := FALSE; alg_1_ko := FALSE; alg_1_d := FALSE;
0244     alg_4_ok := FALSE; alg_4_ko := FALSE; alg_4_d := FALSE;
0245 END_IF
0246
0247 END_IF
```

Figura A.7 Algoritmo geral do programa – parte 5

```
0001 VAR_GLOBAL CONSTANT
0002 (* Constante: qualquer padrão tem no máximo 6 linhas (numeradas de 0 a 5) que é o numero de sensores *)
0003     maximo_linhas : INT := 5;
0004 (* Constante: qualquer padrão tem no máximo 16 colunas (numeradas de 0 a 15) *)
0005     maximo_colunas : INT := 15;
0006 END_VAR
0007
0008 VAR_GLOBAL
0009
0010 (* Matrizes para representação de padrões *)
0011 (* Cada matriz é composta por 16, numerados de 0 a 15. Cada registo denota uma coluna.
0012 As colunas estão organizadas da esquerda para direita.  Cada registo contém 16 bits, numerados de 0 a 15.
0013 Cada bit represeta uma linhas. As linhas estão organizadas de baixo para cima. *)
0014
0015 (* NOTA IMPORTANTE: uma vez que só há 6 sensores, todas as linhas acima da 5 estão
0016 necesariamente a zero no padrão de referência *)
0017
0018     Matriz_a: ARRAY [0..maximo_colunas] OF WORD; (* Matriz de desenho (que serve de referência) *)
0019     Matriz_b: ARRAY [0..maximo_colunas] OF WORD; (* Matriz de trabalho *)
0020     Matriz_c: ARRAY [0..maximo_colunas] OF WORD; (* Matriz temporária para dados intermédios *)
0021
0022 (* Contadores de "1's" nas linhas e colunas da matriz de trabalho *)
0023
0024     Conta_coluna : ARRAY [0..maximo_colunas] OF INT;
0025     Conta_up_coluna : ARRAY [0..maximo_colunas] OF INT;
0026     Conta_dw_coluna : ARRAY [0..maximo_colunas] OF INT;
0027
0028     Conta_linha : ARRAY [0..maximo_linhas] OF INT;
0029     Conta_up_linha : ARRAY [0..maximo_linhas] OF INT;
0030     Conta_dw_linha : ARRAY [0..maximo_linhas] OF INT;
0031
0032     Amostras_coluna: INT; (* Inidica por quantas colunas é composto o padrão de referência *)
0033     Amostras_linha : INT; (* Inidica por quantas linhas é composto o padrão de referência *)
0034     Horizontal : BOOL; (* Flag que indica se a peça esta a ser representada na horizontal ou vertical na matriz de tabalho *)
0035
0036 (* Outras *)
0037
0038 (* Orientação - quadrantes do caso dos triagulos*)
0039     sd_tri: BOOL; se_tri: BOOL; id_tri: BOOL; ie_tri: BOOL;
0040 (* Orientação - quadrantes do caso dos quadrados*)
0041     sd_quad: BOOL; se_quad: BOOL; id_quad: BOOL; ie_quad: BOOL;
0042 (* Orientação - quadrantes do caso dos triagulos_2*)
0043     sd_tri_2: BOOL; ie_tri_2: BOOL; se_tri_2: BOOL; id_tri_2: BOOL;
0044
0045 (*variável auxiliar - para permitir identificar zona de incerteza *)
0046     alg_1_aux: BOOL; algor_2_aux: BOOL; alg_3_aux: BOOL; alg_4_aux: BOOL; alg_5_aux: BOOL; alg_6_aux: BOOL;
0047
0048 END_VAR
0049
0050
0051
```

Figura A.8. Declaração das variáveis globais do programa

```
0001 FUNCTION Novo : BOOL
0002
0003 (*   Padrão de trabalho é adoptado como referência *)
0004
0005 VAR
0006     mascara: WORD;              (* Registo temporário *)
0007     i: INT;                     (* Contador interno *)
0008 END_VAR
0001
0002 (* Copiar para a Matriz de Referência (Matriz_a) a Matriz de Trabalho (Matriz_b) *)
0003 Matriz_a:= Matriz_b;
0004
0005 (* mascara tem 1s nas primeras 6 linhas e zeros nas restantes - mascara corresponde a 63D = 111111B *)
0006 mascara := SHL(WORD#1,maximo_linhas +1)-1;
0007 FOR i:=0 TO maximo_colunas DO          (* Garante que todas as linhas acima da 5 estão em zero *)
0008     Matriz_a[i] := Matriz_a[i] AND mascara;
0009 END_FOR
0010
0011 (* Copia Matriz Referência para Matriz de Trabalho e refaz cálculos necessários *)
0012 Aceitar();
```

Figura A.9. Função “novo”

```
0001 FUNCTION Aceitar : BOOL
0002
0003 (*   Copia a Matriz de Referância para a Matriz de Trabalho *)
0004 (*   Calcula o número de colunas e linhas usadas pelo padrão *)
0005 (*   Não tem parâmetros de entrada  *)
0006
0007 VAR
0008     i: INT;                 (* Contador interno *)
0009     temp: WORD;             (* Registo temporario *)
0010 END_VAR
0001
0002
0003 Matriz_b := Matriz_a;          (* Obtém Matriz de Trabalho a partir da Matriz de Referência *)
0004 Horizontal := TRUE;            (* Define que a imagem está na horizontal *)
0005
0006 (* Calcula as colunas ocupadas pelo padrão, procura, da direita para a esquerda, uma coluna com conteudo *)
0007 (* No fim do FOR, "amostras_coluna" contém o número da última coluna usada pelo padrão (0 a 15)  *)
0008 FOR amostras_coluna := maximo_colunas TO 0 BY -1 DO
0009     IF Matriz_a[amostras_coluna] <> 0 OR amostras_coluna = 0 THEN
0010         EXIT;
0011     END_IF
0012 END_FOR
0013
0014 (* Calcula as linhas ocupadas pelo padrão *)
0015 (* Faz o "OR de todas as colunas usadas e poe resultado no registo "temp" *)
0016 temp := 0;
0017 FOR i := 0 TO amostras_coluna DO
0018     temp := temp OR Matriz_a[i];
0019 END_FOR
0020
0021 (* Percorre o registo "temp" de cima para baixo, procurando uma linha com conteúdo e quando encontra, pára*)
0022 (* No fim do FOR, "amostras_linha" contém o número de linhas usadas pelo padrão  *)
0023 FOR amostras_linha := maximo_linhas TO 0 BY -1 DO
0024     IF (temp AND  SHL (WORD#1,amostras_linha)) <> 0  OR amostras_linha = 0  THEN
0025         EXIT;
0026     END_IF
0027 END_FOR
0028
0029 (* Notar que é atribuida uma dimensão (0,0) a um "padrão vazio"
0030 a fim de evitar eventuais erros em caso de manipulação do mesmo  *)
```

Figura A.10. Função “Aceitar”

```
0001 FUNCTION Contagem_total : BOOL
0002
0003 (*   Contagem de "1's" e de transições "Up" e "Down" para todas as linhas e colunas da matriz de trabalho *)
0004 (*   Não tem parâmetros de entrada *)
0005
0006 VAR
0007     i, j: INT;                          (* Contadores internos *)
0008     temp, mascara, old, new : WORD;     (* Registos temporarios *)
0009 END_VAR
0001
0002
0003 (* Contagem dos "1's" ao longo de todas as colunas - incluindo as não ocupadas *)
0004
0005 FOR j:= 0 TO maximo_colunas DO              (* Analisa as colunas da esquerda para a direita *)
0006     Conta_coluna [j] := 0;                  (* Totalizador da coluna j a zero *)
0007     FOR i:=0 TO maximo_linhas DO            (* Para todas as linhas dessa coluna, de baixo para cima *)
0008         mascara := SHL(WORD#1,i);           (* Define mascara em função da linha  *)
0009         temp := Matriz_b[j] AND mascara;    (* Isola o bit da linha i, coluna j *)
0010         IF temp <> 0 THEN
0011             Conta_coluna [j] := Conta_coluna[j] +1;
0012         END_IF;
0013     END_FOR
0014 END_FOR
0015
0016
0017 (* Contagem dos "1's" ao longo de todas as linhas - incluindo as não ocupadas *)
0018
0019 FOR j:= 0 TO maximo_linhas DO              (* Analisa as linhas de baixo para cima *)
0020     Conta_linha [j] := 0;                   (* Totalizador da linha j a zero *)
0021     mascara := SHL(WORD#1,j);               (* Define mascara em conformidade com a linha *)
0022     FOR i:=0 TO maximo_colunas DO           (* Percorre as colunas da linha, da esquerda para a direita *)
0023         temp := Matriz_b[i] AND mascara;
0024         IF temp <> 0 THEN
0025             Conta_linha [j] := Conta_linha[j] +1;
0026         END_IF;
0027     END_FOR
0028 END_FOR
0029
0030
0031 (* Contagem de transições ascendentes e descendentes ao longo de todas as colunas - incluindo as não ocupadas *)
0032
0033 FOR i:= 0 TO maximo_colunas DO                  (* Analisa as colunas da esquerda para a direita *)
0034     conta_up_coluna[i] := 0;                    (* Respectivos totalizadores a zero *)
0035     conta_dw_coluna[i] := 0;
0036     old := 0;
0037     FOR j:= 0 TO maximo_linhas DO               (* Analisa as linhas de baixo para cima *)
0038         new := SHR((Matriz_b[i] AND SHL(WORD#1,j)), j);
0039         IF new > old THEN                       (* Houve transição de 0 para 1 *)
0040             conta_up_coluna[i] := conta_up_coluna[i] + 1;
0041         END_IF
0042         IF new < old THEN                       (* Houve transição de 1 para 0 *)
0043             conta_dw_coluna[i] := conta_dw_coluna[i] + 1;
0044         END_IF
0045         old := new;
0046     END_FOR
0047     IF old <>0 THEN                             (* Verifica se haveria outra tarsnição de 1 para 0 *)
0048         conta_dw_coluna[i] := conta_dw_coluna[i] + 1;
0049     END_IF
0050 END_FOR
0051
0052
```

Figura A.11. Função “Contagem_total”

```
0001 FUNCTION Flip_horizontal : BOOL
0002
0003 (*   Reflecte a Matriz de Trabalho segundo o eixo horizontal - i.e., vira-a de "cabeça para baixo". *)
0004 (*   Não tem parâmetros de entrada *)
0005
0006 VAR
0007     linhas: INT;                (* Número de linhas a considerar *)
0008     colunas: INT;               (* Número de colunas a considerar *)
0009     i, j: INT;                  (* Contadores Internos *)
0010     temp, mascara: WORD;        (* Registos temporários *)
0011 END_VAR
0001
0002 Matriz_c := Matriz_b;                          (* Guarda Matriz de Trabalho na Matriz C *)
0003
0004 IF horizontal THEN                             (* Se a peça está na horizontal *)
0005     linhas := amostras_linha;
0006     colunas := amostras_coluna;
0007 ELSE                                           (* Se a peça está na vertical *)
0008     linhas := amostras_coluna;
0009     colunas := amostras_linha;
0010 END_IF
0011
0012 FOR j :- 0 TO colunas DO                       (* Começa pela coluna da esquerda *)
0013     Matriz_b[j] := 0;
0014     mascara := 1;                              (* e pela linha de baixo *)
0015     FOR i:- 0 TO linhas DO
0016         temp:= Matriz_c[j] AND mascara;        (* isola o bit da linha inferior e... *)
0017         Matriz_b[j]:= SHL(Matriz_b[j],1);
0018         Matriz_b[j]:= Matriz_b[j] OR temp;     (* ... cola-o na Matriz_B *)
0019         Matriz_c[j] := SHR (Matriz_c[j], 1);   (* Desloca as linhas uma posição para baixo *)
0020     END_FOR
0021 END_FOR
0022
0023
0024
```

Figura A.12 Função “Flip_horizontal”

```
0001 FUNCTION Flip_vertical : BOOL
0002
0003 (*   Reflecte a Matriz de Trabalho segundo o eixo vertical - i.e., vira-a da "esquerda para a direita"...*)
0004 (*   Não tem parâmetros de entrada *)
0005
0006 VAR
0007     colunas: INT;          (* Número de colunas a considerar *)
0008     i: INT;                (* Contador interno *)
0009 END_VAR
0001
0002
0003 (* Variaveis usadas como no "flip horizontal *)
0004
0005 Matriz_c := Matriz_b;                  (* Guarda Matriz de Trabalho na Matriz Auxiliar C *)
0006
0007 IF horizontal THEN
0008     colunas := amostras_coluna;        (* Se a peça está na horizontal *)
0009 ELSE
0010     colunas := amostras_linha;         (* Se a peça está na vertical *)
0011 END_IF
0012
0013 (* Troca esquerda com direita *)
0014
0015 FOR i :- 0 TO colunas DO
0016     Matriz_b[i] := Matriz_c[colunas-i];
0017 END_FOR
0018
```

Figura A.13. Função “Flip_vertical”

```
0001 FUNCTION Limpar_Ref : BOOL
0002
0003 (*  Apaga a matriz de referência    *)
0004 (*  Não tem parâmetros de entrada *)
0005
0006 VAR
0007     j: INT;              (* Contador interno *)
0008 END_VAR
0001
0002
0003 (* Matriz de Referência posta a zero, coluna a coluna *)
0004
0005 FOR j:=0 TO maximo_colunas DO
0006     Matriz_a[j] := 0;
0007 END_FOR
```

Figura A.14. Função “Limpar_Ref”

```
0001 FUNCTION Roda_ck : BOOL
0002
0003 (*  Padrão de trabalho é rodado  90º no sentido horário.*)
0004 (*  Não tem parâmetros de entrada *)
0005
0006 VAR
0007     linhas: INT;              (* Número de linhas a considerar *)
0008     colunas: INT;             (* Número de colunas a considerar *)
0009     i, j: INT;                (* Contadores Internos *)
0010     temp, mascara: WORD; (* Registos temporários *)
0011 END_VAR
0001
0002 (* Rotação do padrão de trabalho no sentido horário *)
0003
0004 Matriz_c := Matriz_b;
0005
0006 IF horizontal THEN
0007     linhas := amostras_linha;
0008     colunas := amostras_coluna;
0009 ELSE
0010     linhas := amostras_coluna;
0011     colunas := amostras_linha;
0012 END_IF
0013
0014 FOR i:= 0 TO linhas DO
0015     mascara := SHL(WORD#1, i);
0016     Matriz_b[i] := 0;
0017     FOR j := 0 TO colunas DO
0018         temp := Matriz_c[j] AND mascara;
0019         IF temp <> 0 THEN
0020             Matriz_b[i]:=Matriz_b[i] OR SHL(WORD#1,colunas-j);
0021         END_IF
0022     END_FOR
0023 END_FOR
0024
0025 IF colunas > linhas THEN
0026     FOR i := colunas TO linhas+1 BY -1 DO
0027         Matriz_b[i] := 0;
0028     END_FOR
0029 END_IF
0030
0031 horizontal:= NOT horizontal;
0032
```

Figura A.15. Função “Roda_ck”

```
0001 FUNCTION Roda_cck : BOOL
0002
0003 (*   Padrão de trabalho é rodado  90º no sentido anti-horário *)
0004 (*   Não tem parâmetros de entrada *)
0005
0006 VAR
0007     linhas: INT;                (* Número de linhas a considerar *)
0008     colunas: INT;               (* Número de colunas a considerar *)
0009     i, j: INT;                  (* Contadores Internos *)
0010     temp, mascara: WORD; (* Registos temporários *)
0011 END_VAR
0001
0002 (* Rotação do padrão de trabalho no sentido anti-horário *)
0003 (* Guarda Matriz de Trabalho na Matriz C *)
0004 Matriz_c := Matriz_b;
0005
0006 IF horizontal THEN                        (* Se a peça está na horizontal *)
0007     linhas := amostras_linha;
0008     colunas := amostras_coluna;
0009 ELSE                                      (* Se a peça está na vertical *)
0010     linhas := amostras_coluna;
0011     colunas := amostras_linha;
0012 END_IF
0013
0014 FOR i:= linhas TO 0 BY -1 DO
0015     mascara := SHL(WORD#1, i);
0016     Matriz_b[linhas-i] := 0;
0017     FOR j := colunas TO 0 BY -1 DO
0018         temp := Matriz_c[j] AND mascara;
0019         IF temp <> 0 THEN
0020             Matriz_b[linhas-i]:=Matriz_b[linhas-i] OR SHL(WORD#1,j);
0021         END_IF
0022     END_FOR
0023 END_FOR
0024
0025 IF colunas > linhas THEN
0026     FOR i := colunas TO linhas+1 BY -1 DO
0027         Matriz_b[i] := 0;
0028     END_FOR
0029 END_IF
0030
0031
0032 horizontal:= NOT horizontal;
0033
```

Figura A.16. Função “Roda_cck”

```
0001 FUNCTION repor : BOOL
0002
0003 (*   Reposição da Matriz de Trabalho, B,  a partir da Matriz Temporária, C. *)
0004 (*   Não tem parâmetros de entrada *)
0001
0002 (* Reposição da matriz de trabalho *)
0003
0004 Matriz_b:= Matriz_c;
0005
0006
```

Figura A.17. Função “repor”

```
0001 FUNCTION Comparador : BOOL
0002
0003 (*   Assinala diferenças entre as Matrizes de Referência e de Trabalho.*)
0004 (*   Não tem parâmetros de entrada *)
0005
0006 VAR
0007     i: INT;                  (* Contador Interno *)
0008 END_VAR
0009
0001
0002 (* A Matriz Auxiliar (Matriz_c) guarda temporariamente os valores da Matriz de Trabalho (Matriz_b) *)
0003 (* A Matriz_b pode ser depois reposta quando chamada a função: "repor" *)
0004 Matriz_c := Matriz_b;
0005
0006 (* As diferenças correspondem ao "Ou exclusivo" das duas matrizes, realizado bit a bit *)
0007 FOR i:=0 TO amostras_coluna DO
0008     Matriz_b[i] := Matriz_b[i] XOR Matriz_a[i];
0009 END_FOR
0010
0011
0012
0013
```

Figura A.18. Função “Comparador”

```
0001 FUNCTION squeeze : BOOL
0002
0003 (*   No padrão de trabalho, retira colunas adjacentes com os mesmos valores - simula leituras "data driven". *)
0004 (*   Não tem parâmetros de entrada  *)
0005
0006 VAR
0007     colunas: INT;          (* Colunas a considerar *)
0008     i, j : INT;            (* Contadores internos *)
0009 END_VAR
0001
0002 (* Guarda padrão de trabalho na matriz auxilar temporária para futura reposição através da função: "repor" *)
0003 Matriz_c:= Matriz_b;
0004
0005 IF horizontal THEN                                   (* Se a peça está na horizontal *)
0006     colunas := amostras_coluna;
0007 ELSE                                                 (* Se a peça está na vertical *)
0008     colunas := amostras_linha;
0009     FOR i:= 0 TO colunas DO
0010         Matriz_b[i]:= Matriz_b[i] AND 2#111111;      (* Rejeita eventuais bits não visíveis *)
0011     END_FOR
0012 END_IF
0013
0014 FOR i := 0 TO colunas-1 DO                           (* Verifica, coluna a coluna, se uma coluna é igual a seguinte *)
0015     WHILE Matriz_b[i] = Matriz_b [i+1]  AND colunas > i DO
0016             retira (n:=i+1);                         (* Se for retira e faz nova verificação *)
0017             colunas := colunas -1;                   (* Desconta a coluna retirada *)
0018     END_WHILE;
0019 END_FOR
0020
0021
0022
0023
0024
0025
```

Figura A.19. Função “Squeez”

Anexo 2

Algoritmos

7.1.1. Caso 3 – Identificação da correcta orientação das reentrâncias de uma peça quadrangular

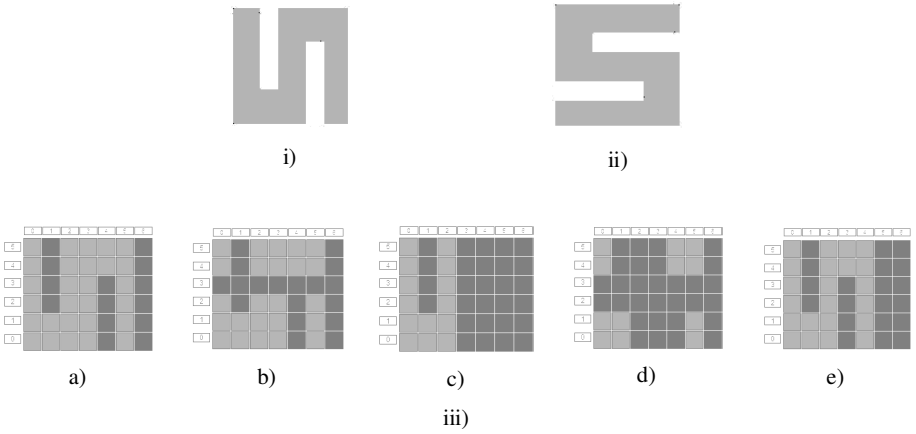


Figura A.20. Peça quadrangular com reentrâncias: i) forma pretendida, ii) forma de rejeição, iii) modelos de ruído testados.

Na figura 7.20. podem observar-se as formas pretendida e de rejeição em i) e ii) respectivamente e os modelos de ruído, considerados para o desenvolvimento e teste dos algoritmos em iii). Estes começam por representar a imagem ideal da forma pretendida no caso a), mas considerando o caso de uma avaria permanente de um dos sensores podemos ter algo semelhante ao caso exposto em b), por outro lado, no caso c) procura-se simular a falha temporária e de curta duração do conjunto de todos os sensores. Já no caso d) procura-se testar a sensibilidade e robustez dos algoritmos sinalizando as zonas mais sensíveis da peça. Por último procura-se reproduzir, no caso e), a resposta esperada da aquisição em *event driven*.

Método 1: Análise por regiões quadrantes

Este Algoritmo pretende analisar a imagem binária a partir da partição ortogonal em quatro regiões iguais, que designaremos de quadrantes. As regiões comparadas são:

- O quadrante inferior esquerdo com o quadrante superior direito, subtraindo o somatório de "1s" do primeiro (*ie*) aos do segundo (*sd*).
- O quadrante inferior direito com o quadrante superior esquerdo, subtraindo o somatório de "1s" do primeiro (*id*) aos do segundo (*se*).

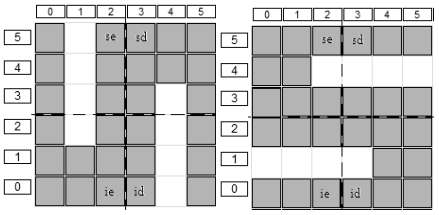


Figura A.21. Identificação por regiões (quadrantes)

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" **-** é a forma pretendido se existir um número consideravelmente maior de "1s" nos quadrantes *ie* e *sd* que nos quadrantes *se* e *id*. Ao mesmo tempo o número de "1s" nos quadrantes *se* e *id* bem como nos quadrantes *sd* e *ie* deverá ser aproximadamente igual, tal que:

$$sd > id \quad \wedge \quad ie > se \quad \wedge \quad \frac{se}{id+1} \leq 1{,}5 \quad \wedge \quad \frac{ie}{sd+1} \leq 1{,}5 \tag{13}$$

- Critério "KO!" – é a forma de rejeição se existir um número consideravelmente maior de "1s" nos quadrantes se e id que nos quadrantes sd e ie. Ao mesmo tempo o número de "1s" nos quadrantes se e id bem como nos quadrantes sd e ie deverá ser aproximadamente igual, tal que:

$$se > sd \quad \wedge \quad id > ie \quad \wedge \quad \frac{id}{se+1} \leq 1{,}5 \quad \wedge \quad \frac{se}{id+1} \leq 1{,}5 \tag{14}$$

- Critério "?" – quando não é possível assegurar qual a forma em presença. O valor de ruído admitido é superado, o que coloca a peça numa zona de incerteza.

Tipo de aquisição de dados aplicável: Time driven - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem. Data driven – aplicável se o número de amostragens da peça for positivo.

Os principais pontos fortes deste algoritmo são: ter um critério de dupla verificação e por outro lado ser pouco extenso (rápido). Já o seu ponto fraco é poder ser muito prejudicado se ocorrer uma "descentragem" da peça;

Método 2: Análise por sequência de transições

De uma forma geral este algoritmo identifica a forma da peça pela presença e sequência de um conjunto de transições esperadas em determinada zona da peça. As zonas determinantes são mais facilmente visualizadas se se recorrer à função comparar, desta forma pode dizer-se que integramos os algoritmos transições com o sobreposição.

Este algoritmo verifica o momento em que se dão as transições positivas na linha 1 e coluna 4, assim como as transições negativas na linha 4 e coluna 2.

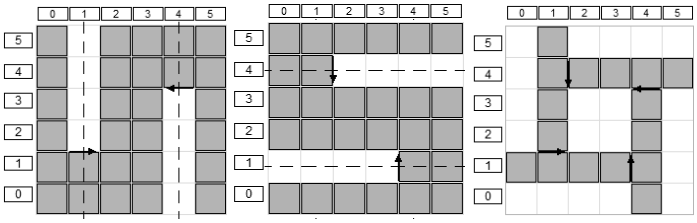


Figura A.22. Identificação por sequência de transições

Os critérios tomados em conta são os seguintes:

- Critério "OK!" – detectadas transições positivas nas posições 5 e 4 na linha 1 e coluna 4 respectivamente, assim como são detectadas transições negativas nas posições 6 e 2 na linha 4 e coluna 1 respectivamente.
- Critério "KO!" – detectadas transições positivas nas posições 4 e 5 na linha 1 e coluna 4 respectivamente, assim como são detectadas transições negativas nas posições 2 e 5 na linha 4 e coluna 1 respectivamente.
- Critério "?" – detecta mais ou menos transições que as estabelecidas nos critérios anteriores e por ordens diferentes.

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real. Tem por pontos forte ser um programa insensível ao ruído fora destas duas linhas e duas colunas, e com ponto fraco a sua sensibilidade ao ruído nas linhas e colunas analisadas

Método 3: Análise por String Matching

De uma forma geral este algoritmo identifica a forma da peça por comparação da sequência de leituras adquiridas pelo varrimento de sensores (colocada na matriz/espaço de amostragem), com a imagem binária pretendida e previamente armazenada em memória. Começa por procurar a palavra inicial t_0' no instante $t = t_0$, e sempre que tal acontece, verifica se a sequência pretendida é conseguida. Se sim, então a forma foi identificada, se não repete o processo para t = t+1.

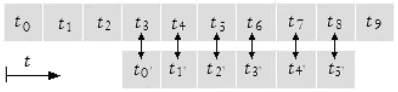


Figura A.23. Identificação por string matching

Os critérios tomados em conta são os seguintes:

- Critério "OK!" – a sequência de palavras que descreve a forma desejada é encontrada dentro da tolerância definida (uma letra diferente por palavra).
  Matriz_d[63, 3, 63, 63, 48, 63] ou [111111.000011.111111.111111.110000.111111]

- Critério "KO!" - **s**equência da forma de rejeição é encontrada dentro da tolerância definida (ruído equivalente a uma letra diferente por palavra).
  Matriz_e[61, 61, 45, 45, 47, 47] ou [111101.111101.101101.101101.101111.101111]

- Critério "?" - Sequencia encontrada com tolerância de duas ou mais letras por palavra.

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real (será possível em data driven se a imagem introduzida para comparação, for introduzida já com esse fim em vista e tenha por isso, a forma esperada para uma aquisição data driven da peça).

Tem por pontos forte ser um programa pequeno (por isso rápido) e como pontos fracos a possível ocorrência de erros na introdução dos dados iniciais e a sua sensibilidade a zonas de ruído concentrado.

7.1.2. Caso 5 – Identificação da correcta orientação de uma peça rectangular

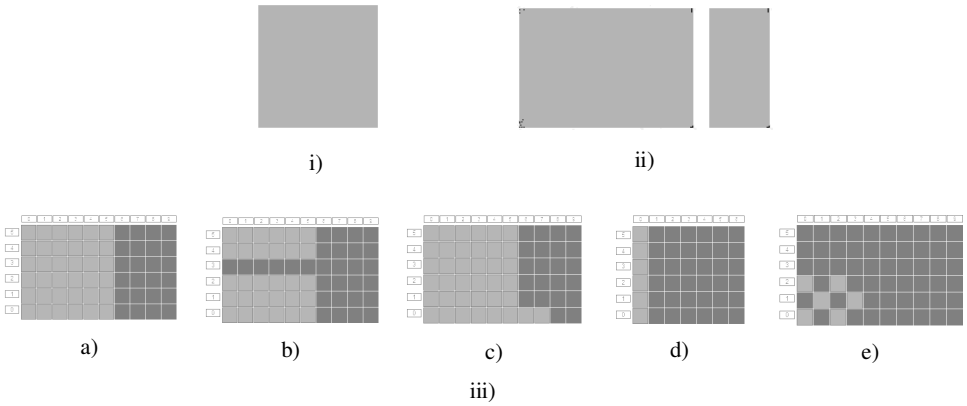


Figura A.24. Peça rectangular: i) forma pretendida, ii) forma de rejeição iii) modelos de ruído testados.

Na figura 7.24. podem observar-se as formas pretendida e de rejeição em i) e ii) respectivamente e os modelos de ruído, considerados para o desenvolvimento e teste dos algoritmos em iii). Estes começam por representar a imagem ideal da forma pretendida no caso a), mas considerando o caso de uma avaria permanente de um dos sensores podemos ter algo semelhante ao caso exposto em b), por outro lado, no caso c) procura-se simular a falha temporária e de curta duração do conjunto de quase todos os sensores. Já no caso e) procura-se testar a sensibilidade e robustez dos diversos algoritmos. Por último procura-se reproduzir, no caso d), a resposta esperada da aquisição em *event driven*.

Método 1: Quadrantes (variação de tamanho)

O algoritmo analisa a imagem binária a partir da qual extrai o número de amostragens recolhidas, depois identifica e compara o número de linhas e colunas ocupados, elemento inicial na definição dos quadrantes, de onde se podem calcular as áreas e o consequente número de "1s" teoricamente aí existentes.

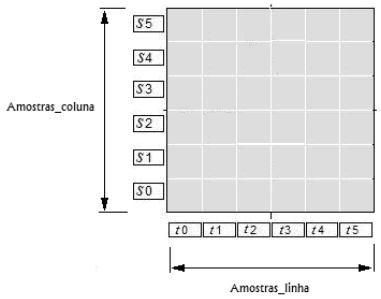


Figura A.25. Peça rectangular - dimensões

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – se o número de linhas e colunas existentes for igual.
- Critério "KO!" – se o número de linhas e colunas existentes for diferente em mais que uma unidade.
- Critério "?" – se o número de linhas e colunas existentes for diferente em apenas uma unidade.

Apenas aplicável em: Time driven - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem. Em data driven, e na ausência de ruído, como a secção da peça não apresenta variação de forma, não seria possível identificar a dimensão comprimento.

Este algoritmo tem com pontos forte o facto de ser um programa bastante curto. O seu ponto fraco é como apenas analisa a dimensão total da região, perdendo por isso a sensibilidade ao ruído existente.

Método 2: Análise por regiões comparadas

O algoritmo analisa a imagem binária a partir da sobreposição dos dois objectos que se pretendem identificar e, utilizando um OU Exclusivo, extrai a região que nessa comparação os mais descriminam (zona especifica e exclusiva de cada objecto), criando um "espaço de características": a forma que se pertence identificar é uma forma regular de lados todos iguais, como tal foi definida uma medida de referência dessa dimensão coincidente com a altura da peça (numero de linha utilizadas). Tendo por base esse valor é criada a região de referência, a qual é comparada com a forma adquirida.

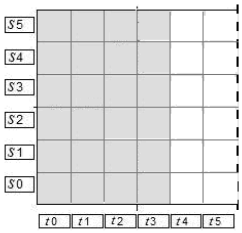


Figura A.26. Peça rectangular – áreas sobrepostas

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – O número de "1s" nas regiões de comparação é igual a zero.
- Critério "KO!" – O número de uns nas regiões comparadas não ultrapassam metade do número de "1s" de uma coluna.
- Critério "?" – quando o número de "1s" das regiões de comparação é superior a metada de uma coluna de dimensão de referência.

Aplicável em: Time driven - aplicável com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem, E em Data driven – aplicável dada a forma variável da peça que em conjunto com o movimento de passagem da peça perante os sensores de varrimento, provoca a detecção da variação e consequente aquisição para o espaço de amostragem.

Este algoritmo tem com pontos forte o facto de ser um programa curto (e por essa razão mais rápido) O seu ponto fraco é como apenas faz a análise na região considerada, não consegue garantir que efectivamente o corpo envolvente seja um triangulo (o que se poderia conseguir combinando um algoritmo de contorno).

Método 3: Análise por sequência de transições

Este algoritmo determina o valor médio em que se dão as transições negativas nas duas dimensões da forma. Se a média das transições das linhas acontecem primeiro que a das colunas então é um rectângulo vertival. Se as transições das colunas acontecerem primeiro então é um rectângulo horizontal. Se acontecerem mais ou menos ao mesmo tempo, então é considerado um quadrado.

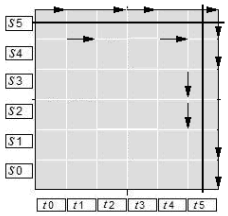


Figura A.27. Peça rectangular – média das transições

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – é a forma pertendida se a média das transições negativas nas linhas for igual à média das transições negativas nas colunas.

$$med\_nv = med\_nh \tag{21}$$

- Critério "KO!" - é a forma de rejeição se a média das transições negativas nas linhas for menor ou maior que uma unidade que a das transições negativas nas colunas

$$med\_nv > med\_nh + 1 \quad \vee \quad med\_nh > med\_nv + 1 \tag{22}$$

- Critério "?" – quando o valor de ruído admitido é superado, o que coloca a peça numa zona de incerteza.

Apenas em Time Driven – os critérios acima referidos só se aplicam para dimensões reais. Os principais pontos fortes deste algoritmo são: alguma tolerância ao ruído por ter por base a média de transições. Já o seu ponto fraco é não poder garantir a forma de quadrado.

7.1.3. Caso 6 – Identificação da correcta orientação de dois orifícios numa peça quadrangular

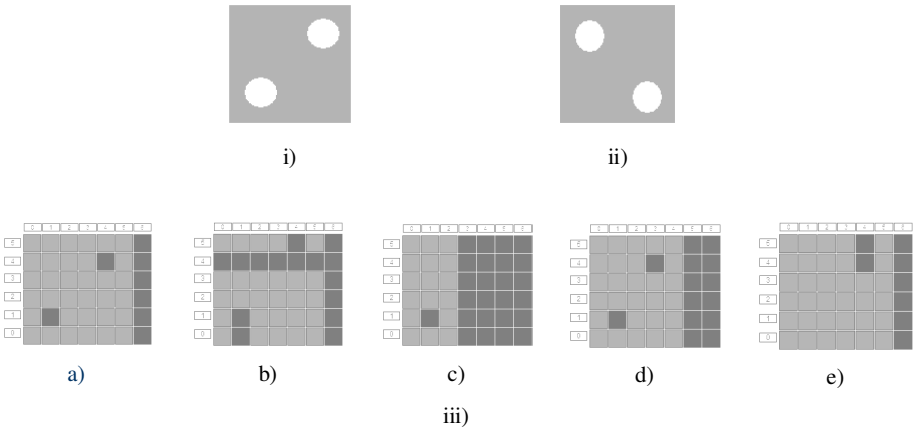


Figura A.28. Peça quadrangular com dois orifícios: i) forma pretendida, ii) forma de rejeição, iii) modelos de ruído testados.

Na figura 7.28. podem observar-se as formas pretendida e de rejeição em i) e ii) respectivamente e os modelos de ruído, considerados para o desenvolvimento e teste dos algoritmos em iii). Estes começam por representar a imagem ideal da forma pretendida, mas considerando o caso de uma avaria permanente de um dos sensores podemos ter algo semelhante ao caso exposto em b), por outro lado, no caso c) procura-se simular a falha temporária e de curta duração do conjunto de todos os sensores. Já no caso e) procura-se testar a sensibilidade e robustez dos algoritmos para o aparecimento de ruído na zona mais sensíveis da peça, mais concretamente na região de contorno do orifício da peça. Por último procura-se reproduzir, no caso d), a resposta esperada da aquisição em *event driven*.

Método 1: Análise por regiões quadrantes

Este Algoritmo pretende analisar a imagem binária a partir da partição ortogonal em quatro regiões iguais, que designaremos de quadrantes. As regiões comparadas são:

- O quadrante inferior esquerdo com o quadrante superior direito, somando o numero de uns do primeiro (*ie*) aos do segundo (*sd*).
- O quadrante inferior direito com o quadrante superior esquerdo, somando o numero de uns do primeiro (*id*) aos do segundo (*se*).

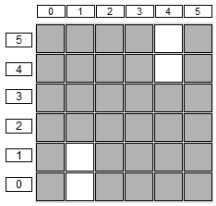


Figura A.29. Identificação por regiões (quadrantes)

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" **-** é o Quadrado pretendido se existir um número consideravelmente maior de "1s" no quadrante *se* e *id* que nos restantes e ao mesmo tempo o número de "1s" nos quadrantes *sd* e *ie* estiver compreendido em determinado intervalo de valores, tal que:

$$sd < se \ \wedge \quad ie < id \quad \wedge \quad id > 7 \quad \wedge \quad se > 7 \tag{23}$$

- Critério "KO!" – é o Quadrado de rejeição se existir um número consideravelmente maior de "1s" no quadrante *ie* e *sd* que nos restantes e ao mesmo tempo o somatório de "1s" desses quadrantes estiver compreendido em determinado intervalo de valores, tal que:

$$(sd \leq se \ \wedge \ \ sd \leq 8) \ \vee \ (ie \leq id \ \wedge \ \ ie \leq 8) \vee \ (se \leq sd \ \wedge \ \ id \leq ie) \quad (24)$$

- Critério "?" – quando o valor de ruído admitido é superado coloca a peça numa zona de incerteza.

Apenas em Time Driven – os critérios acima referidos só se aplicam para dimensões reais

Método 2: Análise da ordem das transições

De uma forma geral este algoritmo identifica a forma da peça pela ordem pela qual se dá a sequência de transições esperadas em determinadas zonas da peça. Verificar se ocorre a detecção do furo inferior esquerdo primeiro que o furo superior direito e se ocorrem ambos.

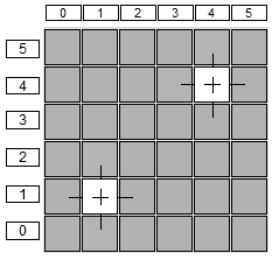


Figura A.30. Identificação por sequência de transições

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – a transição negativa a linha 1 ocorre primeiro que a da linha 4 e a transição positiva da linha 1 ocorre antes da coluna 2 e a transição positiva da linha 4 ocorre depois da coluna 4.
- Critério "KO!" – quando a ordem é invertida ou apenas detecta uma das transições.
- Critério "?" – quando os critérios anteriores não são preenchidos.

Apenas é passível de ser aplicado em Time driven e data driven. Tem por pontos forte ser um programa insensível ao ruído fora destas duas linhas e duas colunas, e como pontos fracos o facto de se a zona de furo estiver descentrada o algoritmo terá um mau desempenho e a sua sensibilidade ao ruído nas linhas e colunas analisadas.

Método 3: Análise por String Matching

De uma forma geral este algoritmo identifica a forma da peça por comparação da sequência de leituras adquiridas pelo varrimento de sensores (colocada na matriz/espaço de amostragem), com a imagem binária pretendida e previamente armazenada em memória. Começa por procurar

a palavra inicial t_0' no instante $t = t_0$, e sempre que tal acontece, verifica se a sequência pretendida é conseguida. Se sim, então a forma foi identificada, se não repete o processo para t = t+1.

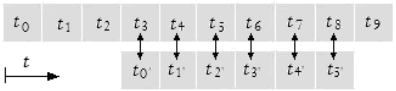


Figura A.31. String matching

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – a sequência do quadrado desejado encontrada dentro da tolerância definida (ruído equivalente a uma letra diferente por palavra).
  Matriz_d[63, 60, 63, 63, 15, 63] ou [111111.111100.111111.111111.001111.111111]

- Critério "KO!" - sequência de palavras que descrevem o quadrado de rejeição encontrada dentro da tolerância definida (uma letra diferente por palavra).
  Matriz_e[63, 47, 63, 63, 61, 63] ou [111111.001111111111.111111.111100.111111]

- Critério "?" - Sequencia encontrada com tolerância de duas ou mais letras por palavra

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real. Tem por pontos forte ser um programa pequeno (por isso rápido) e como pontos fracos a possível ocorrência de erros na introdução dos dados iniciais e a sua sensibilidade a zonas de ruído concentrado.

7.1.4. Caso 7 – Identificação da correcta orientação de três orifícios numa peça quadrangular

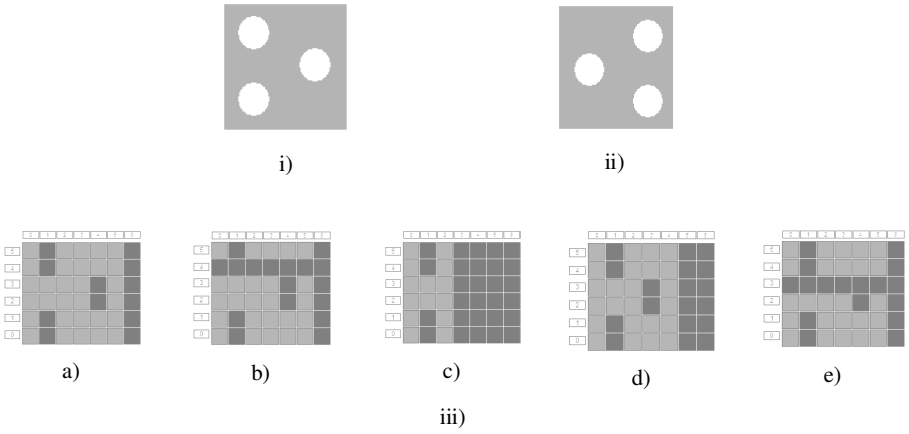


Figura A.32. Peça triangular com três orifícios: i) forma pretendida, ii) forma de rejeição, iii) modelos de ruído testados.

Na figura 7.32. podem observar-se as formas pretendida e de rejeição em i) e ii) respectivamente e os modelos de ruído, considerados para o desenvolvimento e teste dos algoritmos em iii). Estes começam por representar a imagem ideal da forma pretendida no caso a), mas considerando o caso de uma avaria permanente de um dos sensores podemos ter algo semelhante ao caso exposto em b), por outro lado, no caso c) procura-se simular a falha temporária e de curta duração do conjunto de todos os sensores. Já no caso e) procura-se testar a sensibilidade e robustez dos algoritmos para o aparecimento de ruído pela falha de um outro sensor, que tem implicações diferentes do anterior. Por último procura-se reproduzir, no caso e), a resposta esperada da aquisição em *event driven*.

Método 1: Análise por regiões sextantes

Este algoritmo pretende analisar a imagem binária a partir da partição ortogonal em nove regiões iguais, que designaremos de sextantes e do número de "1s" e "0s" neles contidos.

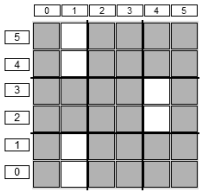


Figura A.33. Identificação por regiões (sextantes)

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" **-** é a forma pretendido se existir um número de "1s" nos sextantes *se*, *ie* e *md* tal que:

$$(se \wedge ie \wedge md) \geq 1 \quad \wedge \quad (se \wedge ie \wedge md) \leq 3 \tag{25}$$

- Critério "KO!" **-** é a forma pretendido se existir um número de "1s" nos sextantes *sd*, *id* e *me* tal que:

$$(sd \wedge id \wedge me) \geq 1 \quad \wedge \quad (sd \wedge id \wedge me) \leq 3 \tag{26}$$

- Critério "?" – quando não é possível assegurar o valor de ruído admitido e definidos nos critérios "OK!" e "KO!", colocando a peça numa zona de incerteza.

Tipo de aquisição de dados apenas aplicável em Time driven - com velocidade de tapete constante e frequência de amostragem para a representação da peça dentro do espaço de amostragem.

O principal ponto forte deste algoritmo é o facto de este ser insensível fora das regiões visadas. Já o seu ponto fraco é poder ser muito prejudicado se ocorrer uma "descentragem" da peça e não puder ser aplicado em data driven.

Método 2: Análise da ordem das transições

De uma forma geral este algoritmo identifica a forma da peça pela ordem pela qual se dá a sequência de transições esperadas em determinada zona da peça. Verificar se ocorre a detecção dos furos inferior e superior esquerdo simultaneamente e primeiro que o furo direito e se todos ocorrem.

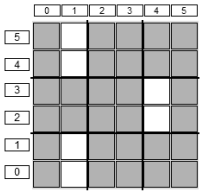


Figura A.34. Identificação por sequência de transições

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – transições ascendentes nas linhas 1 e 4 ocorrem ao mesmo tempo e primeiro que as das linhas 2 e 3.
- Critério "KO!" – transições ascendentes nas linhas 1 e 4 ocorrem ao mesmo tempo e depois que as das linhas 2 e 3.
- Critério "?" – quando o ruído interfere e nenhuma das condições anteriores se verificar.

É passível de ser aplicado tanto em Time driven como em data driven visto ser a sequência que está a ser testada e não valores absolutos relativos a uma imagem "real" da peça. Tem por pontos forte poder ser aplicado em data driven, e como ponto fracos o facto ser sensível ao ruído nas linhas analisadas (todas).

Método 3: Análise por String Matching

De uma forma geral este algoritmo identifica a forma da peça por comparação da sequência de leituras adquiridas pelo varrimento de sensores (colocada na matriz/espaço de amostragem), com a imagem binária pretendida e previamente armazenada em memória. Começa por procurar a palavra inicial t_0' no instante $t = t_0$, e sempre que tal acontece, verifica se a sequência

pretendida é conseguida. Se sim, então a forma foi identificada, se não repete o processo para $t = t+1$.

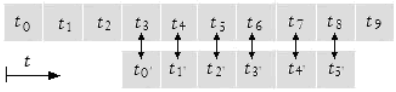


Figura A.35. Identificação por String matching

Os critérios considerados na análise do algoritmo são:

- Critério "OK!" – a sequência do quadrado desejado encontrada dentro da tolerância definida (ruído equivalente a uma letra diferente por palavra).
  Matriz_d[631263635163] ou [111111.101101.111111.111111.110011.111111]

- Critério "KO!" - sequência de palavras que descrevem o quadrado de rejeição encontrada dentro da tolerância definida (uma letra diferente por palavra).
  Matriz_e[63, 51, 63, 63, 12, 63] ou [111111.110011.111111.111111.101101.111111]

  - Critério "?" - Sequência encontrada com tolerância de duas ou mais letras por palavra.

Apenas é passível de ser aplicado em Time driven uma vez que a peça é previamente "desenha" em memória, admitindo a sua imagem real.

Tem por pontos forte ser um programa pequeno (por isso rápido) e como pontos fracos a possível ocorrência de erros na introdução dos dados iniciais e a sua sensibilidade a zonas de ruído concentrado, cujo tamanho e posicionamento dos furos relativamente aos sensores, poderá favorecer.

7.1.5. Caso 8 – Identificação da correcta orientação de uma marca triangular

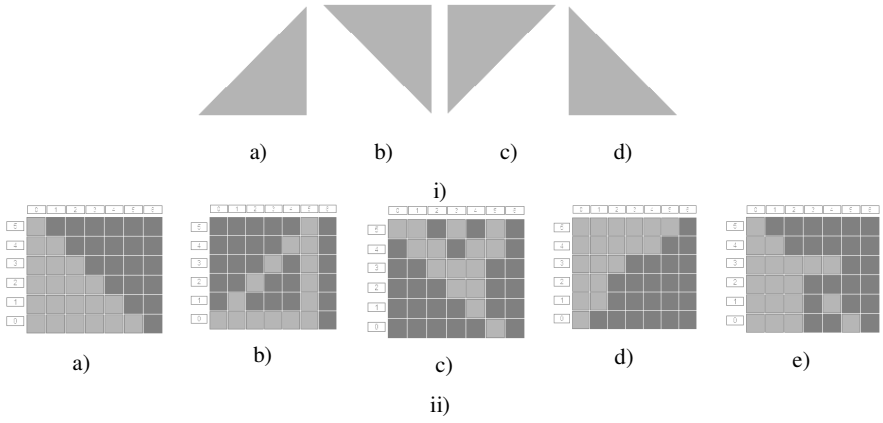


Figura A.36. i) Orientações de marca triangular, ii) modelos de ruído testados

Método 1: Análise por regiões quadrantes

Este algoritmo pretende analisar a imagem binária a partir da partição ortogonal em quatro regiões iguais, que designaremos de quadrantes e do número de "1s" e "0s" neles contidos. Os critérios de definição da orientação são respectivamente:

$$\text{a) } se < (sd \wedge ie \wedge id) \tag{27}$$

$$\text{b) } ie < (se \wedge sd \wedge id) \tag{28}$$

$$\text{c) } id < (se \wedge ie \wedge sd) \tag{29}$$

$$\text{d) } sd < (se \wedge ie \wedge id) \tag{30}$$

Método 2: Análise da ordem das transições

De uma forma geral este algoritmo identifica a forma da peça pela ordem pela qual se dá a sequência de transições esperadas

7.1.6. Caso 9 – Identificação da correcta orientação de uma marca quadrangular

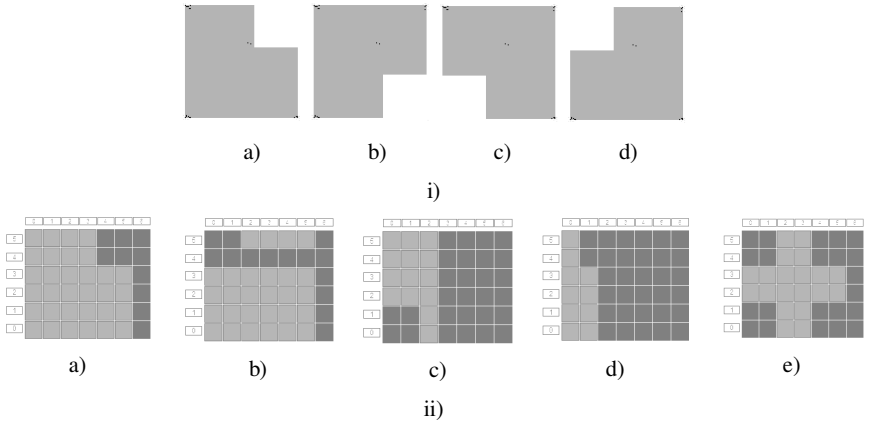


Figura A.37. i) Orientações de uma marca quadrangular ii) modelos de ruído testados

Método 1: Análise por regiões quadrantes

Este algoritmo pretende analisar a imagem binária a partir da partição ortogonal em quatro regiões iguais, que designaremos de quadrantes e do número de "1s" e "0s" neles contidos. Os critérios de definição da orientação são respectivamente:

$$\text{a) } se < (sd \wedge ie \wedge id) \tag{27}$$

b) $ie < (se \wedge sd \wedge id)$ (28)

c) $id < (se \wedge ie \wedge sd)$ (29)

d) $sd < (se \wedge ie \wedge id)$ (30)

Método 2: Análise da ordem das transições

De uma forma geral este algoritmo identifica a forma da peça pela ordem pela qual se dá a sequência de transições esperadas em determinada zona da peça.

7.1.7. Caso 10 – Identificação da correcta orientação da reentrância de uma marca quadrangular

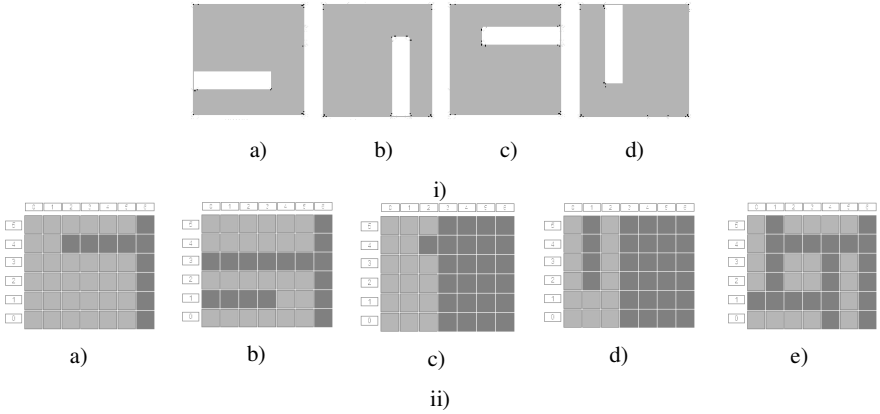


Figura A.38. i) Orientações da marca quadrangular com reentrâncias, ii) modelos de ruído testados

Método 1: Análise por regiões quadrantes

Este algoritmo pretende analisar a imagem binária a partir da partição ortogonal em quatro regiões iguais, que designaremos de quadrantes e do número de "1s" e "0s" neles contidos. Os critérios de definição da orientação são respectivamente:

a) $se < (sd \wedge ie \wedge id)$ (27)

b) $ie < (se \wedge sd \wedge id)$ (28)

c) $id < (se \wedge ie \wedge sd)$ (29)

d) $sd < (se \wedge ie \wedge id)$ (30)

Método 2: Análise da ordem das transições

De uma forma geral este algoritmo identifica a forma da peça pela posição em que se dá determinada transição e pela ordem por que se dão as transições. Verifica as transições ascendentes (*ph*) na linha 1, as transições descendentes (*nh*) na linha 4, as positivas horizontais (*pv*) na coluna 4 e as negativas horizontais (*nv*) na coluna 1.

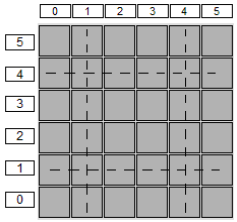


Figura A.39. Orientação de peça quadrangular com reentrância

a) se $nh = 2$ e $pv = 5$

b) se $nh = 1$ e $nv = 2$

c) se $ph = 4$ e $nv = 1$

d) se $ph = 5$ e $pv = 4$

Anexo 3

Programação de Algoritmos

```
0001 FUNCTION Comparados : INT
0002 VAR_INPUT
0003 END_VAR
0004 VAR
0005     K1, K2, K3, K4, K5:INT;
0006     a: INT;
0007     b: INT;
0008 END_VAR
0001
0002 (* _____ Algoritmo 2 - Regiões Comparadas _____*)
0003
0004
0005 (* ---------- Caso 1 - Triângulo ----------*)
0006
0007 IF bt_caso1 THEN              (* Botão "Caso 1" foi premido *)
0008
0009 k1 := Contagem_coluna (coluna := 0, in_1:=1, In_2:=5) + Contagem_coluna (coluna := 1, in_1:=2, In_2:=4) +
0010 Contagem_coluna (coluna := 2, in_1:=3, In_2:=3);
0011 k2 := Contagem_coluna (coluna := 5, in_1:=1, In_2:=5) + Contagem_coluna (coluna := 4, in_1:=2, In_2:=4) +
0012 Contagem_coluna (coluna := 3, in_1:=3, In_2:=3);
0013
0014     K1:= k1+1;
0015     K2:= K2 +1;
0016
0017     IF k1/k2 > 2.5 THEN
0018         Comparados := 1;
0019     ELSE
0020         IF K2/K1 > 2.5 THEN
0021             Comparados := 0;
0022         ELSE
0023             IF (K1/K2 <= 2.5) AND (K2/K1 <= 2.5) THEN
0024                 algor_2_aux := 1;
0025             END_IF
0026         END_IF
0027     END_IF
0028
0029 END_IF
0030
0031 (* ---------- Caso 2 - Quadrado Furado ----------*)
0032
0033 IF bt_caso2 THEN              (* Botão "Caso 2" foi premido *)
0034
0035     k1 := Contagem_coluna (coluna := 2, in_1:=2, In_2:=3) + Contagem_coluna (coluna := 3, in_1:=2, In_2:=3);
0036
0037         IF k1 <=2 THEN
0038             Comparados := 1;
0039         ELSE
0040             IF k1 > 3 THEN
0041             Comparados := 0;
0042             ELSE
0043                 IF k1 = 3 THEN
0044                     algor_2_aux := 1;
0045                 END_IF
0046             END_IF
0047         END_IF
0048
0049 END_IF
0050
0051 (* ---------- Caso 3 - Quadrado com reentrância ----------*)
0052 (* não aplicável *)
0053
0054
0055 (* ---------- Caso 4 - Triângulo furado ----------*)
0056
```

Figura A.40. Algoritmo Sobreposição – parte 1

```
0057 IF bt_caso4 THEN              (* Botão "Caso 4" foi premido *)
0058
0059 K3 := Contagem_coluna (coluna := 1, in_1:=2, In_2:=3) + Contagem_coluna (coluna := 2, in_1:=1, In_2:=3) +
0060  Contagem_coluna (coluna := 3, in_1:=1, In_2:=2);
0061
0062     IF k3 >=5 THEN
0063         Comparados := 1;
0064     ELSE
0065         IF k3 <=2THEN
0066             Comparados := 0;
0067         ELSE
0068             IF k3 = 3 OR k3 = 4 THEN
0069                 algor_2_aux := 1;
0070             END_IF
0071         END_IF
0072     END_IF
0073
0074 END_IF
0075
0076 (* ---------------------------------------------------- Caso 5 - Rectângulos ---------------------------------------------------- *)
0077
0078 IF bt_caso5 THEN              (* Botão "Caso 5" foi premido *)
0079
0080 (* o número de linhas é o valor de referências - compara peças com comprimentos diferentes mas com "alturas" iguais *)
0081     k1 := 0;
0082     IF amostras_coluna > amostras_linha THEN
0083         FOR a := amostras_linha TO amostras_coluna DO
0084             k1 := k1 + Contagem_coluna (coluna := a, in_1:=0, In_2:=amostras_linha);
0085         END_FOR
0086     END_IF
0087
0088     k2 := 0;
0089     IF amostras_coluna < amostras_linha THEN
0090         FOR b := amostras_coluna TO amostras_linha DO
0091             k2 := k2 + Contagem_coluna (coluna := b, in_1:=0, In_2:=amostras_linha);
0092         END_FOR
0093     END_IF
0094
0095     IF k1 = 0 AND k2 = 0 THEN
0096         Comparados := 1;
0097     ELSE
0098         IF K2 > amostras_linha/2 OR k1 > amostras_linha/2 THEN
0099             Comparados := 0;
0100         ELSE
0101             algor_2_aux := 1;
0102         END_IF
0103     END_IF
0104
0105 END_IF
0106
0107 (* ---------------------------------------------------- Caso 6 - Quadrado de dois furos---------------------------------------- *)
0108
0109 (* não aplicável - equivalente (com maior ou menos precisão) ao algoritmo do quadrante já implementado *)
0110
0111
0112 (* ---------------------------------------------------- Caso 7 - Quadrado de três furos---------------------------------------- *)
0113
0114 (* não aplicável - equivalente (com maior ou menor precisão) ao algoritmo do sextante já implementado*)
0115
```

Figura A.41. Algoritmo Sobreposição – parte 2

```
0001 FUNCTION Contagem_coluna : INT
0002
0003 (*   Retorna o número de "1's" existentes na:  .. *)
0004
0005 VAR_INPUT
0006     coluna : INT;              (* ...coluna "coluna" entre os valores "in_1" e "in_2", inclusive *)
0007     in_1: INT;
0008     in_2 : INT;
0009 END_VAR
0010
0011 VAR
0012     maximo, minimo : INT;   (* valores minimos e máximos da busca *)
0013     mascara : WORD;            (* Registo temporário *)
0014     i: INT;                    (* Contador Interno *)
0015 END_VAR
```

```
0001
0002
0003 minimo := MIN (in_1, In_2);      (* Define valor mais baixo *)
0004 maximo := MAX (in_1, in_2);      (* Define valor mais alto *)
0005
0006 IF coluna > 15 OR coluna < 0 OR minimo <0 OR Maximo > 5 THEN
0007     Contagem_coluna := -1;       (* se parâmetros de entrada são inválidos *)
0008     RETURN;
0009 END_IF
0010
0011 Contagem_coluna := 0;              (* Totalizador a zero *)
0012 FOR i := Minimo TO Maximo DO
0013     Mascara := SHL(WORD#1,i);      (* Percorre linhas de baixo para cima *)
0014     IF (Matriz_b[coluna] AND mascara) > 0 THEN
0015         Contagem_coluna:= Contagem_coluna +1;
0016     END_IF
0017 END_FOR
0018
0019
0020
0021
0022
```

Figura A.42. Função Contagem_coluna

```
0001 FUNCTION Quadrantes : INT
0002 VAR_INPUT
0003 END_VAR
0004 VAR
0005     k2: INT; k1: INT;
0006     i: INT; j: INT;
0007     Contagem_zona_ie: INT; Contagem_zona_id: INT; Contagem_zona_se: INT; Contagem_zona_sd: INT;
0008     coluna_mid: INT; linha_mid: INT;
0009     Mascara: WORD;
0010
0011     Contagem: BOOL;
0012     minimo: INT;
0013     alg_1_ok: BOOL;
0014     alg_1_ko: BOOL;
0015     alg_1_d: BOOL;
0016 END_VAR
0001
0002 (* ___________ Algoritmo 1 - Quadrantes ___________*)
0003
0004 (*------------------------------------------ Corpo geral do algoritmo ------------------------------------------*)
0005
0006 (* Calculo prévio de alguns valores necessários. Número de "1s" em cada um dos Quadrantes *)
0007
0008 coluna_mid := amostras_coluna/2;                    (* determinação da coluna intermédia da forma *)
0009 linha_mid := amostras_linha/2;                      (* determinação da linha intermédia da forma *)
0010
0011 Contagem_zona_ie := 0;                              (* Inicia o contador associado à região a zero *)
0012 FOR i := 0 TO coluna_mid DO                         (*percorre as colunas desde a zero até à coluna do meio*)
0013     FOR j := 0 TO linha_mid DO                          (* Percorre linhas de baixo para cima até à linha do meio*)
0014         Mascara := SHL(WORD#1,j);
0015         IF (Matriz_b[i] AND mascara) > 0 THEN          (* sempre que detecta algum "1" à medida que percorre a região definida*)
0016             Contagem_zona_ie:= Contagem_zona_ie +1;  (* incrementa o contador*)
0017         END_IF
0018     END_FOR
0019 END_FOR
0020
0021 Contagem_zona_id := 0;                              (* Totalizador a zero *)
0022 FOR i := coluna_mid +1 TO amostras_coluna DO
0023     FOR j := 0 TO linha_mid DO
0024         Mascara := SHL(WORD#1,j);                      (* Percorre linhas de baixo para cima *)
0025         IF (Matriz_b[i] AND mascara) > 0 THEN
0026             Contagem_zona_id:= Contagem_zona_id +1;
0027         END_IF
0028     END_FOR
0029 END_FOR
0030
0031 Contagem_zona_se := 0;                                 (* Totalizador a zero *)
0032 FOR i := 0 TO coluna_mid DO
0033     FOR j := linha_mid +1 TO amostras_linha DO
0034         Mascara := SHL(WORD#1,j);                      (* Percorre linhas de baixo para cima *)
0035         IF (Matriz_b[i] AND mascara) > 0 THEN
0036             Contagem_zona_se:= Contagem_zona_se +1;
0037         END_IF
0038     END_FOR
0039 END_FOR
0040
0041 Contagem_zona_sd := 0;                                 (* Totalizador a zero *)
0042 FOR i := coluna_mid +1 TO amostras_coluna DO
0043     FOR j := linha_mid +1 TO amostras_linha DO
0044         Mascara := SHL(WORD#1,j);                      (* Percorre linhas de baixo para cima *)
0045         IF (Matriz_b[i] AND mascara) > 0 THEN
0046             Contagem_zona_sd:= Contagem_zona_sd +1;
0047         END_IF
0048     END_FOR
```

Figura A.43. Algoritmo Quadrantes – parte 1

```
0049 END_FOR
0050
0051 (*_____ Problemas de Identificação____*)
0052
0053 (* ---------------------------------------------- Caso 1 - Triângulo ---------------------------------------------- *)
0054
0055 IF bt_caso1 THEN              (* Botão "Caso 1" foi premido *)
0056
0057 (* Critério "OK!" - se existir um número consideravelmente maior de "1s" no quadrante ie que no sd*)
0058 (* e ao mesmo tempo o numero de "1s" nos quadrantes se e id for aproximadamente igual*)
0059     IF ((Contagem_zona_ie + 1)/(Contagem_zona_sd +1)) >= 3 AND ((Contagem_zona_id + 1)/(Contagem_zona_se +1)) >= 0.75
0060     AND ((Contagem_zona_id + 1)/(Contagem_zona_se +1)) <= 1.5 THEN
0061         Quadrantes := 1;
0062     ELSE
0063         (* Critério "KO!" - se existir um número consideravelmente maior de uns no quadrante id que no se *)
0064         (*e ao mesmo tempo o numero de uns nos quadrantes sd e ie for aproximadamente igual*)
0065         IF ((Contagem_zona_id + 1)/(Contagem_zona_se +1)) >= 3 AND ((Contagem_zona_ie + 1)/(Contagem_zona_sd +1)) >= 0.75
0066         AND ((Contagem_zona_ie + 1)/(Contagem_zona_sd +1)) <= 1.5 THEN
0067             Quadrantes := 0;
0068         ELSE
0069             (* Critério "?" - quando não é possível assegurar qual o triângulo em presença. *)
0070             (*O valor de ruído admitido é superado, o que coloca a peça numa zona de incerteza*)
0071             IF ((Contagem_zona_ie + 1)/(Contagem_zona_sd +1)) < 3 AND ((Contagem_zona_id + 1)/(Contagem_zona_se +1)) < 3 THEN
0072                 alg_1_aux := 1;
0073             END_IF
0074         END_IF
0075     END_IF
0076
0077 END_IF
0078
0079 (* ---------------------------------------------- Caso 2 - Quadrado furado ---------------- não aplicável ----------*)
0080
0081 (*não é aplicadol o sextante - por ser coincidir com a zona analisada pelo algoritmo das regiões comparadas*)
0082
0083
0084 (* ---------------------------------------------- Caso 3 - Quadrante com Reentrancias ---------------------------------------------- *)
0085
0086 IF bt_caso3 THEN              (* Botão "Caso 3" foi premido *)
0087
0088 (*Critério "OK!" - se existir um número consideravelmente maior de "1s" nos quadrantes ie e sd *)
0089 (* que nos quadrantes se e id, sendo cada um destes aproximadamente iguais*)
0090     IF Contagem_zona_ie > (Contagem_zona_se) AND (Contagem_zona_id) < Contagem_zona_sd AND
0091     ((Contagem_zona_id)/(Contagem_zona_se +1)) <= 1.5 AND ((Contagem_zona_se)/(Contagem_zona_id +1)) <= 1.5 THEN
0092         Quadrantes := 1;
0093     ELSE
0094         (* Critério "KO!" - se existir um número consideravelmente maior de uns no quadrante id que no se *)
0095         (* e ao mesmo tempo o numero de uns nos quadrantes sd e ie for aproximadamente igual*)
0096         IF Contagem_zona_se > (Contagem_zona_sd) AND Contagem_zona_id > (Contagem_zona_ie) AND
0097         ((Contagem_zona_id)/(Contagem_zona_se +1)) <= 1.5 AND ((Contagem_zona_se)/(Contagem_zona_id +1)) <= 1.5 THEN
0098             Quadrantes := 0;
0099         ELSE
0100             (* Critério "?" - quando não é possível assegurar qual o triângulo em presença. *)
0101             (*O valor de ruído admitido é superado, o que coloca a peça numa zona de incerteza*)
0102                 alg_1_aux := 1;
0103         END_IF
0104     END_IF
0105
0106 END_IF
0107
0108 (* ---------------------------------------------- Caso 4 - Triângulo Furado ---------------------------------------------- *)
0109 (* não aplicável - aplicável o algoritmo dos sextantes *)
0110
0111 (* ---------------------------------------------- Caso 5 - Rectangulos ----------------------------------------------*)
0112
```

Figura A.44. Algoritmo Quadrantes – parte 2

```
0113 IF bt_caso5 THEN              (* Botão "Caso 5" foi premido *)
0114
0115 (*considerando uma altura constante - poderia-se clcular recorrendo ao numero de "1s" dentro da área *)
0116     IF amostras_coluna = amostras_linha THEN
0117         Quadrantes := 1;
0118     ELSE
0119         IF amostras_coluna > (amostras_linha +1) OR amostras_linha > (amostras_coluna +1) THEN
0120             Quadrantes := 0;
0121         ELSE
0122             alg_1_aux := 1;
0123         END_IF
0124     END_IF
0125
0126 END_IF
0127
0128 (* ------------------------------------------- Caso 6 - Quadrado de dois Furos ------------------------------------------*)
0129
0130 IF bt_caso6 THEN              (* Botão "Caso 6" foi premido *)
0131
0132 (* Critério "OK!" - se existir um número consideravelmente maior de "1s" no quadrante ie que no sd *)
0133 (*e ao mesmo tempo o numero de "1s" nos quadrantes se e id for aproximadamente igual*)
0134     IF Contagem_zona_sd < Contagem_zona_se AND Contagem_zona_ie < Contagem_zona_id AND
0135     Contagem_zona_id > 7 AND Contagem_zona_se > 7 THEN
0136         Quadrantes := 1;
0137     ELSE
0138         (* Critério "KO!" - se existir um número consideravelmente maior de uns no quadrante id que no se *)
0139         (*e ao mesmo tempo o numero de uns nos quadrantes sd e ie for aproximadamente igual*)
0140         IF (Contagem_zona_sd <= Contagem_zona_se AND Contagem_zona_sd <= 8) OR
0141 (Contagem_zona_ie <= Contagem_zona_id  AND Contagem_zona_ie <= 8) OR (Contagem_zona_se <= Contagem_zona_sd
0142 AND Contagem_zona_id <= Contagem_zona_ie) THEN
0143             Quadrantes := 0;
0144         ELSE
0145             (* Critério "?" - quando não é possível assegurar qual o triângulo em presença. *)
0146             (* O valor de ruído admitido é superado, o que coloca a peça numa zona de incerteza*)
0147             alg_1_aux := 1;
0148         END_IF
0149     END_IF
0150
0151 END_IF
0152
0153 (* ------------------------------------------- Caso 7 - Quadrado de três Furos ------------------------------------------- *)
0154 (* não aplicável - preferível o algoritmo do sextantes *)
0155
0156
0157 (*_____ Problemas de Orientação_____*)
0158
0159
0160 (* ------------------------------------------- Caso 8 - Orientação de triangulos ------------------------------------------*)
0161
0162 IF bt_caso8 THEN              (* Botão "Caso 8" foi premido *)
0163
0164     IF Contagem_zona_sd < Contagem_zona_se AND Contagem_zona_sd < Contagem_zona_ie AND
0165     Contagem_zona_sd < Contagem_zona_id THEN
0166         sd_tri := TRUE;
0167     ELSE
0168         sd_tri := FALSE;
0169     END_IF
0170
0171     IF Contagem_zona_se < Contagem_zona_sd AND Contagem_zona_se < Contagem_zona_ie AND
0172     Contagem_zona_se < Contagem_zona_id THEN
0173         se_tri := TRUE;
0174     ELSE
0175         se_tri := FALSE;
0176     END_IF
```

Figura A.45. Algoritmo Quadrantes – parte 3

```
0177
0178      IF Contagem_zona_id < Contagem_zona_se AND Contagem_zona_id < Contagem_zona_ie AND
0179      Contagem_zona_id < Contagem_zona_sd THEN
0180          id_tri := TRUE;
0181      ELSE
0182          id_tri := FALSE;
0183      END_IF
0184
0185      IF Contagem_zona_ie < Contagem_zona_se AND Contagem_zona_ie < Contagem_zona_sd AND
0186      Contagem_zona_ie < Contagem_zona_id THEN
0187          ie_tri := TRUE;
0188      ELSE
0189          ie_tri := FALSE;
0190      END_IF
0191  ELSE
0192      ie_tri := FALSE; id_tri := FALSE; se_tri := FALSE; sd_tri := FALSE;
0193  END_IF
0194
0195
0196  (* ---------------------------------------------- Caso 8 - Caso de orientação dos Quadrados sem cantos ----------------------------------------*)
0197
0198  IF bt_caso9 THEN              (* Botão "Caso 9" foi premido *)
0199  alg_1_ok := FALSE; alg_1_ko := FALSE; alg_1_d := FALSE;
0200      IF Contagem_zona_sd < Contagem_zona_se AND Contagem_zona_sd < Contagem_zona_ie AND
0201      Contagem_zona_sd < Contagem_zona_id THEN
0202          sd_quad := TRUE;
0203      ELSE
0204          sd_quad := FALSE;
0205      END_IF
0206
0207      IF Contagem_zona_se < Contagem_zona_sd AND Contagem_zona_se < Contagem_zona_ie AND
0208      Contagem_zona_se < Contagem_zona_id THEN
0209          se_quad := TRUE;
0210      ELSE
0211          se_quad := FALSE;
0212      END_IF
0213
0214      IF Contagem_zona_id < Contagem_zona_se AND Contagem_zona_id < Contagem_zona_ie AND
0215      Contagem_zona_id < Contagem_zona_sd THEN
0216          id_quad := TRUE;
0217      ELSE
0218          id_quad := FALSE;
0219      END_IF
0220
0221      IF Contagem_zona_ie < Contagem_zona_se AND Contagem_zona_ie < Contagem_zona_sd AND
0222      Contagem_zona_ie < Contagem_zona_id THEN
0223          ie_quad := TRUE;
0224      ELSE
0225          ie_quad := FALSE;
0226      END_IF
0227  ELSE
0228      ie_quad := FALSE; id_quad := FALSE; se_quad := FALSE; sd_quad := FALSE;
0229  END_IF
0230
0231
0232  (* ---------------------------------------------- Caso 8 - Caso de orientação dos Quadrados com reentrâncias ----------------------------------*)
0233
0234  IF bt_caso10 THEN             (* Botão "Caso 10" foi premido *)
0235
0236      IF Contagem_zona_sd < Contagem_zona_se AND Contagem_zona_sd < Contagem_zona_ie AND
0237      Contagem_zona_sd < Contagem_zona_id THEN
0238          sd_reent := TRUE;
0239      ELSE
0240          sd_reent := FALSE;
```

Figura A.46. Algoritmo Quadrantes – parte 4

```
0241    END_IF
0242
0243    IF Contagem_zona_se < Contagem_zona_sd AND Contagem_zona_se < Contagem_zona_ie AND
0244    Contagem_zona_se < Contagem_zona_id THEN
0245        se_reent := TRUE;
0246    ELSE
0247        se_reent := FALSE;
0248    END_IF
0249
0250    IF Contagem_zona_id < Contagem_zona_se AND Contagem_zona_id < Contagem_zona_ie AND
0251    Contagem_zona_id < Contagem_zona_sd THEN
0252        id_reent := TRUE;
0253    ELSE
0254        id_reent := FALSE;
0255    END_IF
0256
0257    IF Contagem_zona_ie < Contagem_zona_se AND Contagem_zona_ie < Contagem_zona_sd AND
0258    Contagem_zona_ie < Contagem_zona_id THEN
0259        ie_reent := TRUE;
0260    ELSE
0261        ie_reent := FALSE;
0262    END_IF
0263 ELSE
0264    ie_reent := FALSE; id_reent := FALSE; se_reent := FALSE; sd_reent := FALSE;
0265 END_IF
```

Figura A.47. Algoritmo Quadrantes – parte 5

```
0001    FUNCTION sextante : INT
0002 VAR_INPUT
0003 END_VAR
0004 VAR
0005    mascara: WORD;
0006    i: INT;
0007    j: INT;
0008
0009    Contagem_zona_ie: INT; Contagem_zona_im: INT; Contagem_zona_id: INT;
0010    Contagem_zona_se: INT; Contagem_zona_sm: INT; Contagem_zona_sd: INT;
0011    Contagem_zona_me: INT; Contagem_zona_mm: INT; Contagem_zona_md: INT;
0012
0013 END_VAR
0001
0002 (* _____ Algoritmo 7 - Sextantes _____*)
0003
0004 (*------------------------------------------- Corpo geral do algoritmo -------------------------------------------*)
0005
0006 (* Calculo prévio de alguns valores necessários. Número de "1s" em cada um dos Quadrantes *)
0007
0008 (* Primeiro caso: contagem de 1's *)
0009
0010 Contagem_zona_ie := 0;                          (* Totalizador a zero *)
0011 FOR i := 0 TO 1 DO
0012    FOR j := 0 TO 1 DO
0013       mascara := SHL(WORD#1,j);                  (* Percorre linhas de baixo para cima *)
0014       IF (Matriz_b[i] AND mascara) > 0 THEN
0015          Contagem_zona_ie:= Contagem_zona_ie +1;
0016       END_IF
0017    END_FOR
0018 END_FOR
0019
0020 Contagem_zona_im := 0;                          (* Totalizador a zero *)
0021 FOR i := 2 TO 3 DO
0022    FOR j := 0 TO 1 DO
0023       mascara := SHL(WORD#1,j);                  (* Percorre linhas de baixo para cima *)
0024       IF (Matriz_b[i] AND mascara) > 0 THEN
0025          Contagem_zona_im:= Contagem_zona_im +1;
0026       END_IF
0027    END_FOR
0028 END_FOR
0029
0030 Contagem_zona_id := 0;                          (* Totalizador a zero *)
0031 FOR i := 4 TO 5 DO
0032    FOR j := 0 TO 1 DO
0033       mascara := SHL(WORD#1,j);                  (* Percorre linhas de baixo para cima *)
0034       IF (Matriz_b[i] AND mascara) > 0 THEN
0035          Contagem_zona_id:= Contagem_zona_id +1;
0036       END_IF
0037    END_FOR
0038 END_FOR
0039
0040 Contagem_zona_se := 0;                          (* Totalizador a zero *)
0041 FOR i := 0 TO 1 DO
0042    FOR j := 4 TO 5 DO
0043       mascara := SHL(WORD#1,j);                  (* Percorre linhas de baixo para cima *)
0044       IF (Matriz_b[i] AND mascara) > 0 THEN
0045          Contagem_zona_se:= Contagem_zona_se +1;
0046       END_IF
0047    END_FOR
0048 END_FOR
0049
0050 Contagem_zona_sm := 0;                          (* Totalizador a zero *)
0051 FOR i := 2 TO 3 DO
```

Figura A.48. Algoritmo Sextantes – parte 1

```
0052     FOR j := 4 TO 5 DO
0053         mascara := SHL(WORD#1,j);                    (* Percorre linhas de baixo para cima *)
0054         IF (Matriz_b[i] AND mascara) > 0 THEN
0055             Contagem_zona_sm:= Contagem_zona_sm +1;
0056         END_IF
0057     END_FOR
0058 END_FOR
0059
0060 Contagem_zona_sd := 0;                               (* Totalizador a zero *)
0061 FOR i := 4 TO 5 DO
0062     FOR j := 4 TO 5 DO
0063         mascara := SHL(WORD#1,j);                    (* Percorre linhas de baixo para cima *)
0064         IF (Matriz_b[i] AND mascara) > 0 THEN
0065             Contagem_zona_sd:= Contagem_zona_sd +1;
0066         END_IF
0067     END_FOR
0068 END_FOR
0069
0070 Contagem_zona_me := 0;                               (* Totalizador a zero *)
0071 FOR i := 0 TO 1 DO
0072     FOR j := 2 TO 3 DO
0073         mascara := SHL(WORD#1,j);                    (* Percorre linhas de baixo para cima *)
0074         IF (Matriz_b[i] AND mascara) > 0 THEN
0075             Contagem_zona_me:= Contagem_zona_me +1;
0076         END_IF
0077     END_FOR
0078 END_FOR
0079
0080 Contagem_zona_mm := 0;                               (* Totalizador a zero *)
0081 FOR i := 2 TO 3 DO
0082     FOR j := 2 TO 3 DO
0083         mascara := SHL(WORD#1,j);                    (* Percorre linhas de baixo para cima *)
0084         IF (Matriz_b[i] AND mascara) > 0 THEN
0085             Contagem_zona_mm:= Contagem_zona_mm +1;
0086         END_IF
0087     END_FOR
0088 END_FOR
0089
0090 Contagem_zona_md := 0;                               (* Totalizador a zero *)
0091 FOR i := 4 TO 5 DO
0092     FOR j := 2 TO 3 DO
0093         mascara := SHL(WORD#1,j);                    (* Percorre linhas de baixo para cima *)
0094         IF (Matriz_b[i] AND mascara) > 0 THEN
0095             Contagem_zona_md:= Contagem_zona_md +1;
0096         END_IF
0097     END_FOR
0098 END_FOR
0099
0100 (*-------------------------------------Caso 1 - triangulo ----------------------------------------------------*)
0101 (* não apicado por ser semelhante ao dos quadrantes*)
0102
0103 (*-------------------------------------Caso 2 - quadrado com furo no meio-------------------------------*)
0104 (*não aplicado por a zona coicidir com a do algoritmo de regiões comparadas *)
0105
0106 (*-------------------------------------Caso 3 - Quadrado com reentrâncias ------------------------------*)
0107 (* não apicado por ser semelhante ao dos quadrantes*)
0108
0109 (*  ------------------------------------ Caso 4 - triangulo com furo no meio-------------------------------*)
0110
0111 IF bt_caso4 THEN              (* Botão "Caso 4" foi premido *)
0112
0113     IF Contagem_zona_mm > 2 THEN      (*nas zonas sd e ie só admite um ou nenhum "1" activado*)
0114         sextante := 1;
0115     ELSE
```

Figura A.49. Algoritmo Sextantes – parte 2

```
0116          IF Contagem_zona_mm <= 1 THEN
0117          sextante := 0;
0118          END_IF
0119      END_IF
0120
0121  END_IF
0122
0123  (*-------------------------------------Caso 5 - Rectangulo ---------------------- ----------- -- ----------------*)
0124  (* não aplicável *)
0125
0126  (*-------------------------------------Caso 6 - quadrado dois furos ------------------------------------- *)
0127  (* não apicado por ser semelhante ao dos quadrantes*)
0128
0129  (*-------------------------------------Caso 7 - quadrado com três furos ---------------------------------*)
0130
0131  IF bt_caso7 THEN                (* Botão "Caso 7" foi premido *)
0132      IF Contagem_zona_se >= 1 AND Contagem_zona_ie >= 1 AND Contagem_zona_md >= 1 AND Contagem_zona_se <= 3
0133      AND Contagem_zona_ie <= 3 AND Contagem_zona_md <= 3 THEN          (*nas zonas sd e ie só admite um ou nenhum "1" activado*)
0134          sextante := 1;
0135      ELSE
0136          IF Contagem_zona_sd >= 1 AND Contagem_zona_id >= 1 AND Contagem_zona_me >= 1 AND Contagem_zona_me >= 1
0137          AND Contagem_zona_sd <= 3 AND Contagem_zona_id <= 3 AND Contagem_zona_me <= 3 THEN
0138          sextante := 0;
0139          END_IF
0140      END_IF
0141  END_IF
0142
```

Figura A.50. Algoritmo Sextantes – parte 3

```
0001    FUNCTION string_matching : INT
0002 VAR_INPUT
0003 END_VAR
0004 VAR
0005    Matriz_d: ARRAY [0..maximo_colunas] OF WORD; (* Matriz memória de "tabela de valores" da forma do triangulo desejado*)
0006    Matriz_e: ARRAY [0..maximo_colunas] OF WORD; (* Matriz memória com valores correspondentes à forma de rejeição*)
0007
0008    i: INT; t: INT; k: INT; d: INT; c: INT;
0009    E: BOOL; F: BOOL;
0010    contg_coluna: INT;
0011    Verifica_ok: BOOL;
0012    Verifica_ko: BOOL;
0013    mascara: WORD;
0014    Palavra_ok: WORD;
0015    Palavra_ko: WORD;
0016
0017    contg_coluna_ok: INT;
0018    contg_coluna_ko: INT;
0019    g: INT;
0020 END_VAR
```

```
0001
0002 (* _____ Algoritmo 3 - String Matching _____*)
0003
0004 (* ---------- Caso 1 - Triângulo ----------*)
0005
0006 IF bt_caso1 THEN            (* Botão "Caso 1" foi premido *)
0007
0008    (* sequência de palavras que descrevem a forma desejada*)
0009    Matriz_d[0] := 63; Matriz_d[1] := 31; Matriz_d[2] := 15; Matriz_d[3] := 7; Matriz_d[4] := 3; Matriz_d[5] := 1;
0010    (* a sua concetenação dá: 111111.011111.001111.000111.000011.000001 *)
0011
0012    (* sequência de palavras que descrevem forma de rejeição *)
0013    Matriz_e[0] := 1; Matriz_e[1] := 3; Matriz_e[2] := 7; Matriz_e[3] := 15; Matriz_e[4] := 31; Matriz_e[5] := 63;
0014    (* a sua concetenação dá: 000001.000011.000111.001111.011111.111111 *)
0015
0016 END_IF
0017
0018 (* ---------- Caso 2 - Quadrado furado ----------*)
0019
0020 IF bt_caso2 THEN            (* Botão "Caso 2" foi premido *)
0021
0022    (* sequência de palavras que descrevem a forma desejada*)
0023    Matriz_d[0] := 63; Matriz_d[1] := 63; Matriz_d[2] := 51; Matriz_d[3] := 51; Matriz_d[4] := 63; Matriz_d[5] := 63;
0024    (* a sua concetenação dá: 111111.111111.110011.110011.111111.111111 *)
0025
0026    (* sequência de palavras que descrevem forma de rejeição *)
0027    Matriz_e[0] := 63; Matriz_e[1] := 63; Matriz_e[2] := 63; Matriz_e[3] := 63; Matriz_e[4] := 63; Matriz_e[5] := 63;
0028    (* a sua concetenação dá: 111111.111111.111111.111111.111111.111111 *)
0029
0030 END_IF
0031
0032 (* ---------- Caso 3 - Quadrado com reentrâncias ----------*)
0033
0034 IF bt_caso3 THEN            (* Botão "Caso 3" foi premido *)
0035
0036    (* sequência de palavras que descrevem a forma desejada*)
0037    Matriz_d[0] := 63; Matriz_d[1] := 3; Matriz_d[2] := 63; Matriz_d[3] := 63; Matriz_d[4] := 48; Matriz_d[5] := 63;
0038    (* a sua concetenação dá: 111111.000011.111111.111111.110000.111111 *)
0039
0040    (* sequência de palavras que descrevem forma de rejeição *)
0041    Matriz_e[0] := 61; Matriz_e[1] := 61; Matriz_e[2] := 45; Matriz_e[3] := 45; Matriz_e[4] := 47; Matriz_e[5] := 47;
0042    (* a sua concetenação dá: 111101.111101.101101.101101.101111.101111 *)
0043
0044 END_IF
```

Figura A.51. Algoritmo String Matching – parte 1

```
0045
0046 (* ---------- Caso 4 - Triângulo furado ----------*)
0047
0048 IF bt_caso4 THEN              (* Botão "Caso 4" foi premido *)
0049
0050     (* sequência de palavras que descrevem a forma desejada*)
0051     Matriz_d[0] := 63; Matriz_d[1] := 31; Matriz_d[2] := 15; Matriz_d[3] := 7; Matriz_d[4] := 3; Matriz_d[5] := 1;
0052     (* a sua concetenação dá: 111111.011111.001111.000111.000011.000001 *)
0053
0054     (* sequência de palavras que descrevem forma de rejeição *)
0055     Matriz_e[0] := 63; Matriz_e[1] := 19; Matriz_e[2] := 1; Matriz_e[3] := 1; Matriz_e[4] := 3; Matriz_e[5] := 1;
0056     (* a sua concetenação dá: 111111.010001.000001.000001.000011.000001 *)
0057
0058 END_IF
0059
0060 (* ---------- Caso 5 - Rectângulo ----------*)
0061 (* não aplicável *)
0062
0063
0064 (* ---------- Caso 6 - Quadrado com dois furos----------*)
0065
0066 IF bt_caso6 THEN              (* Botão "Caso 6" foi premido *)
0067
0068     (* sequência de palavras que descrevem a forma desejada*)
0069     Matriz_d[0] := 63; Matriz_d[1] := 60; Matriz_d[2] := 63; Matriz_d[3] := 63; Matriz_d[4] := 15; Matriz_d[5] := 63;
0070     (* a sua concetenação dá: 111111.111100.111111.111111.001111.111111 *)
0071
0072     (* sequência de palavras que descrevem forma de rejeição *)
0073     Matriz_e[0] := 63; Matriz_e[1] := 47; Matriz_e[2] := 63; Matriz_e[3] := 63; Matriz_e[4] := 61; Matriz_e[5] := 63;
0074     (* a sua concetenação dá: 111111.001111111111.111111.111100.111111 *)
0075
0076 END_IF
0077
0078 (* ---------- Caso 7 - Quadrado com três furos----------*)
0079
0080 IF bt_caso7 THEN              (* Botão "Caso 7" foi premido *)
0081
0082     (* sequência de palavras que descrevem a forma desejada*)
0083     Matriz_d[0] := 63; Matriz_d[1] := 12; Matriz_d[2] := 63; Matriz_d[3] := 63; Matriz_d[4] := 51; Matriz_d[5] := 63;
0084     (* a sua concetenação dá: 111111.101101.111111.111111.110011.111111 *)
0085
0086     (* sequência de palavras que descrevem forma de rejeição *)
0087     Matriz_e[0] := 63; Matriz_e[1] := 51; Matriz_e[2] := 63; Matriz_e[3] := 63; Matriz_e[4] := 12; Matriz_e[5] := 63;
0088     (* a sua concetenação dá: 111111.110011.111111.111111.101101.111111 *)
0089
0090 END_IF
0091
0092 (*---------- Corpo geral do algoritmo ---------- *)
0093
0094 i := 0;                                   (* o i é o índice das colunas *)
0095 E := FALSE;
0096 Verifica_ok := FALSE;
0097 Verifica_ko := FALSE;
0098 WHILE i < 15 AND E = FALSE DO  (*enquanto nos encontrarmos dentro do espaço de amostragem o ciclo é executado *)
0099     IF Matriz_b[i] = Matriz_d[0] THEN   (* se a palavra i for igual à 1ª  procurada então inicia a confirmação da sequência*)
0100         Verifica_ok := TRUE;
0101     END_IF
0102     IF Matriz_b[i] = Matriz_e[0] THEN   (* se a palavra i for igual à 1ª procurada então inicia a confirmação da sequência*)
0103         Verifica_ko := TRUE;
0104     END_IF
0105         d := i;  (*incrementa valor da posição no espaço de caracteristicas para determinação da sequência*)
0106         c := 0; (* incrementa valor da sequência que está a ser verificada*)
0107         g := i;
0108         t := 0;
```

Figura A.52. Algoritmo String Matching – parte 2

```
0109
0110        WHILE Verifica_ok AND c <= 5DO      (* vai verificar se a sequência da forma pretendida se verifica *)
0111            contg_coluna_ok := 0;
0112            Palavra_ok := Matriz_b[d] XOR Matriz_d[c];
0113            FOR k := 0 TO 5 DO
0114                mascara := SHL(WORD#1,k);           (* Percorre linhas de baixo para cima *)
0115                IF (Palavra_ok AND mascara) > 0 THEN
0116                    Contg_coluna_ok:= Contg_coluna_ok +1;
0117                END_IF
0118            END_FOR
0119            IF Contg_coluna_ok <=1 THEN             (* permite apenas um erro (símbolo diferente) por palavra*)
0120                Verifica_ok := TRUE;
0121            ELSE
0122                Verifica_ok := FALSE;
0123            END_IF
0124            d:=d+1;
0125            c:=c+1;
0126        END_WHILE
0127        c := c-1;                           (* posição atingida correctamente na sequência desejada*)
0128        d := d-1;                           (* posição atingida na no espaço de características*)
0129        IF c = 5 THEN
0130            E := TRUE;
0131            EXIT;
0132        END_IF
0133
0134        WHILE Verifica_ko AND t <= 5DO      (* vai verificar se a sequencia da forma de rejeição se verifica *)
0135            contg_coluna_ko := 0;
0136            Palavra_ko := Matriz_b[g] XOR Matriz_e[t];
0137            FOR k := 0 TO 5 DO
0138                mascara := SHL(WORD#1,k);       (* Percorre linhas de baixo para cima *)
0139                IF (Palavra_ko AND mascara) > 0 THEN
0140                    Contg_coluna_ko := Contg_coluna_ko +1;
0141                END_IF
0142            END_FOR
0143            IF Contg_coluna_ko <=1 THEN
0144                Verifica_ko := TRUE;
0145            ELSE
0146                Verifica_ko := FALSE;
0147            END_IF
0148            g := g+1;
0149            t := t+1;
0150        END_WHILE
0151        t := t-1;
0152        IF t = 5 THEN
0153            F := TRUE;
0154            EXIT;
0155        END_IF
0156
0157    i := i+1;                       (*incrementa o i e volta a procurar encontrar a palavra desejada*)
0158 END_WHILE
0159
0160 IF E = TRUE THEN
0161     string_matching :=1;
0162 ELSE
0163     IF F = TRUE THEN
0164         string_matching :=0;
0165     ELSE
0166         alg_3_aux := TRUE;
0167     END_IF
0168 END_IF
0169
```

Figura A.53. Algoritmo String Matching – parte 3

```
0001 FUNCTION nh_find : INT
0002 VAR_INPUT
0003     linha : INT;
0004     inicio : INT;
0005 END_VAR
0006 VAR
0007
0008     mascara : WORD;
0009     old, new: WORD;          (* Registos temporários *)
0010     i: INT;
0011     maximo: INT;
0012
0013
0014 END_VAR
```

```
0001
0002 (* Transições Descendentes *)
0003 (* transição negativa (de 1 para 0) quando se procura numa linha horizontal de esquerda para a direita*)
0004
0005 (* definida uma linha - o algoritmo vai de coluna em coluna à procura de uma transição descendente *)
0006 (* reporta o valor i da coluna em que é detectada a última transição *)
0007
0008 IF horizontal THEN
0009     maximo := amostras_coluna;
0010 ELSE
0011     maximo := maximo_linhas;
0012 END_IF
0013
0014 mascara := SHL (WORD#1, linha);
0015
0016 IF inicio = 0 THEN
0017     old := 0;
0018 ELSE
0019     old := Matriz_b[inicio-1] AND mascara; (* palavra "old" na posição/coluna anterior *)
0020 END_IF
0021
0022
0023 FOR i:= inicio TO maximo DO
0024     new := Matriz_b[i] AND mascara;          (* palavra "new" na posição/coluna actual *)
0025     IF new < old THEN                        (* detecta uma transição descendente *)
0026         EXIT;
0027     ELSE
0028         old := new;
0029     END_IF
0030 END_FOR
0031
0032 IF i > maximo THEN
0033     IF old > 0 THEN
0034         nh_find := i;
0035     ELSE
0036         nh_find := -1;
0037     END_IF
0038 ELSE
0039     nh_find:= i;
0040 END_IF
0041
```

Figura A.54. Função nh_find

```
0001 FUNCTION nv_find : INT
0002 VAR_INPUT
0003     coluna : INT;
0004     inicio : INT;
0005 END_VAR
0006
0007 VAR
0008     mascara : WORD;
0009     old, new : WORD;
0010     i: INT;
0011 END_VAR
```

```
0001
0002 (* Transições Horizontal Negativa *)
0003 (* transição negativa (de 1 para 0) quando se procura numa coluna vertical de baixo para cima*)
0004
0005 (* definida uma coluna - o algoritmo vai de linha em linha à procura de uma transição negativa *)
0006 (* reporta o valor i da linha em que é detectada a última transição*)
0007
0008 IF inicio = 0 THEN
0009     old := 0;
0010 ELSE
0011     old := Matriz_b[coluna] AND SHL(WORD#1, inicio-1);
0012 END_IF
0013
0014 FOR i := inicio TO 5 DO
0015     new := SHR((Matriz_b[coluna] AND SHL(WORD#1,i)), i);
0016     IF new < old THEN
0017             EXIT;
0018     END_IF
0019     old := new;
0020 END_FOR
0021
0022
0023 IF i=6 THEN
0024         IF old <> 0 THEN
0025             nv_find := i;
0026         ELSE
0027             nv_find := -1;
0028         END_IF
0029 ELSE
0030         nv_find := i;
0031 END_IF
```

Figura A.55. Função nv_find

```
0001 FUNCTION ph_find : INT
0002 VAR_INPUT
0003     linha : INT; (* adquire o valor do i - que vai variar de 5 até 0 - percorre as linha no sento decrescente *)
0004     inicio : INT; (* adquire o valor do k - e salte de valor em valor obtido nesta mesma função (ph_find final) *)
0005     (* coluna de inicio de novo ciclo*)
0006 END_VAR
0007
0008
0009 VAR
0010
0011     mascara : WORD;
0012     old, new : WORD;
0013     i: INT;
0014     maximo: INT;
0015
0016 END_VAR
```

```
0001
0002 (* Transições Ascendentes *)
0003 (* transição positiva (de 0 para 1) quando se procura numa linha horizontal de esquerda para a direita*)
0004
0005 (* definida uma linha - o algoritmo vai de coluna em coluna à procura de uma transição ascendente *)
0006 (* reporta o valor i da coluna em que é detectada a última transição*)
0007
0008 IF horizontal THEN
0009     maximo := amostras_coluna;
0010 ELSE
0011     maximo := maximo_linhas;
0012 END_IF
0013
0014 mascara := SHL (WORD#1, linha);
0015
0016 IF inicio = 0 THEN
0017     old := 0;
0018 ELSE
0019     old := Matriz_b[inicio-1] AND mascara;          (* palavra correspondente à coluna anteriar àquela que se pretende testar *)
0020 END_IF
0021
0022
0023 FOR i:= inicio TO maximo DO (* da coluna inicial no sentido crescente. no máximo atinge o valor máximo *)
0024     new := Matriz_b[i] AND mascara;
0025     IF new > old THEN (* se detectar uma transição ascendente *)
0026         EXIT;
0027     ELSE
0028         old := new;
0029     END_IF
0030 END_FOR
0031
0032 IF i > maximo THEN
0033     ph_find := -1; (*se não encontrar nenhuma new > old*)
0034 ELSE
0035     ph_find:= i; (*se encontrar reporta valor *)
0036 END_IF
0037
0038
0039
0040
0041
```

Figura A.56. Função ph_find

```
0001 FUNCTION pv_find : INT
0002 VAR_INPUT
0003     coluna : INT;
0004     inicio : INT;
0005 END_VAR
0006 VAR
0007     mascara : WORD;
0008     old, new : WORD;
0009     i: INT;
0010 END_VAR
0001
0002 (* Transições Horizontal Positiva *)
0003 (* transição positiva (de 0 para 1) quando se procura numa coluna vertical de baixo para cima*)
0004
0005 (* definida uma coluna - o algoritmo vai de linha em linha à procura de uma transição positiva *)
0006 (* reporta o valor i da linha em que é detectada a última transição*)
0007
0008 IF inicio = 0 THEN
0009     old := 0;
0010 ELSE
0011     old := Matriz_b[coluna] AND SHL(WORD#1, inicio-1);
0012 END_IF
0013
0014
0015
0016 FOR i := inicio TO 5 DO
0017     new := SHR((Matriz_b[coluna] AND SHL(WORD#1,i)), i);
0018     IF new > old THEN
0019             EXIT;
0020     END_IF
0021     old := new;
0022 END_FOR
0023
0024
0025 IF i=6 THEN
0026         pv_find := -1;
0027 ELSE
0028         pv_find := i;
0029 END_IF
```

Figura A.57. Função pv_find

```
0001     FUNCTION triangulo_contorno : INT
0002 VAR_INPUT
0003
0004 END_VAR
0005
0006 VAR
0007     i: INT; j: INT; k: INT;
0008
0009 (* Estados do sistema *)
0010     A: BOOL; B: BOOL; C: BOOL; D: BOOL; E: BOOL;
0011     G: BOOL; U: BOOL; V: BOOL; F: BOOL; H: BOOL;
0012
0013  (* adquire valores das primitivas - chamando as subrotinas correspondentes*)
0014     retorna_pv: INT; retorna_ph: INT; retorna_nv: INT; retorna_nh: INT;
0015
0016 (*variáveis auxiliares *)
0017     l: INT; t: INT; rr: INT; w: INT;
0018     m: INT; p: INT; q: INT; n: INT; x: INT; y: INT;
0019
0020 END_VAR
```

```
0001 (* Algoritmo - contorno integral - detecta o contorno exacto do triangulo pretendido *)
0002
0003 IF bt_caso1 THEN              (* Botão "Caso 1" foi premido *)
0004
0005 (* Ciclo que detecta o contorno vertival inicial *)
0006 i := 0;
0007 j := 0;
0008 retorna_pv := primitiva_pv(i,j);              (* subrotina om valor da primitiva - transição ascendente (positiva/vertical) *)
0009 IF retorna_pv = 1 THEN
0010     i := i+1;
0011     k := 1;
0012     A := TRUE;                          (* estado A activado *)
0013     WHILE A = TRUE DO
0014         retorna_pv := primitiva_pv(i,j);
0015         IF retorna_pv = 1 THEN
0016             i := i+1;
0017             k := k+1;                   (* k - contador das transições ascendentes*)
0018         ELSE
0019             A := FALSE;                 (* se não houver transição ascendente significa que se saiu desse estado *)
0020             i := i-1;
0021         END_IF
0022     END_WHILE
0023 END_IF
0024
0025
0026 IF A = FALSE  THEN
0027     t := 0;              (* t - contador das transições descendentes*)
0028     retorna_ph := primitiva_ph(i,j);
0029     IF retorna_ph = 1 THEN          (* se ocurrer transiçã horizontal positiva passa para o estado B *)
0030         l := 0;                     (* l - contador das transições horizontais positivas*)
0031         B := TRUE;
0032         C := FALSE;
0033         WHILE B = TRUE OR C = TRUE DO
0034             WHILE B = TRUE  DO
0035                 retorna_ph := primitiva_ph(i,j);
0036                 IF retorna_ph = 1 THEN
0037                     j := j+1;
0038                     l := l+1;
0039                 ELSE
0040                     j := j-1;
0041                     B := FALSE;
0042                     retorna_nv := primitiva_nv(i,j);
0043                     IF retorna_nv = 1 THEN
0044                         C:=TRUE;
```

Figura A.58. Algoritmo Contorno – parte 1

```
0045                    END_IF
0046                END_IF
0047            END_WHILE
0048            WHILE C = TRUE DO
0049                retorna_nv := primitiva_nv(i,j);
0050                IF retorna_nv =1 THEN
0051                    i := i-1;
0052                    t := t+1;
0053                ELSE
0054                    i :=i+1;
0055                    C := FALSE;
0056                    retorna_ph := primitiva_ph(i,j);
0057                    IF retorna_ph =1 THEN
0058                        B:=TRUE;
0059                    END_IF
0060                    retorna_nh: =primitiva_nh(i,j);
0061                    IF retorna_nh=1 AND i=0 THEN
0062                        B:=FALSE;
0063                    END_IF
0064                END_IF
0065            END_WHILE
0066            i:=i-1;
0067            j:=j+1;
0068        END_WHILE
0069    END_IF
0070 END_IF
0071
0072 i:=i+1;
0073 j:=j-1;
0074 IF (B=FALSE) AND (C=FALSE) THEN
0075    IF  t = k THEN                          (* se houve o mesmo numero de transições ascendentes como descentes *)
0076        retorna_nh : = primitiva_nh(i,j);
0077        IF retorna_nh = 1 THEN
0078            rr := 1;
0079            D := TRUE;
0080            WHILE D = TRUE AND j>0 DO
0081                retorna_nh : = primitiva_nh(i,j);
0082                IF retorna_nh = 1 THEN
0083                    j := j-1;
0084                    rr :=rr+1;
0085                END_IF
0086                IF j=0 THEN
0087                D:=FALSE;
0088                END_IF
0089            END_WHILE
0090        END_IF
0091    END_IF
0092 END_IF
0093
0094 IF D=FALSE THEN
0095    retorna_pv := primitiva_pv(i,j);
0096    IF retorna_pv = 1 AND k=t AND rr=l THEN
0097        E := TRUE;
0098    ELSE
0099        E := FALSE;
0100    END_IF
0101 END_IF
0102
0103 (* Identificação da forma de rejeição *)
0104
0105 x := 0;
0106 y := 0;
0107 m := 0;                          (* m - contador das transições ascendentes*)
0108 p := 0;                          (* p - contador das transições horizontais positivas*)
```

Figura A.59. Algoritmo Contorno – parte 2

```
0109 retorna_pv := primitiva_pv(x,y);
0110 IF retorna_pv = 1 THEN
0111         x := x+1;
0112         m := m+1;
0113         G := TRUE;                    (* se detectar  transição ascendente passa para o estado G *)
0114         H := FALSE;
0115         WHILE G = TRUE OR H = TRUE DO
0116             WHILE G = TRUE  DO
0117                 retorna_pv := primitiva_pv(x,y);
0118                 IF retorna_pv = 1 THEN
0119                     x := x+1;
0120                     m := m+1;
0121                 ELSE
0122                     x := x-1;
0123                     G := FALSE;
0124                     retorna_ph := primitiva_ph(x,y);
0125                     IF retorna_ph = 1 THEN
0126                         H:=TRUE;
0127                     END_IF
0128                 END_IF
0129             END_WHILE
0130             WHILE H = TRUE DO
0131                 retorna_ph := primitiva_ph(x,y);
0132                 IF retorna_ph =1 THEN
0133                     y := y+1;
0134                     p := p+1;
0135                 ELSE
0136                     y :=y-1;
0137                     H := FALSE;
0138                     retorna_pv := primitiva_pv(x,y);
0139                     IF retorna_pv =1 THEN
0140                         G:=TRUE;
0141                     END_IF
0142                     retorna_nv:=primitiva_nv(x,y);
0143                     IF retorna_nv=1 AND x=5 THEN
0144                         G:=FALSE;
0145                     END_IF
0146                 END_IF
0147             END_WHILE
0148             y:=y+1;
0149             x:=x+1;
0150         END_WHILE
0151     END_IF
0152
0153 (* Ciclo que detecta o contorno vertival inicial *)
0154 x:=x-1;
0155 y:=y-1;
0156 n:= 0;
0157 IF (G=FALSE) AND (H=FALSE) THEN
0158 retorna_nv := primitiva_nv(x,y);          (* subrotina om valor da primitiva - transição ascendente (positiva/vertical) *)
0159     IF retorna_nv = 1 THEN
0160         x := x-1;
0161         n := n+1;
0162         F := TRUE;                        (* estado A activado *)
0163         WHILE F = TRUE DO
0164             retorna_nv := primitiva_nv(x,y);
0165             IF retorna_nv = 1 THEN
0166                 x := x-1;
0167                 n := n+1;                 (* k - contador das transições ascendentes*)
0168             ELSE
0169                 F := FALSE;               (* se não houver transição ascendente significa que se saiu desse estado *)
0170             END_IF
0171         END_WHILE
0172     END_IF
```

Figura A.60. Algoritmo Contorno – parte 3

```
0173 END_IF
0174
0175 x:= x+1;
0176 IF F=FALSE THEN
0177     IF  n = m THEN                          (* se houve o mesmo numero de transições ascendentes como descentes *)
0178         retorna_nh := primitiva_nh(x,y);
0179         IF retorna_nh = 1 THEN
0180             q := 1;
0181             U := TRUE;
0182             WHILE U = TRUE AND y>0 DO
0183                 retorna_nh := primitiva_nh(x,y);
0184                 IF retorna_nh = 1 THEN
0185                     y := y-1;
0186                     q :=q+1;
0187                 END_IF
0188                 IF y=0 THEN
0189                 U:=FALSE;
0190                 END_IF
0191             END_WHILE
0192         END_IF
0193     END_IF
0194 END_IF
0195
0196 IF U=FALSE THEN
0197     retorna_pv := primitiva_pv(x,y);
0198     IF retorna_pv = 1 AND n=m AND p=q THEN
0199         V := TRUE;
0200     ELSE
0201         V := FALSE;
0202     END_IF
0203 END_IF
0204
0205 IF E THEN
0206     triangulo_contorno := 1;
0207 ELSE
0208     IF V THEN
0209         triangulo_contorno := 0;
0210     ELSE
0211         alg_5_aux := TRUE;
0212     END_IF
0213 END_IF
0214
0215 END_IF
```

Figura A.61. Algoritmo Contorno – parte 4

```
0001 FUNCTION triangulos_2 : INT
0002 VAR_INPUT
0003 END_VAR
0004 VAR
0005     i: INT; K: INT; P: INT; t: INT; j: INT; v: INT; u: INT; x: INT; y: INT;
0006     OK: BOOL; KO: BOOL; OK_2: BOOL; KO_2: BOOL;
0007     new: WORD; mascara: WORD; old: WORD;   mascara_old: WORD; mascara_new: WORD;
0008
0009     retorna_nv_vector: ARRAY [0..15] OF INT; retorna_nh_vector: ARRAY [0..15] OF INT;
0010     retorna_ok: INT; retorna_ko: INT;
0011     retorna_nh: INT; retorna_ph: INT; retorna_nv: INT; retorna_pv: INT;
0012     retorna_nh_s: INT; retorna_nh_i: INT; retorna_ph_i: INT; retorna_ph_s: INT; retorna_ph_m: INT;
0013
0014     conta_trans_ph: INT; conta_trans_nh: INT; conta_trans_pv: INT; conta_trans_nv: INT;
0015     conta_trans_nh_i: INT; conta_trans_nh_s: INT; conta_trans_ph_i: INT; conta_trans_ph_s: INT; conta_trans_ph_m: INT;
0016 END_VAR
0001
0002 (* _____ Algoritmo 4 - Sequência de transições (Caso 1) _____*)
0003
0004 (*---------- Corpo geral do algoritmo ----------*)
0005
0006 (*---------- Caso 1 - Identificaçã do Triângulo ----------*)
0007
0008 IF bt_caso1 THEN              (* Botão "Caso 1" foi premido *)
0009
0010     K := 0;
0011     KO := FALSE;
0012     FOR i := 0 TO 5 DO
0013         retorna_ko:= ph_find (i,k);        (*diz se na linha i e apartir da coluna k detecta alguma transição positiva *)
0014         IF retorna_ko = -1 THEN
0015             KO := FALSE;
0016             (*RETURN;*)
0017             EXIT;
0018         ELSE
0019             K:= retorna_ko+1;     (*avança na linha - e avança uma coluna relativamente à da ultima transição detectada *)
0020             KO := TRUE;
0021         END_IF
0022     END_FOR
0023
0024     P := 0;
0025     OK := FALSE;
0026     FOR t := 5 TO 0 BY -1 DO
0027         retorna_ok:= nh_find (t,P);        (*diz se na linha t e apartir da coluna P detecta alguma transição negativa *)
0028         IF retorna_ok = -1 THEN
0029             OK := FALSE;
0030             (*RETURN;*)
0031             EXIT;
0032         ELSE
0033         P:= retorna_ok+1;              (* "avançar" na linha - avança uma coluna relativamente à da ultima transição detectada *)
0034         OK := TRUE;
0035         END_IF
0036     END_FOR
0037
0038 END_IF
0039 (*---------- caso 2 - identificaçã de quadrado furado ----------*)
0040
0041 IF bt_caso2 THEN              (* Botão "Caso 2" foi premido *)
0042
0043     conta_trans_ph := 0;
0044     conta_trans_nh := 0;
0045     FOR x := 2 TO 3 DO                    (* x- coluna ; y - linha *)
0046         FOR y := 1 TO 4 DO
0047             retorna_nh:= primitiva_nh(y,x);      (*diz se na linha y e na coluna x detecta alguma transição horizontal negativa *)
0048             IF retorna_nh = 1 THEN
```

Figura A.62. Algoritmo Transições – parte 1

```
0049                conta_trans_nh := conta_trans_nh +1;
0050            END_IF
0051            retorna_ph:= primitiva_ph(y,x);      (*diz se na linha y e na coluna x detecta alguma transição horizontal negativa *)
0052            IF retorna_ph = 1 THEN
0053                conta_trans_ph := conta_trans_ph +1;
0054            END_IF
0055        END_FOR
0056    END_FOR
0057
0058    conta_trans_pv := 0;
0059    conta_trans_nv := 0;
0060    FOR v := 2 TO 3 DO                       (* u- coluna ; v - linha *)
0061        FOR u := 1 TO 4 DO
0062            retorna_nv:= primitiva_nv(v,u);      (*diz se na linha y e na coluna x detecta alguma transição horizontal negativa *)
0063            IF retorna_nv = 1 THEN
0064                conta_trans_nv := conta_trans_nv +1;
0065            END_IF
0066            retorna_pv:= primitiva_pv(v,u);      (*diz se na linha y e na coluna x detecta alguma transição horizontal negativa *)
0067            IF retorna_pv = 1 THEN
0068                conta_trans_pv := conta_trans_pv +1;
0069            END_IF
0070        END_FOR
0071    END_FOR
0072
0073    IF (conta_trans_pv + conta_trans_ph) >=4 AND (conta_trans_nv + conta_trans_nh) >=4 THEN
0074        OK := TRUE;
0075        KO := FALSE;
0076    END_IF
0077    IF (conta_trans_pv + conta_trans_ph) <= 2 AND (conta_trans_nv + conta_trans_nh) <= 2 THEN
0078        OK := FALSE;
0079        KO := TRUE;
0080    END_IF
0081
0082
0083 (*
0084    conta_trans_pv := 0;
0085    conta_trans_nv := 0;
0086    FOR x := 1 TO 4 DO                       (* x- coluna ; y - linha *)
0087        FOR y := 2 TO 3 DO
0088            mascara := SHL (WORD#1, y);
0089            old := Matriz_b[x] AND mascara;
0090            new := Matriz_b[x+1] AND mascara;
0091
0092            IF old > new THEN
0093                conta_trans_nv := conta_trans_nv +1;
0094            END_IF
0095            IF old < new THEN
0096                conta_trans_pv := conta_trans_pv +1;
0097            END_IF
0098        END_FOR
0099    END_FOR
0100
0101    IF conta_trans_pv  >= 2 THEN
0102        OK_2 := TRUE;
0103        KO_2 := FALSE;
0104    END_IF
0105    IF conta_trans_pv  < 1  THEN
0106        OK_2 := FALSE;
0107        KO_2 := TRUE;
0108    END_IF
0109 *)
0110 END_IF
0111
0112 (*------------------------------------------ caso 3 - identificaçã de quadrado reentrâncias ----------------------------------------------*)
```

Figura A.63. Algoritmo Transições – parte 2

```
0113
0114 IF bt_caso3 THEN              (* Botão "Caso 3" foi premido *)
0115
0116     conta_trans_pv := 0;
0117     conta_trans_nv := 0;
0118     FOR y := 0 TO 5 DO                    (* x- coluna ; y - linha *)
0119         retorna_nv := nv_find(1,y);       (* procura na coluna 1 a posição da transição horizontal positiva *)
0120         IF retorna_nv = -1 THEN
0121             alg_4_aux := TRUE;
0122         ELSE
0123             conta_trans_nv := retorna_nv;
0124         END_IF
0125         retorna_pv := pv_find(4,y);       (* procura na coluna 4 a posição da transição horizontal negativa *)
0126         IF retorna_pv = -1 THEN
0127             alg_4_aux := TRUE;
0128         ELSE
0129             conta_trans_pv := retorna_pv;
0130         END_IF
0131     END_FOR
0132
0133     conta_trans_ph := 0;
0134     conta_trans_nh := 0;
0135     FOR x := 0 TO 5 DO                    (* x- coluna ; y - linha *)
0136         retorna_ph := ph_find(1,x);       (* procura na linha 1 a posição da transição ascendente *)
0137         IF retorna_ph = -1 THEN
0138             alg_4_aux := TRUE;
0139         ELSE
0140             conta_trans_ph := retorna_ph;
0141         END_IF
0142         retorna_nh := nh_find(4,x);       (* procura na linha 4 a posição da transição descendente *)
0143         IF retorna_nh = -1 THEN
0144             alg_4_aux := TRUE;
0145         ELSE
0146             conta_trans_nh := retorna_nh;
0147         END_IF
0148     END_FOR
0149
0150 (* valores encontrado dizem respeito às ultimas transições daquele tipo, encontradas no padrão *)
0151
0152     IF conta_trans_pv = 4 AND conta_trans_ph = 5 AND conta_trans_nv =2 AND conta_trans_nh = 6 THEN
0153         OK := TRUE;
0154         KO := FALSE;
0155         alg_4_aux := FALSE;
0156     END_IF
0157     IF conta_trans_pv = 5 AND conta_trans_ph = 4 AND conta_trans_nv =5 AND conta_trans_nh = 2 THEN
0158         OK := FALSE;
0159         KO := TRUE;
0160         alg_4_aux := FALSE;
0161     END_IF
0162
0163 END_IF
0164
0165 (*------------------------------------------ caso 4 - identificaçã de Triângulo furado ------------------------------------------*)
0166
0167 IF bt_caso4 THEN              (* Botão "Caso 4" foi premido *)
0168
0169     conta_trans_nv := 0;
0170     FOR y := 0 TO 5 DO                    (* x- coluna ; y - linha *)
0171         retorna_nv := nv_find(3,y);       (* procura na coluna 1 a posição da transição horizontal positiva *)
0172         IF retorna_nv = -1 THEN
0173             alg_4_aux := TRUE;
0174         ELSE
0175             conta_trans_nv := retorna_nv;
0176         END_IF
```

Figura A.64. Algoritmo Transições – parte 3

```
0177    END_FOR
0178
0179    conta_trans_nh := 0;
0180    FOR x := 0 TO 5 DO                          (* x- coluna ; y - linha *)
0181            retorna_nh := nh_find(3,x);         (* procura na linha 4 a posição da transição descendente *)
0182            IF retorna_nh = -1 THEN
0183                alg_4_aux := TRUE;
0184            ELSE
0185                conta_trans_nh := retorna_nh;
0186            END_IF
0187    END_FOR
0188
0189 (* valores encontrado dizem respeito às ultimas transições daquele tipo, encontradas no padrão *)
0190    IF conta_trans_nv =3 AND conta_trans_nh = 3 THEN
0191        OK := TRUE;
0192        KO := FALSE;
0193        alg_4_aux := FALSE;
0194    END_IF
0195    IF conta_trans_nv =1 AND conta_trans_nh = 1 THEN
0196        OK := FALSE;
0197        KO := TRUE;
0198        alg_4_aux := FALSE;
0199    END_IF
0200
0201 END_IF
0202
0203 (*------------------------------------------ caso 5 - Rectângulo ------------------------------------------*)
0204
0205 IF bt_caso5 THEN              (* Botão "Caso 5" foi premido *)
0206
0207    FOR x := 0 TO amostras_coluna DO
0208        FOR y := 0 TO amostras_linha +1 DO                      (* x- coluna ; y - linha *)
0209            retorna_nv_vector[x] := nv_find(x,y);       (* procura na coluna 1 a posição da transição horizontal positiva *)
0210            IF retorna_nv_vector[x] = -1 THEN
0211                alg_4_aux := TRUE;
0212            END_IF
0213        END_FOR
0214    END_FOR
0215    conta_trans_nv := 0;
0216    med_nv := 0;
0217    FOR x := 0 TO amostras_coluna DO
0218        conta_trans_nv := conta_trans_nv +retorna_nv_vector[x];
0219    END_FOR
0220    med_nv := conta_trans_nv/(amostras_coluna + 1);
0221
0222    FOR y := 0 TO amostras_linha DO
0223        FOR x := 0 TO amostras_coluna DO                        (* x- coluna ; y - linha *)
0224            retorna_nh_vector[y] := nh_find(y,x);       (* procura na linha 4 a posição da transição descendente *)
0225            IF retorna_nh_vector[y] = -1 THEN
0226                alg_4_aux := TRUE;
0227            END_IF
0228        END_FOR
0229    END_FOR
0230    conta_trans_nh := 0;
0231    med_nh := 0;
0232    FOR y := 0 TO amostras_linha DO
0233        conta_trans_nh := conta_trans_nh +retorna_nh_vector[y];
0234    END_FOR
0235    med_nh := conta_trans_nh/(amostras_linha +1);
0236
0237 (* valores encontrado dizem respeito às ultimas transições daquele tipo, encontradas no padrão *)
0238    IF med_nv = med_nh THEN
0239        OK := TRUE;
0240        KO := FALSE;
```

Figura A.65. Algoritmo Transições – parte 4

```
0241        alg_4_aux := FALSE;
0242    ELSE
0243        IF med_nv > (med_nh +1) OR med_nh > (med_nv +1) THEN (*rectângulos vertical e horizontal respectivamente *)
0244            OK := FALSE;
0245            KO := TRUE;
0246            alg_4_aux := FALSE;
0247        ELSE
0248            alg_4_aux := TRUE;
0249            KO := FALSE;
0250            OK := FALSE;
0251        END_IF
0252    END_IF
0253
0254 END_IF
0255
0256 (*------------------------------------------ caso 6 - Quadrado com dois furos ------------------------------------------*)
0257
0258 IF bt_caso6 THEN              (* Botão "Caso 6" foi premido *)
0259
0260    conta_trans_ph_i := 0;
0261    conta_trans_ph_s := 0;
0262    FOR x := 0 TO 5 DO                          (* x- coluna ; y - linha *)
0263            retorna_ph_i := ph_find(1,x);       (* procura na linha 1 a posição da transição ascendente *)
0264            IF retorna_ph_i = -1 THEN
0265                alg_4_aux := TRUE;
0266            ELSE
0267                conta_trans_ph_i := retorna_ph_i;
0268            END_IF
0269            retorna_ph_s := ph_find(4,x);       (* procura na linha 4 a posição da transição descendente *)
0270            IF retorna_ph_s = -1 THEN
0271                alg_4_aux := TRUE;
0272            ELSE
0273                conta_trans_ph_s := retorna_ph_s;
0274            END_IF
0275    END_FOR
0276
0277 (* valores encontrado dizem respeito às ultimas transições daquele tipo, encontradas no padrão *)
0278
0279    IF conta_trans_ph_i <= 2 AND conta_trans_ph_s >= 4 THEN
0280        OK := TRUE;
0281        KO := FALSE;
0282        alg_4_aux := FALSE;
0283    END_IF
0284    IF (conta_trans_ph_i >= 4 AND conta_trans_ph_s <= 2) XOR conta_trans_ph_i = 2 XOR conta_trans_ph_s = 5 OR
0285    (conta_trans_ph_s <= 1 AND conta_trans_ph_i <= 1) THEN
0286        OK := FALSE;
0287        KO := TRUE;
0288        alg_4_aux := FALSE;
0289    END_IF
0290
0291 END_IF
0292
0293 (*------------------------------------------ caso 7 - Quadrado com três furos ------------------------------------------*)
0294
0295 IF bt_caso7 THEN              (* Botão "Caso 7" foi premido *)
0296
0297    conta_trans_ph_i := 0;
0298    conta_trans_ph_s := 0;
0299    conta_trans_ph_m := 0;
0300    (* função ( linha , coluna)  *)
0301
0302 FOR x := 0 TO 2 DO
0303    retorna_ph_i := ph_find(1,x);       (* procura na linha 1 a posição da transição ascendente *)
0304    IF retorna_ph_i = -1 THEN
```

Figura A.66. Algoritmo Transições – parte 5

```
0305        alg_4_aux := TRUE;
0306    ELSE
0307        conta_trans_ph_i := retorna_ph_i;
0308    END_IF
0309
0310    retorna_ph_s := ph_find(4,x);        (* procura na linha 4 a posição da transição descendente *)
0311    IF retorna_ph_s = -1 THEN
0312        alg_4_aux := TRUE;
0313    ELSE
0314        conta_trans_ph_s := retorna_ph_s;
0315    END_IF
0316 END_FOR
0317
0318 FOR x := 0 TO 5 DO
0319    retorna_ph_m := ph_find(3,x);        (* procura na linha 4 a posição da transição descendente *)
0320    IF retorna_ph_m = -1 THEN
0321        alg_4_aux := TRUE;
0322    ELSE
0323        conta_trans_ph_m := retorna_ph_m;
0324    END_IF
0325 END_FOR
0326 (* valores encontrado dizem respeito às ultimas transições daquele tipo, encontradas no padrão *)
0327
0328    IF conta_trans_ph_i = conta_trans_ph_s AND conta_trans_ph_i < conta_trans_ph_m THEN
0329        OK := TRUE;
0330        KO := FALSE;
0331        alg_4_aux := FALSE;
0332    END_IF
0333    IF conta_trans_ph_i = conta_trans_ph_s AND conta_trans_ph_i > conta_trans_ph_m THEN
0334        OK := FALSE;
0335        KO := TRUE;
0336        alg_4_aux := FALSE;
0337    END_IF
0338
0339 END_IF
0340 (*-------------------------------------------------------------------------------------------------------------------*)
0341
0342    IF OK OR OK_2 THEN
0343        triangulos_2 := 1;
0344    ELSE
0345        IF KO OR KO_2 THEN
0346            triangulos_2 := 0;
0347        ELSE
0348            alg_4_aux := TRUE;
0349        END_IF
0350    END_IF
```

Figura A.67. Algoritmo Transições – parte 6

```
0001 FUNCTION triangulos_2_orientacao : INT
0002 VAR_INPUT
0003 END_VAR
0004 VAR
0005     K: INT; P: INT;
0006     i: INT; t: INT;
0007     x: INT; y: INT;
0008
0009     retorna_se: INT; retorna_id: INT; retorna_sd: INT; retorna_ie: INT;
0010     retorna_ph: INT; retorna_nh: INT; retorna_nv: INT; retorna_pv: INT;
0011     conta_trans_ph: INT; conta_trans_ph_s: INT; conta_trans_nh: INT; conta_trans_nh_s: INT;
0012     conta_trans_nv: INT; conta_trans_pv: INT;
0013
0014 END_VAR
```

```
0001
0002 (* ---------- Caso 8 - Caso de orientação dos Triângulos ----------*)
0003
0004 IF bt_caso8 THEN              (* Botão "Caso 8" foi premido *)
0005
0006 K := 0;
0007 se_tri_2 := FALSE;
0008 FOR i := 5 TO 0 BY -1 DO
0009     retorna_se:= ph_find (i,k);
0010     IF retorna_se = -1 THEN
0011         se_tri_2 := TRUE;
0012     ELSE
0013         K:= retorna_se;
0014     END_IF
0015 END_FOR
0016
0017 K := 0;
0018 id_tri_2 := FALSE;
0019 FOR i := 5 TO 0 BY -1 DO
0020     retorna_id:= nh_find (i,k);
0021     IF retorna_id = -1 THEN
0022         id_tri_2 := TRUE;
0023     ELSE
0024         K:= retorna_id;
0025     END_IF
0026 END_FOR
0027
0028 P := 0;
0029 sd_tri_2 := FALSE;
0030 FOR t := 0 TO 5 DO
0031     retorna_sd:= nv_find (t,P);
0032     IF retorna_sd = -1 THEN
0033         sd_tri_2 := TRUE;
0034     ELSE
0035     P:= retorna_sd;
0036     END_IF
0037 END_FOR
0038
0039 P := 0;
0040 ie_tri_2 := FALSE;
0041 FOR t := 0 TO 5 DO
0042     retorna_ie:= pv_find (t,P);
0043     IF retorna_ie = -1 THEN
0044         ie_tri_2 := TRUE;
0045     ELSE
0046     P:= retorna_ie;
0047     END_IF
0048 END_FOR
0049
0050 ELSE
```

Figura A.68. Algoritmo Orientação_triangulos – parte 1

```
0051    ie_tri_2 := FALSE; id_tri_2 := FALSE; se_tri_2 := FALSE; sd_tri_2 := FALSE;
0052 END_IF
0053
0054 (* ---------------------------------------------- Caso 9 - Caso de orientação dos Quadrados sem esquina----------------------------------------*)
0055
0056 (* verificar quando ocorrem as transições ascendentes e descendentes nas linhas 0 e 5 *)
0057
0058 IF bt_caso9 THEN              (* Botão "Caso 9" foi premido *)
0059
0060    conta_trans_ph := 0;
0061    conta_trans_nh := 0;
0062    conta_trans_ph_s := 0;
0063    conta_trans_nh_s := 0;
0064
0065    FOR x := 0 TO 5 DO                      (* x- coluna ; y - linha *)
0066        retorna_nh := nh_find(4,x);       (* procura na linha 4 a posição da transição descendente *)
0067        IF retorna_nh = -1 THEN
0068            EXIT;
0069        ELSE
0070            conta_trans_nh_s := retorna_nh;
0071        END_IF
0072    END_FOR
0073
0074    FOR x := 0 TO 5 DO                      (* x- coluna ; y - linha *)
0075        retorna_nh := nh_find(1,x);       (* procura na linha 1 a posição da transição descendente *)
0076        IF retorna_nh = -1 THEN
0077            EXIT;
0078        ELSE
0079            conta_trans_nh := retorna_nh;
0080        END_IF
0081    END_FOR
0082
0083    FOR x := 0 TO 5 DO                      (* x- coluna ; y - linha *)
0084        retorna_ph := ph_find(4,x);       (* procura na linha 4 a posição da transição ascendente *)
0085        IF retorna_ph = -1 THEN
0086            EXIT;
0087        ELSE
0088            conta_trans_ph_s := retorna_ph;
0089        END_IF
0090    END_FOR
0091
0092
0093    FOR x := 0 TO 5 DO                      (* x- coluna ; y - linha *)
0094        retorna_ph := ph_find(1,x);       (* procura na linha 1 a posição da transição ascendente *)
0095        IF retorna_ph = -1 THEN
0096            EXIT;
0097        ELSE
0098            conta_trans_ph := retorna_ph;
0099        END_IF
0100    END_FOR
0101
0102 (* valores encontrado dizem respeito às ultimas transições daquele tipo, encontradas no padrão *)
0103    IF conta_trans_ph_s <= (conta_trans_ph -1) THEN
0104        ie_quad_2 := TRUE; se_quad_2 := FALSE; sd_quad_2 := FALSE; id_quad_2 := FALSE;
0105    END_IF
0106    IF conta_trans_ph <= (conta_trans_ph_s -1) THEN
0107        se_quad_2 := TRUE; ie_quad_2 := FALSE; sd_quad_2 := FALSE; id_quad_2 := FALSE;
0108    END_IF
0109    IF conta_trans_nh_s <= (conta_trans_nh -1) THEN
0110        sd_quad_2 := TRUE; ie_quad_2 := FALSE; se_quad_2 := FALSE; id_quad_2 := FALSE;
0111    END_IF
0112    IF conta_trans_nh <= (conta_trans_nh_s -1) THEN
0113        id_quad_2 := TRUE; ie_quad_2 := FALSE; se_quad_2 := FALSE; sd_quad_2 := FALSE;
0114    END_IF
```

Figura A.69. Algoritmo Orientação_triangulos – parte 2

```
0115
0116 ELSE
0117     ie_quad_2 := FALSE; id_quad_2 := FALSE; se_quad_2 := FALSE; sd_quad_2 := FALSE;
0118 END_IF
0119
0120
0121 (* ---------------------------------------------- Caso 10 - Caso de orientação dos Quadrados com reentrâncias ----------------------------------------------*)
0122
0123 IF bt_caso10 THEN            (* Botão "Caso 10" foi premido *)
0124
0125     conta_trans_pv := 0;
0126     conta_trans_nv := 0;
0127     FOR y := 0 TO 5 DO                          (* x- coluna ; y - linha *)
0128             retorna_nv := nv_find(1,y);        (* procura na coluna 1 a posição da transição horizontal positiva *)
0129             IF retorna_nv = -1 THEN
0130                 EXIT;
0131             ELSE
0132                 conta_trans_nv := retorna_nv;
0133             END_IF
0134             retorna_pv := pv_find(4,y);        (* procura na coluna 4 a posição da transição horizontal negativa *)
0135             IF retorna_pv = -1 THEN
0136                 EXIT;
0137             ELSE
0138                 conta_trans_pv := retorna_pv;
0139             END_IF
0140     END_FOR
0141
0142     conta_trans_ph := 0;
0143     conta_trans_nh := 0;
0144     FOR x := 0 TO 5 DO                          (* x- coluna ; y - linha *)
0145             retorna_ph := ph_find(1,x);        (* procura na linha 1 a posição da transição ascendente *)
0146             IF retorna_ph = -1 THEN
0147                 EXIT;
0148             ELSE
0149                 conta_trans_ph := retorna_ph;
0150             END_IF
0151             retorna_nh := nh_find(4,x);        (* procura na linha 4 a posição da transição descendente *)
0152             IF retorna_nh = -1 THEN
0153                 EXIT;
0154             ELSE
0155                 conta_trans_nh := retorna_nh;
0156             END_IF
0157     END_FOR
0158
0159 (* valores encontrado dizem respeito às ultimas transições daquele tipo, encontradas no padrão *)
0160     IF conta_trans_pv = 4 AND conta_trans_ph = 5 THEN
0161         ie_reent_2 := FALSE; se_reent_2 := FALSE; sd_reent_2 := FALSE; id_reent_2 := TRUE;
0162     END_IF
0163     IF conta_trans_nv = 2 AND conta_trans_nh = 1 THEN
0164         ie_reent_2 := FALSE; se_reent_2 := TRUE; sd_reent_2 := FALSE; id_reent_2 := FALSE;
0165     END_IF
0166     IF conta_trans_ph = 4 AND conta_trans_nv = 1 THEN
0167         ie_reent_2 := TRUE; se_reent_2 := FALSE; sd_reent_2 := FALSE; id_reent_2 := FALSE;
0168     END_IF
0169     IF conta_trans_nh = 2 AND conta_trans_pv = 5 THEN
0170         ie_reent_2 := FALSE; se_reent_2 := FALSE; sd_reent_2 := TRUE; id_reent_2 := FALSE;
0171     END_IF
0172
0173 ELSE
0174     ie_reent_2 := FALSE; id_reent_2 := FALSE; se_reent_2 := FALSE; sd_reent_2 := FALSE;
0175 END_IF
```

Figura A.70. Algoritmo Orientação_triangulos – parte 3

```
0001 FUNCTION Regressao : INT
0002 VAR_INPUT
0003 END_VAR
0004 VAR
0005     i: INT;
0006     med_x: LREAL;            (* med_x - faz a média de uns existentes por coluna*)
0007     soma_colunas: INT;
0008     soma_linhas: INT;
0009     med_y: LREAL;            (* med_y - faz a média de uns existentes por coluna*)
0010     sxx: LREAL;
0011     sxy: LREAL;
0012     declive: LREAL;
0013 END_VAR
0001
0002 (* ___________ Algoritmo 6 - Regressão ___________*)
0003
0004 (*------------------------------------------ Corpo geral do algoritmo ------------------------------------------------------------------*)
0005
0006 (* Calculo prévio de alguns valores necessários. Nota: y - conta_coluna;  x - conta_linha  *)
0007     soma_colunas := 0;
0008     FOR i := 0 TO amostras_coluna DO
0009         soma_colunas := soma_colunas + conta_coluna[i];        (* vai à posição i da varável global, vector de inteiros,*)
0010                                                                (* "conta_coluna" e faz o somatório dos valores do vector*)
0011     END_FOR
0012     med_y := soma_colunas/(amostras_coluna+1);                (* med_y - faz a média de uns existentes por coluna*)
0013
0014     soma_linhas := 0;
0015     FOR i := 0 TO amostras_linha DO
0016         soma_linhas := soma_linhas + conta_linha[i];          (* vai à posição i da varável global, vector de inteiros, *)
0017                                                               (* "conta_linha" e faz o somatório dos valores do vector*)
0018     END_FOR
0019     med_x := soma_linhas/(amostras_linha+1);                  (* med_x - faz a média de uns existentes por coluna*)
0020
0021     sxx := 0;
0022     FOR i := 0 TO amostras_linha DO
0023     sxx := sxx + (conta_linha[i] - med_x)*(conta_linha[i] - med_x);
0024     END_FOR
0025
0026     sxy := 0;
0027     FOR i := 0 TO amostras_coluna DO
0028     sxy := sxy + (conta_linha[i] - med_x)*(conta_coluna[i]-med_y);
0029     END_FOR
0030
0031     declive := sxx/sxy;
0032
0033 (*------------------------------------------ Fim da parte geral do programa ------------------------------------------------------------*)
0034
0035 (*-----------------------------------------caso 1 - identificaçã de triangulo ------------------------------------------------------------*)
0036
0037 IF bt_caso1 THEN              (* Botão "Caso 1" foi premido *)
0038
0039     IF declive >= 0.75 AND declive <= 1.5 THEN
0040         Regressao := 1;
0041     ELSE
0042         IF declive >= -1.5 AND declive <= -0.75 THEN
0043             Regressao := 0;
0044         ELSE
0045             IF declive > -0.75 AND declive < 0.75 THEN
0046                 alg_6_aux := TRUE;
0047             END_IF
0048         END_IF
0049     END_IF
0050
0051 END_IF
```

Figura A.71. Algoritmo Regressão – parte 1

```
0052
0053 (*---------------------------------------- caso 2 - identificaçã de quadrado furado ----------------------------------------*)
0054
0055 (*O declive será sempre zero mas a existencia do furo fará a linha de tendencia baixar*)
0056
0057 IF bt_caso2 THEN              (* Botão "Caso 2" foi premido *)
0058
0059     IF med_y  <= 5 AND med_y >= 4 THEN
0060         Regressao := 1;
0061     ELSE
0062         IF med_y  <= 6 AND med_y > 5.5 THEN
0063             Regressao := 0;
0064         ELSE
0065             IF med_y  <= 5.5 AND med_y  > 5OR med_y < 4 THEN
0066                 alg_6_aux := TRUE;
0067             END_IF
0068         END_IF
0069     END_IF
0070
0071 END_IF
0072
0073
0074 (* caso 3 - não aplicavel *)
0075 (* caso 4 - não aplicável *)
0076 (* caso 5 - não aplicável *)
0077 (* caso 6 - não aplicável *)
0078 (* caso 7 - não aplicável *)
```

Figura A.72. Algoritmo Regressão – parte 2

Anexo 4

Programação PLC

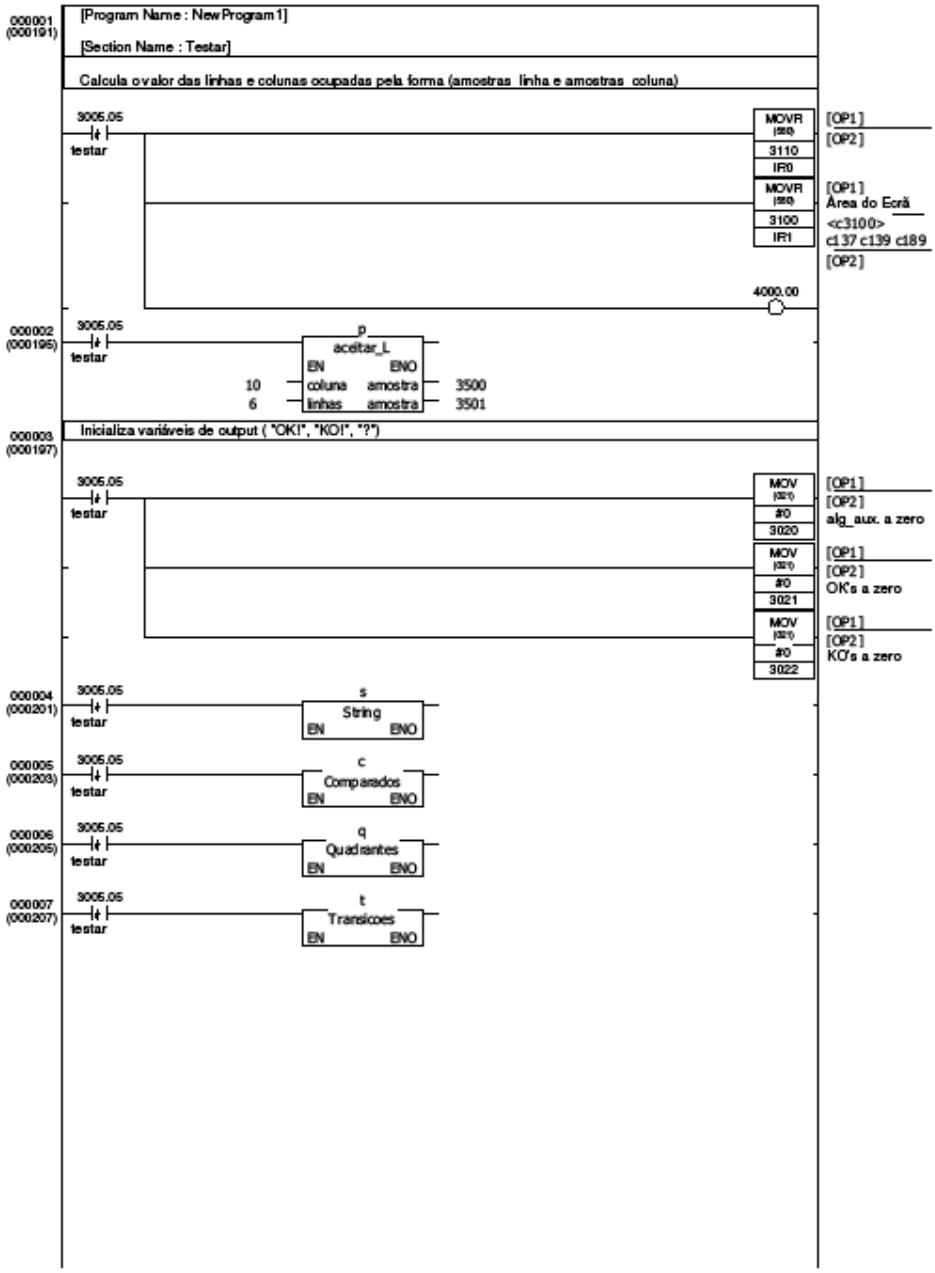


Figura A.73. Função "Testar" (Ladder)

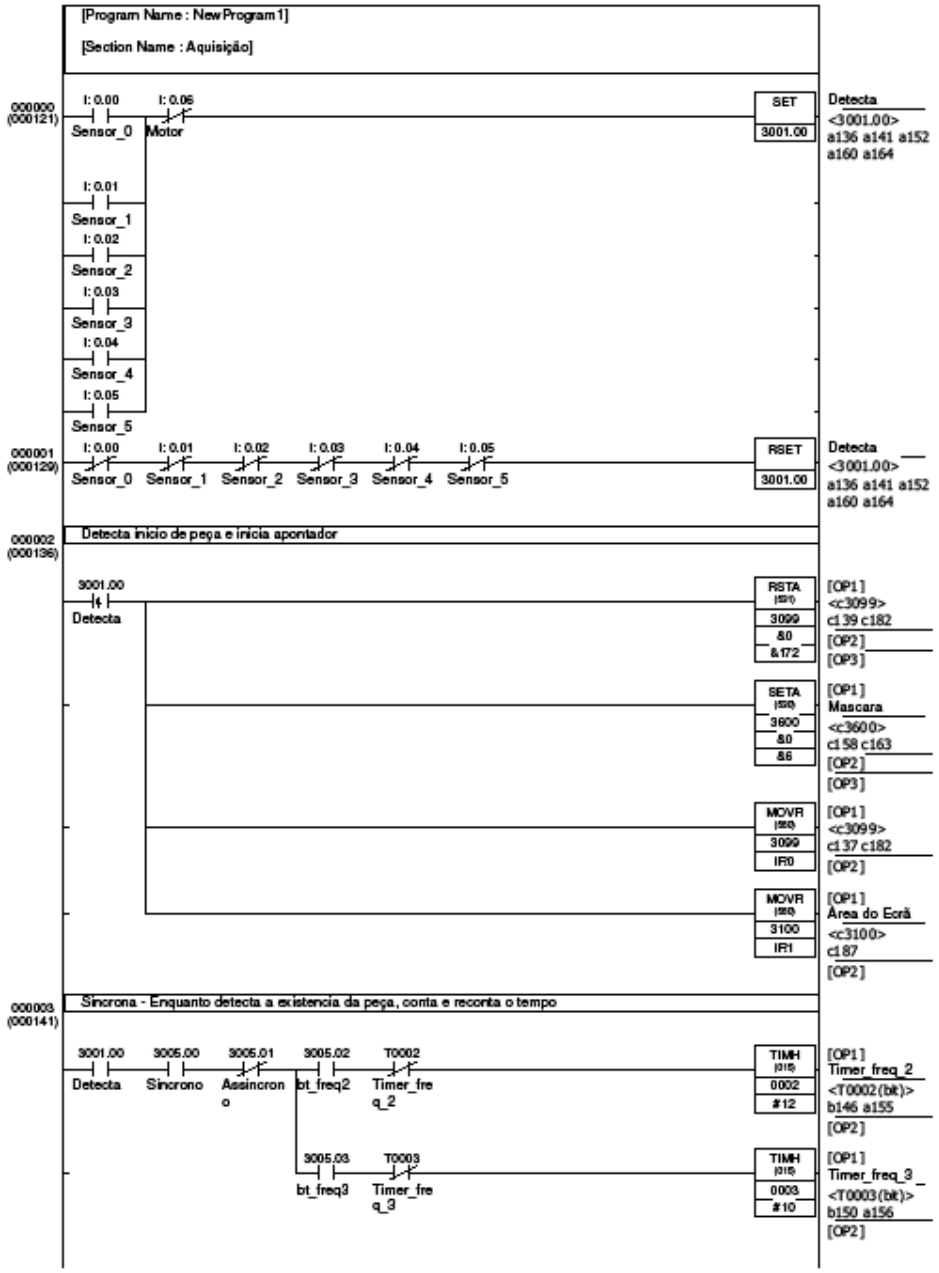


Figura A.74. Aquisição de dados síncrona e assíncrona – parte 1

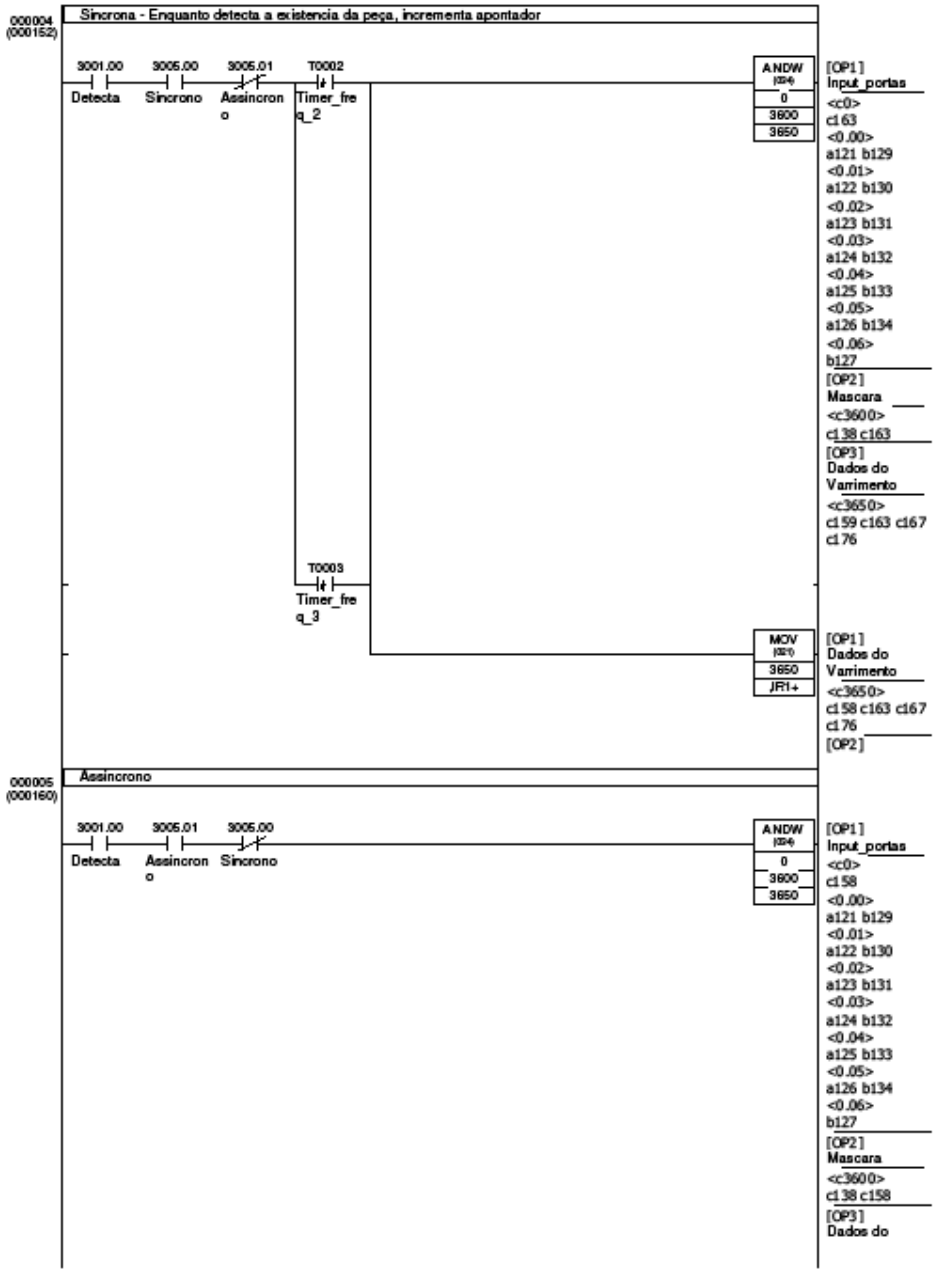


Figura A.75. Aquisição de dados síncrona e assíncrona – parte 2

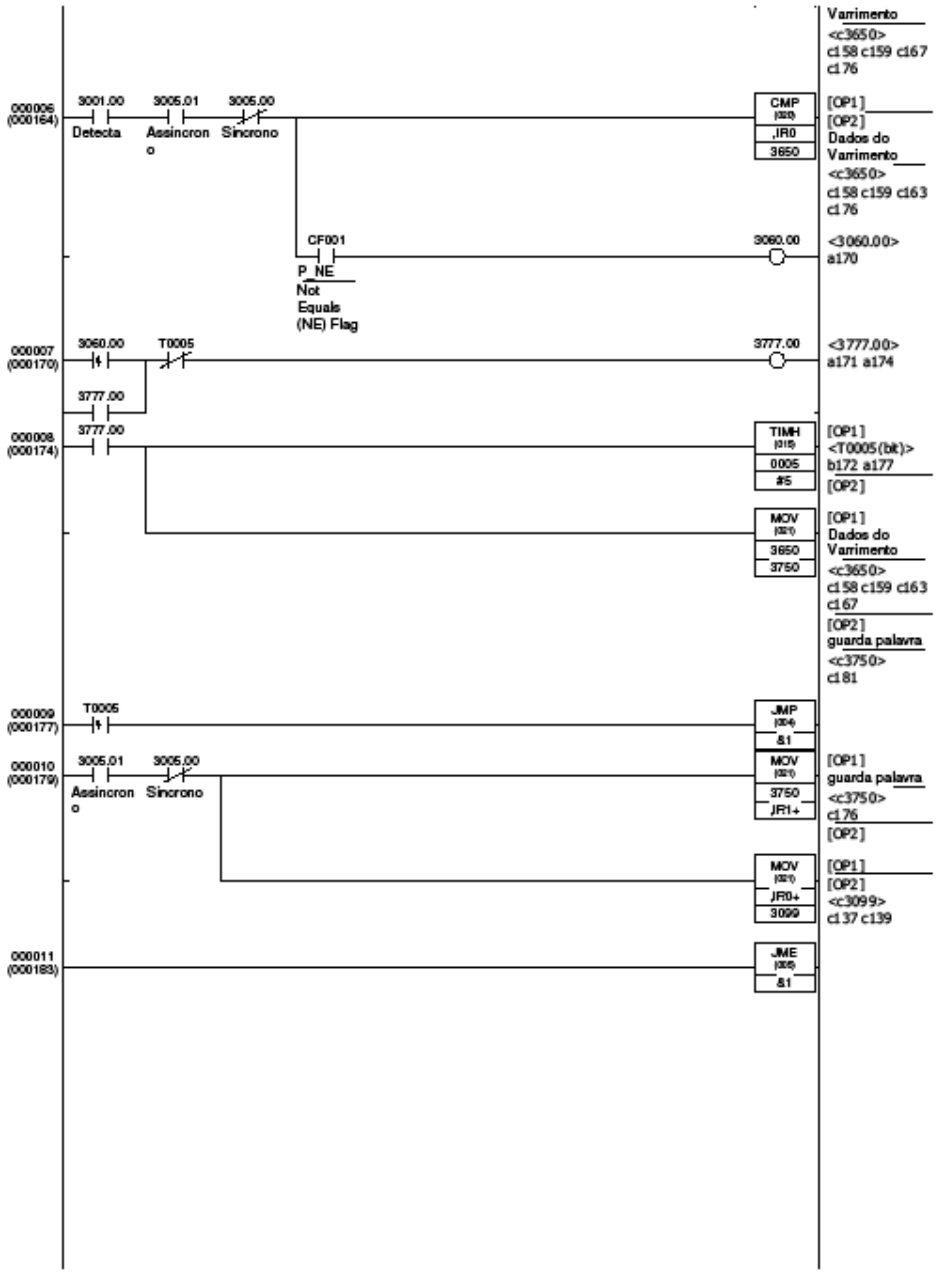


Figura A.76. Aquisição de dados síncrona e assíncrona – parte 3

Variable Type	Name	Data Type	Retained	AT	Initial Value	Comment
Inputs	EN	BOOL	No		FALSE	Controls execution of the Function Block.
Inputs	coluna	INT	No		0	
Inputs	linhas	INT	No		0	
Outputs	ENO	BOOL	No		FALSE	Indicates successful execution of the Function Block.
Outputs	amostra_coluna	INT	No		0	
Outputs	amostra_linha	INT	No		0	
Internals	teste	BOOL	No		FALSE	
Internals	mascara	WORD	No		0	
Internals	posicao	WORD	No		0	
Internals	contador	INT	No		0	
Internals	ciclos_c	INT	No		0	
Internals	subtraido_c	INT	No		0	
Internals	temp	WORD	No		0	
Internals	ciclos_l	INT	No		0	
Internals	masc_shift	WORD	No		0	
Internals	temp_2	WORD	No		0	
Internals	teste_2	BOOL	No		FALSE	
Internals	subtraido_l	INT	No		0	
Internals	contador_c	INT	No		0	

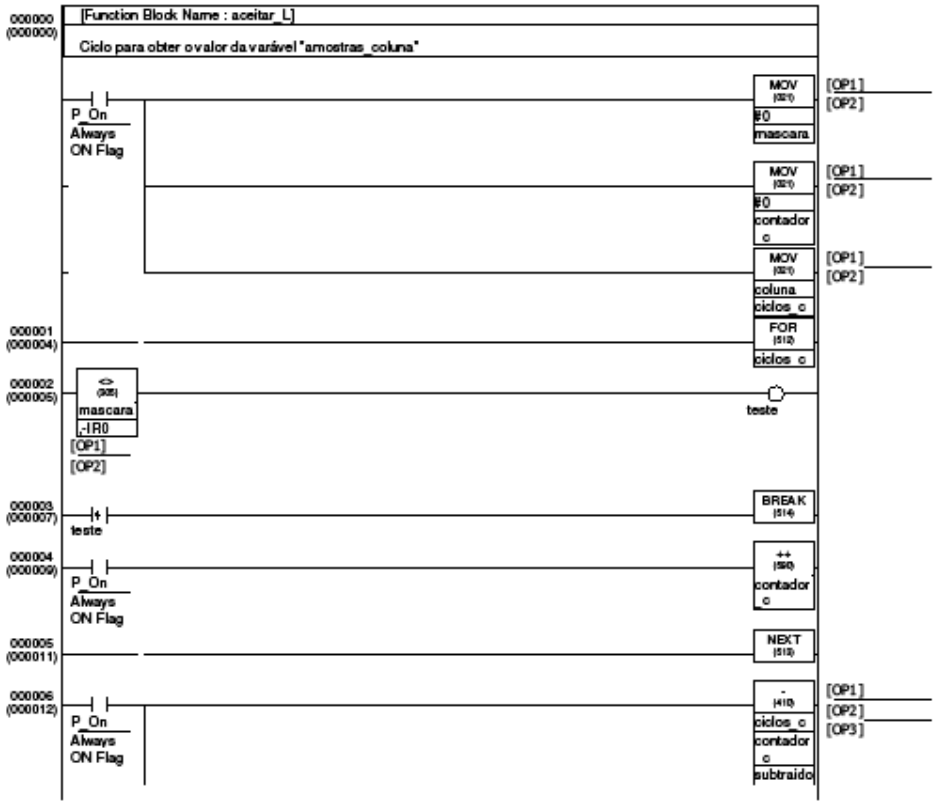


Figura A.77. Função aceitar – parte 1

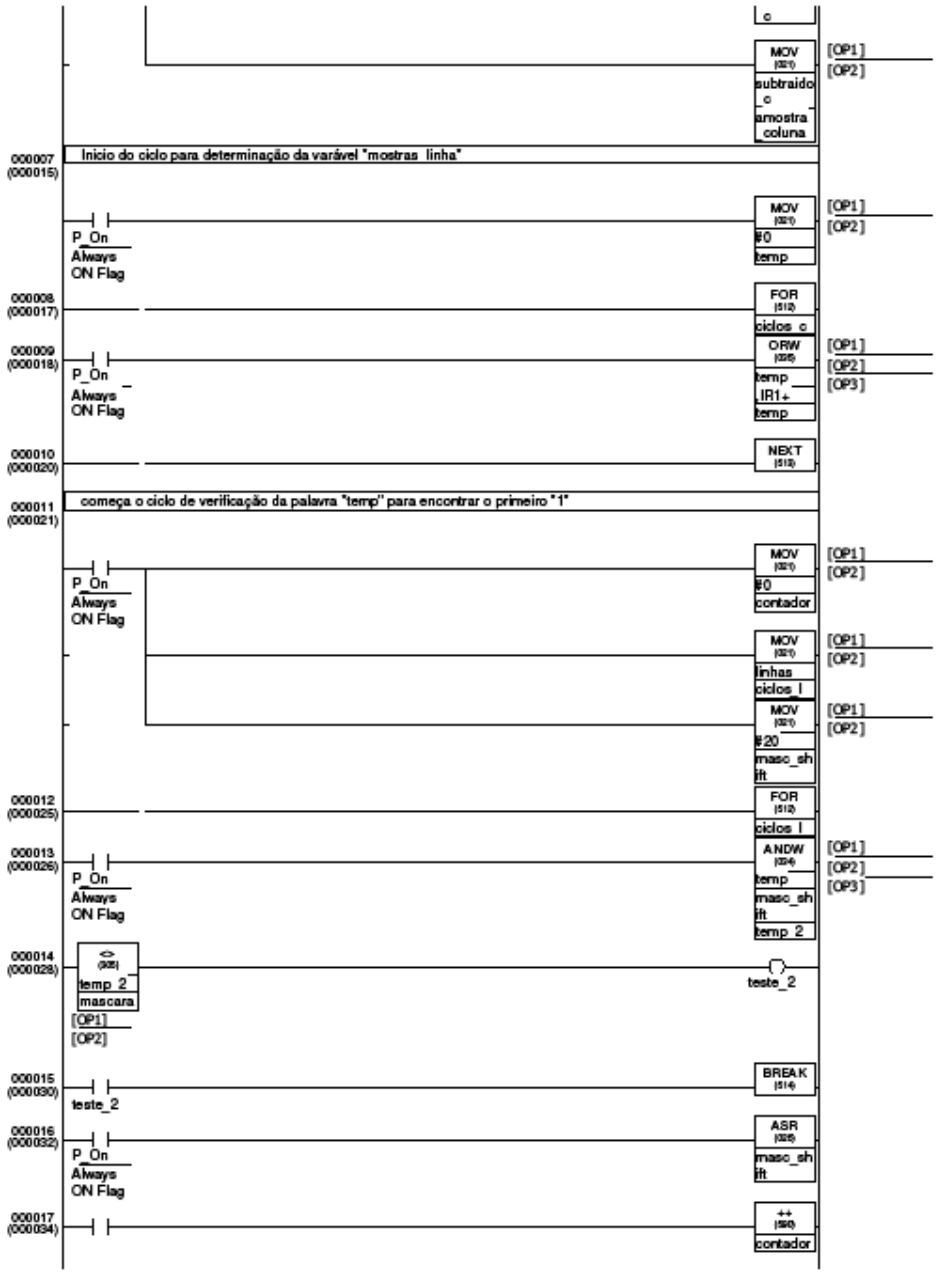


Figura A.78. Função aceitar – parte 2

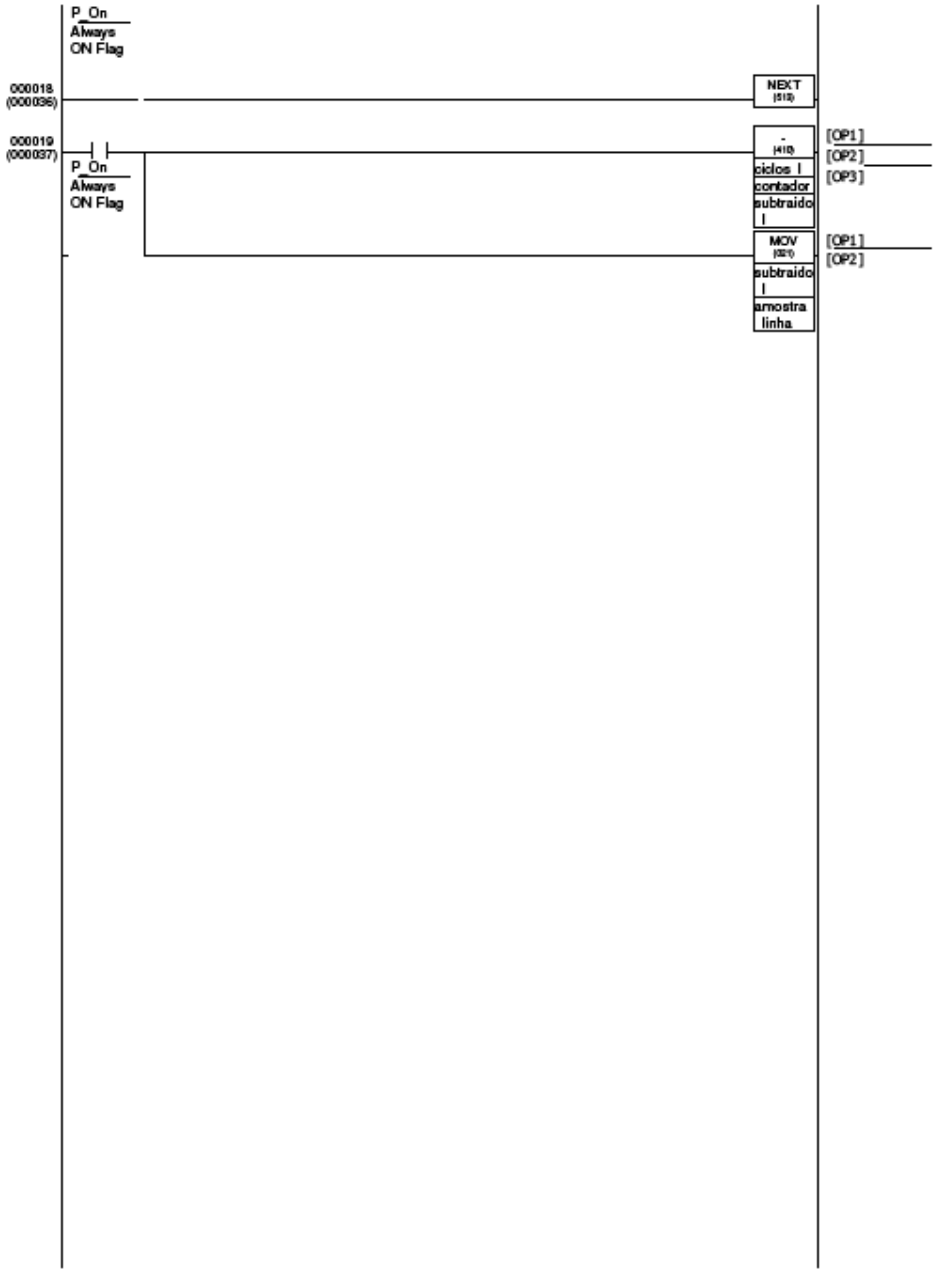


Figura A.79. Função aceitar – parte 3