Universidade de Aveiro
2023

João Duarte
Bacalhau Pombeiro

Inspeção automática da presença de furos roscados em peças maquinadas

Automatic inspection for the presence of threaded holes in machined parts

Universidade de Aveiro
2023

João Duarte
Bacalhau Pombeiro

Inspeção automática da presença de furos roscados em peças maquinadas

Automatic inspection for the presence of threaded holes in machined parts

Trabalho de Projeto apresentado à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de José Paulo Oliveira dos Santos, Professor Auxiliar, do Departamento de Engenharia Mecânica da Universidade de Aveiro.

Esta dissertação teve o apoio dos projetos UIDB/00481/2020 e UIDP/00481/2020 - Fundação para a Ciência e a Tecnologia; e CENTRO-01-0145 FEDER-022083 - Programa Operacional Regional do Centro (Centro2020), através do Portugal 2020 e do Fundo Europeu de Desenvolvimento Regional.

O júri / The jury

Presidente / President	**Prof. Doutor António Manuel de Bastos Pereira** Professor Associado C/ Agregação da Universidade de Aveiro
Vogais / Committee	**Doutor José Nuno Panelas Nunes Lau** Professor Associado da *Universidade de Aveiro*
	Prof. Doutor José Paulo Oliveira dos Santos Professor Auxiliar da Universidade de Aveiro (orientador principal)

Agradecimentos / Acknowledgements

Agradeço à Universidade de Aveiro, instituição que, durante todo o meu percurso académico, permitiu que obtivesse a formação adequada e necessária à elaboração desta dissertação. Ao meu orientador, Professor Doutor José Paulo de Oliveira Santos, pelo acompanhamento, disponibilidade e dedicação durante a realização deste projeto. Agradeço também à Renault Cacia S.A., pela oportunidade que me foi dada e que eu espero ter correspondido às espetativas que depositaram em mim, bem como os meios e instalações cedidas para a realização do projeto. A todos os elementos da empresa associados ao projeto, em especial ao Engenheiro Diogo Ramos, ao Engenheiro João Gonçalves, ao Engenheiro Veríssimo e ao Engenheiro Jaime, pela disponibilidade, profissionalismo e acompanhamento. À minha família, em especial aos meus pais, pela cooperação, paciência, ajuda e orientação nas opções pessoais a tomar. Aos meus colegas e amigos, pelo companheirismo, motivação, discussão e ajuda nas dificuldades, bem como pelos momentos de diversão e descontração. Por fim resta-me agradecer a todos por acreditarem no meu percurso académico.

Keywords Artificial Vision, Machine Learning, Quality Control, Image Processing, Support Vector Machine

Abstract Currently, the identification of anomalies at the level of threaded holes in parts machined on the industrial machining lines of Renault Cacia, S.A. It is carried out through visual control that is unreliable. This being the performed manually, losses are associated with the level of activities NVA (non-value added) and human error. Faced with this type of troll, consequently arise downstream of this machining process, parts with unthreaded holes, not identified by manual control. Faced with this type of anomaly, consequently, parts with non-threaded holes appear in the process, not identified by manual control.

The objective of this study is to develop and implement a monitoring system visualization that allow obtaining images of threaded holes in the parts machined in an industrial environment, in such a way as to enable identification and detection of absences of these threaded holes precisely.

The target objects of the study are 3-cylinder crankcases, produced at Renault Cacia, whose quality inspection is currently carried out manually. The main objective will be to resort to an algorithm that has the capacity to inspect the machined parts, using a database to save the information from the inspection in an industrial environment, in addition to having a ability to adapt to different lighting and, in turn, notify the operator when detecting any anomaly. Throughout the document, all the work carried out is described and explained, since the construction of the visualization system for image acquisition, to the development of the algorithms carried out for the classification of crankcases check obtained. The developed algorithms, with based on Halcon software, were Template Matching Base-Shaped and Support Vector Machine so that later, with the results obtained, can make a comparison and come to the conclusion about the best algorithm to be applied. The tests carried out allowed us to conclude that the de- developed, focusing on 200 images both for the piece "OK" as well as for the "NOK" one, it was able to correctly detect all the parameters of the parts intended for the two algorithms used, with an F1 score of around 96% for both.

The system ended up not being implemented on the line due to unforeseen events. occurred during the projection period for the implementation of the system. This fact was also due to the decision taken by the company, already in the final part of that period, to opt for the implementation of a system multifunctional much more comprehensive.

Palavras-chave Visão Artificial, Aprendizagem Automática, Controlo de Qualidade, Processo de Imagem, Máquina de Vetores Suporte

Resumo

Atualmente a identificação de anomalias ao nível de furos roscados das peças maquinadas nas linhas industriais de maquinação da Renault Cacia, S.A. é realizado através de um controlo visual não automatizado. Sendo este controlo feito de forma manual, estão associadas perdas ao nível de atividades NVA (não valor acrescentado) e de erro humano. Perante este tipo de controlo, surgem, consequentemente, a jusante deste processo de maquinação, peças com furos não roscados, não identificadas pelo controlo manual. Para fazer face a estas anomalias, surge então a necessidade de automatizar essa verificação, deixando esta de depender de controlo humano.

O objectivo deste estudo foi desenvolver e implementar um sistema de visualização que permita a obtenção de imagens de furos roscados nas peças maquinadas em ambiente industrial, de tal forma que possibilite a identificação e a deteção de eventuais ausências destes furos roscados com maior rigor.

Os objetos alvo do estudo são cárteres de 3 cilindros, produzidos na Renault Cacia, cuja inspeção de qualidade é atualmente efetuada de forma manual. O objectivo principal será recorrer a um algoritmo que possua a capacidade de inspecionar as peças maquinadas, recorrendo a uma base de dados com imagens pré-captadas em ambiente industrial, para além de possuir uma capacidade de se adaptar a uma iluminação diferenciada e, por sua vez, notificar o operador aquando da detecção de eventual anomalia.

O conteúdo do documento consiste na descrição e explicação detalhada de todo o trabalho desenvolvido ao longo do período concedido para o efeito, desde a construção do sistema de visualização para obtenção de imagens, até ao desenvolvimento dos algoritmos levados a cabo com vista à classificação dos cárteres e dos seus resultados. Os algoritmos utilizados, com base no software Halcon, foram o Template Matching Shape-Based e o Support Vector Machine para que, posteriormente, com os resultados obtidos, se possa fazer uma comparação e chegar à conclusão sobre o melhor algoritmo a ser aplicado. Os ensaios realizados permitiram concluir que o sistema desenvolvido, com incidência em 200 imagens tanto para a peça "conforme" como também para a "não conforme", conseguiu detectar grande parte dos parâmetros das peças pretendidos para os dois algoritmos utilizados, com uma pontuação F1 cerca de 96% para ambos.

O sistema acabou por não ser implementado na linha devido a imprevistos ocorridos durante o período de estágio para a implementação do sistema. Tal facto também ficou a dever-se à decisão tomada pela empresa, já na parte final do referido período, em optar pela implementação de um sistema multifuncional bastante mais abrangente, apesar desta opção tomada pela RENAULT integrar a funcionalidade e objectivo a implementar na linha, aos quais visava o projecto aqui descrito.

Índice

Lista de Tabelas

Intentionally blank page.

Lista de Figuras

Intentionally blank page.

Lista de Listagens

Intentionally blank page.

Lista de Acrónimos

CAD Computer-Aided Design.

CCD Charge-Coupled Device.

I/O Input/Output.

IP Internet Protocol.

LED Light-Emitting Diode.

ML Machine Learning.

PLC Programmable Logic Controller.

SVA Sistema de Visão Artificial.

SVM Support Vector Machine.

TCP Transmission Control Protocol.

Intentionally blank page.

Capítulo 1

Introdução

Neste capítulo será feita uma breve apresentação do enquadramento ao tema da tese. Será também fornecida informação sobre o processo produtivo e será apresentado de forma explicita o problema atualmente associado a uma das atuais linhas de produção. Será, ainda, descrita a razão/motivação que levou ao seu desenvolvimento, qual o objetivo a implementar e por fim a descrição da estrutura de todo o documento.

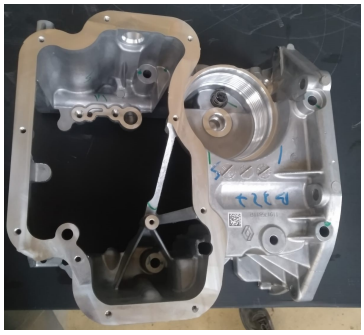
1.1 Enquadramento

A Renault CACIA é uma empresa que produz peças e órgãos mecânicos para motores e veículos automóveis. Para melhorar a precisão na inspeção dessas mesmas peças e órgãos, o recurso a sistemas de inspeção automática revela-se cada vez mais indispensável, uma vez que são estes que garantem o aumento do rigor, da consistência e da confiança da inspeção, reduzindo ou, até mesmo, eliminando assim a possibilidade de erro humano no sistema.

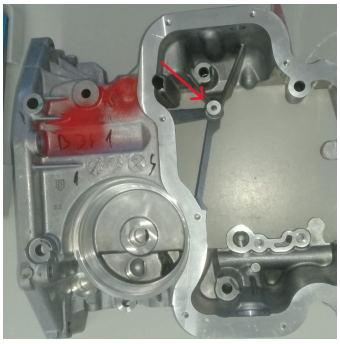
Atualmente a produção é feita com recurso à realização de um controlo visual para identificação de anomalias ao nível da presença de roscas das peças maquinadas, nomeadamente os cárteres intermédios localizados no atelier 6 da linha 3. Como o controlo é feito de forma manual, estão associadas perdas ao nível de atividades NVA (não valor acrescentado) e sujeitas à eventualidade de erro humano. Pelo que, consequentemente, é frequente o surgimento a jusante deste processo de maquinação, peças sem roscas não identificadas pelo controlo manual.

A deteção automática de falhas usando *Machine Learning* tornou-se uma área de pesquisa empolgante e promissora. Isto porque é uma forma precisa e oportuna de gerir e classificar com o mínimo de esforço humano. O processo de deteção de presença-ausência tem recebido uma atenção considerável por parte da indústria automóvel, uma vez que as características ausentes têm elevados custos de produção e de retrabalho. A redução do número de N-OKs na inspeção dos produtos, veio aumentar a produtividade, permitindo assim às empresas uma melhor e maior produção com os mesmos recursos. Contudo, para que este fator se revelasse uma realidade, seria primordial uma redução significativa das operações na linha de produção.

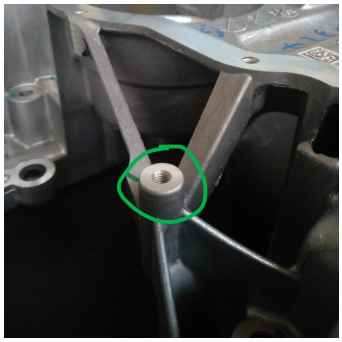
A figura 1.1 apresenta a comparação de uma peça maquinada, nomeadamente um cárter de 3 cilindros produzido na empresa Renault CACIA com os dois furos M6 a serem inspecionados, sendo uma com defeito e outra sem defeito. De facto, é difícil visualizar com rigor se os furos M6 apresentam rosca através do olho humano.



(a) Cárter sem defeitos

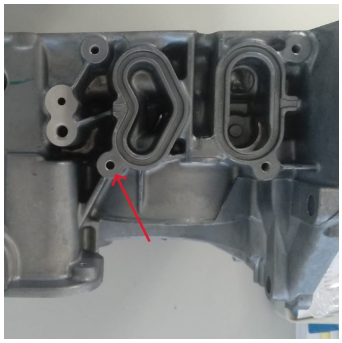


(b) Cárter defeituoso



(c) Furo M6 roscado

(d) Furo M6 não roscado



(e) Vista lateral do cárter com defeito

(f) Furo M6 não roscado na zona lateral do cárter

Figura 1.1: Peças maquinadas com e sem defeitos

Durante o desenvolvimento do projeto, o sistema terá de ser capaz de se adaptar à diferente iluminação passível de acontecer dentro do ambiente industrial e, por sua vez, notificar o operador o nível aceitável de erro.

1.2 Processo produtivo

Na Renault CACIA são produzidos dois tipos de cárteres, os de três cilindros e os de quatro cilindros. O cárter a ser inspecionado e que se insere no projeto é um cárter de três cilindros.

Existem diversas operações de produção do cárter, as quais são apresentadas por ordem cronológica na figura seguinte:

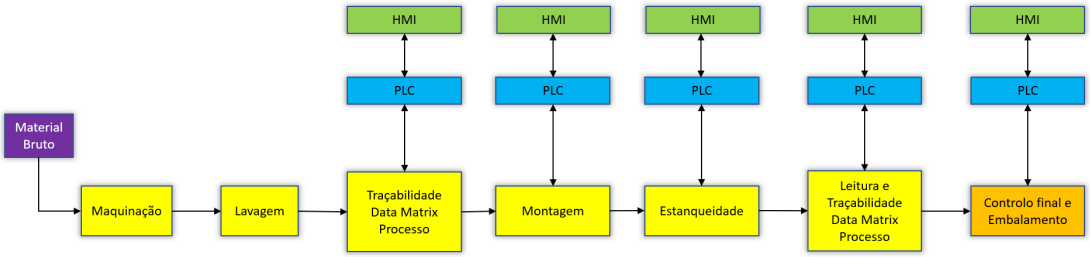


Figura 1.2: Controlo do processo produtivo na linha dos cárteres de 3 cilindros

O controlo existente no processo de produção consiste no processamento das peças sem qualquer tipo de anomalia. Os vários blocos que constituem o processo, são dotados por um sistema de comunicação, utilizada pela empresa (AGI TRACE), em que parte desse sistema constitui uma forma de transmitir à fase seguinte a mensagem de que o cárter em produção passou pela etapa anterior sem problemas.

As várias etapas do processo são as seguintes:

- **Abastecimento bruto** — O material bruto em alumínio proveniente de outros fornecedores da empresa é introduzido no local de início para a primeira operação.

Figura 1.3: Abastecimento bruto dos cárteres de 3 cilindros

- **Maquinação** – Primeira operação a realizar, onde passará por dois processos, que são a maquinação vertical e horizontal. Estes processos apenas diferem na mudança

de posição da peça com as diferentes operações de remoção de material, furação e roscagem.

Figura 1.4: Maquinação

- **Inspeção múltipla** – Esta fase consiste na inspeção do último cárter feito na etapa de maquinação do turno. O operador utiliza as ferramentas de calibre para a medição dos furos e roscas, verificando a existência de alguma anomalia que tenha ocorrido no processo de maquinação e verifica a presença de alguma porosidade ou outros tipos de defeitos. Caso seja detetada pelo operador algum tipo de anomalia no último cárter, todos os cárteres que passaram nesse turno, terão como destino a sucata para uma verificação.

- **Lavagem** - Operação na qual se procede à remoção de restos de material indesejado existente na peça para não perturbar as operações seguintes.

Figura 1.5: Processo de lavagem

- **Traçabilidade Data Matrix Processo** - Operação onde a peça será marcada por Data Matrix dos processos que o cárter sofreu até então e de que irá ser alterada nas etapas seguintes (Figura 1.8). A partir desta etapa, o cárter só poderá passar para as etapas seguintes, se a etapa anterior tiver sido realizada recentemente, ou seja, no final de cada etapa é inserida uma mensagem na Data Matrix do cárter de que a etapa foi realizada e, ao mesmo tempo, essa mensagem é passada para máquina da etapa seguinte. Caso a máquina da etapa seguinte não receba a mensagem nesse sentido, o cárter será rejeitado. Este tipo de tratamento é essencial para que o cárter contenha um registo histórico dos processos que realizou.

Figura 1.6: Traçabilidade Data Matrix Processo

- **Montagem** – Operação em que o cárter será montado juntamente com uma esfera, tampão 22, válvula do permutador, tampão 16 e válvula do filtro de óleo, por esta ordem (Figura 1.7).
- **Estanqueidade** - Operação de teste onde será feita um controlo de qualidade, quer da maquinação, quer da montagem, sendo injetado uma massa de ar e sendo medida a pressão do mesmo após entrada na peça (Figura 1.7). Esta inspeção é feita através de um sistema de *poka-yoke*, ou seja, um mecanismo colocado em prática e que tem como objetivo inibir, corrigir ou destacar um erro humano à medida que ele ocorre.

- **Montagem anilhas** - Etapa onde se procede à montagem de anilhas de forma a fixar a placa de anti-emulsão.
- **Traçabilidade Data Matrix Produto** - Etapa onde é feita a traçabilidade da Data Matrix Produto (Figura 1.8), ou seja, onde é inserido todo o registo histórico dos processos que o cárter sofreu até então. Trata-se de uma etapa bastante importante perante uma eventual reclamação do cliente, no sentido de que o produto foi entregue com anomalias. Nesse caso, temos acesso ao registo histórico das etapas realizadas, o que permite verificar qual a anomalia e corrigí-la, caso se confirme.

Figura 1.7: Processos de montagem (direita) e estanqueidade (esquerda) do cárter

Figura 1.8: Montagem de anilhas e traçabilidade Data Matrix Produto

- **Controlo final/Embalamento** - Etapa final onde o operador faz uma inspeção rápida do cárter no caso de alguma existência de anomalia das últimas etapas referidas, Faz o registo deste e introduz no carrinho dos cárteres finalizados (Figura 1.9).

1.2.1 Problema

Perante as etapas descritas na secção anterior, existe a probabilidade dos dois furos M6 apresentarem algumas anomalias, designadamente ao nível da presença de rosca. Tal facto fica a dever-se a uma possível falha na elaboração desses furos durante a etapa "Operação de Maquinação". Os referidos furos encontram-se descritos na face 200 (vista principal) denominada por furo 240 e na face 300 (vista lateral) denominada por furo 312.

Quanto às causas associadas aos referidos incidentes, podemos apontar: quebra de ferramentas; avaria da máquina; acidente com o robot; erro humano; queda da peça; que-

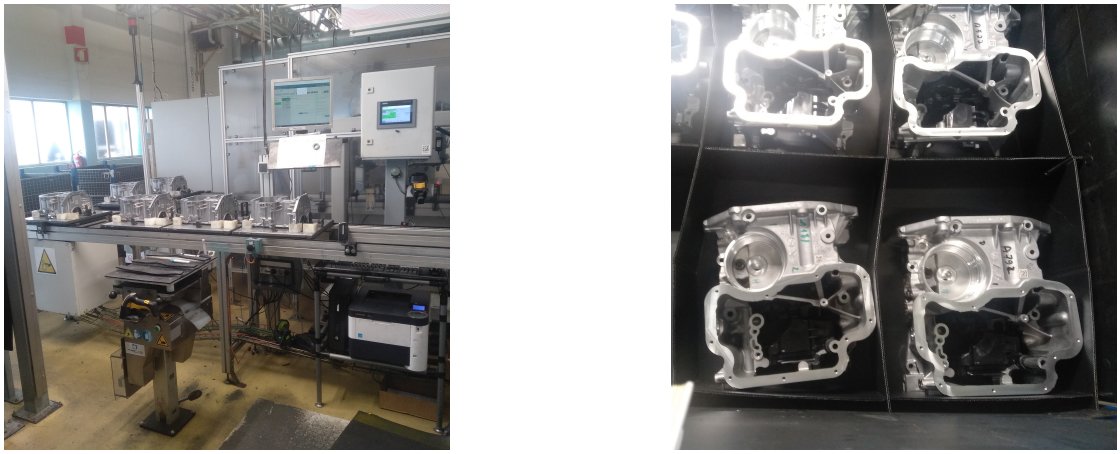


Figura 1.9: Controlo final (esquerda) e um carrinho com um lote de cárteres finalizados (direita)

bra da peça, ou peça mal posicionada. A frequência com que ocorrem estes problemas na linha de produção é grande. Percebe-se uma preocupante tendência de pouca variação nas causas que os desencadeiam. Uma abordagem mais proativa, voltada à identificação e correção desses fatores recorrentes, tem vindo a aumentar, tornando-se essencial para melhorar a eficiência, a segurança e a qualidade do processo produtivo como um todo.

No gráfico da figura 1.10, com auxílio dos dados fornecidos pela empresa no que refere às quantidades de peças produzidas em determinado espaço de tempo, é possível constatar que nunca foi conseguido o objetivo do número de peças estimadas, muito por culpa dos incidentes, uma vez que estes, a ocorrerem, implicam atraso no tempo de produção.

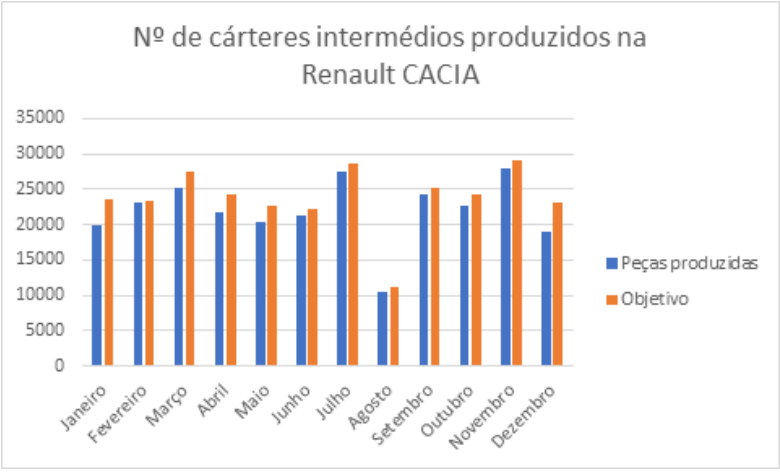


Figura 1.10: Gráfico do número de cárteres produzidos na empresa por mês e o seu objetivo

O caso específico da inexistência de rosca nos furos do cárter atrás identificados,

deve-se, essencialmente, a anomalia na broca e ocorre durante a etapa da maquinação. Assim sendo, e atendendo a que o processo de roscagem é feito em vários furos, a prática diz que basta apenas fazer o controlo junto da última rosca realizada. Ou seja, será apenas provável a anomalia em qualquer rosca da peça, caso se verifique a inexistência de rosca no último furo. Perante este facto, apesar da máquina de operação dar o aviso do incidente que ocorreu na maquinação, tal não impede a passagem do cárter, ainda que defeituoso, pois a falha não se deve à falta de aviso da máquina, mas sim à falha do operador que, por distração, a coloca no processo seguinte sem que a anomalia da falha nas roscas seja detetada.

1.3 Motivação

Tendo em conta que na inspeção feita pelo olho humano a possibilidade de não deteção de erro cresce com o passar do tempo despendido pelo operador em consequência do cansaço, grande parte das peças maquinadas poderão ser aprovadas para a secção seguinte, mesmo que estas apresentem defeitos. O operário possui também outras tarefas de inspeção para além da do furo roscado, como por exemplo a inspeção das ferramentas da máquina se encontram aptas para continuar o processo de maquinação ou então a peça possui algum tipo de defeito de fabrico por parte da empresa externa. Portanto o desenvolvimento de um sistema autónomo na deteção da presença de furos roscados M6 é importante e altamente desejável, pois poderá traduzir-se na redução dos custos de produção, designadamente na redução de mão de obra, e, por sua vez, uma melhoria no rigor e na eficácia da inspeção às referidas peças.

1.4 Objetivo

O objetivo principal deste trabalho passa pelo desenvolvimento e implementação de um sistema de visão que permita a aquisição de imagens dos furos nas peças maquinadas, produzidas na Renault Cacia, e que identifique a presença, ou não, de rosca. Para isso, será necessário atingir as seguintes tarefas:

- Desenvolver um algoritmo de *Machine Learning* capaz de classificar a conformidade da peça se apresenta furos roscados, recorrendo a uma base de dados com as imagens da região do furo captadas em ambiente industrial.
- O sistema terá de ter a capacidade de se adaptar aos diferentes graus de luminosidade possíveis de se verificarem no interior do ambiente industrial.
- Notificar o operador a existência de eventuais anomalias da peça, nomeadamente a não presença de rosca.

1.5 Organização do documento

O presente documento encontra-se divido em 6 capítulos.

O primeiro serve de introdução e aborda o que a motivou o desenvolvi mento do projeto de dissertação de Mestrado, o seu enquadramento no tema escolhido e o objetivo.

Para uma melhor perceção dos assuntos abordados no segundo capítulo, este irá pautar-se pela apresentação do estado da arte, no qual serão demonstrados alguns conceitos fundamentais do tema, desde a visão por computador até às redes neuronais convolucionais. Também serão analisados alguns trabalhos realizados por outros autores relacionadas com o presente documento.

No terceiro capítulo será abordada a solução proposta, ao mesmo tempo que é realizado um estudo na arquitetura, quer no *hardware*, quer no software, sendo que a proposta de solução se dirige no sentido de resolver o problema inicial.

A apresentação da implementação do projeto terá lugar no quarto capítulo. Será, então, feita a descrição desde a concepção de um protótipo de inspeção, até às implementações finais, designadamente a solução comunicação com a câmara, bem como a solução que consiste na inspeção visual com os algoritmos desenvolvidos.

No quinto capítulo serão documentados os resultados provenientes das soluções desenvolvidas no projeto.

Finalmente, no sexto e último capítulo serão abordadas as considerações gerais consideradas relevantes, designadamente no que toca à execução e implementação final do projeto, bem como eventuais beneficiações a desenvolver no futuro.

Intentionally blank page.

Capítulo 2

Estado da arte

No presente capítulo terá lugar a introdução de alguns conceitos teóricos relevantes, com vista a garantir uma boa prática na realização da solução proposta, tais como a estrutura de um sistema de visão artificial e *Machine Learning* (ML). Por fim, é feita uma breve análise de trabalhos desenvolvidos por outros autores com objetivos semelhantes à presente dissertação.

2.1 Contextualização da visão artificial na indústria

Irão ser esclarecidos os conceitos básicos de um sistema de visão artificial e uma breve apresentação na qual se insere o *Machine Learning*, especialmente para problemas de classificação e análise de imagens.

2.1.1 Estrutura de um Sistema de Visão Artificial

Um Sistema de Visão Artificial (SVA) é descrito como um sistema computorizado capaz de adquirir, processar e interpretar imagens correspondentes a cenas reais. A figura a seguir (Figura 2.1) demonstra esquematicamente um diagrama de blocos de um SVA.

De uma forma muito breve, importa referir que, inicialmente, é necessário dominar o problema em causa no que confere à sua existência e às causas a ele associadas. Para tal, o primeiro passo a dar será a aquisição de imagens, sendo necessário um sensor, que converta a informação óptica em sinal elétrico. E, por sua vez, um scanner, que transformará a imagem analógica em imagem digital. Dever-se-á ter em conta o tipo de sensor a ser utilizado, assim como o conjunto de lentes, as condições de iluminação e o número de níveis de tom de cinza da imagem digitalizada.

O pré-processamento tem como função aprimorar a qualidade da imagem, uma vez que a imagem resultante do passo anterior pode apresentar algumas imperfeições como a presença de *pixels* ruidosos, contraste ou brilho inadequado.

A segmentação tem como tarefa dividir uma imagem em unidades significativas, ou seja, nos objetos de interesse que a compõem, sendo que a etapa seguinte - Extração de Características das Imagens, resulta da segmentação através de regiões, dos quais são

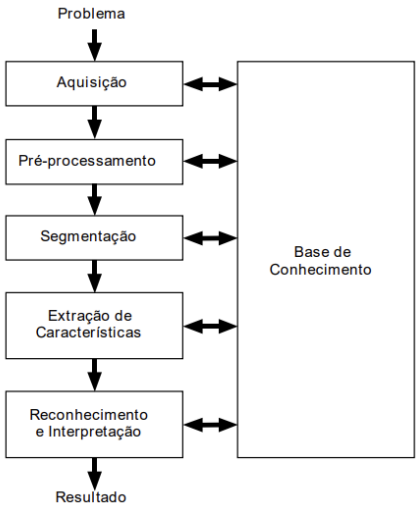


Figura 2.1: Principais etapas de um sistema de visão artifical. Fonte: [3]

representados por uma estrutura de dados adequado ao algoritmo de reconhecimento. Os descritores atrás referidos têm capacidade de apresentar várias métricas diferentes e conseguem distinguir diferentes objetos, por exemplo, a área que determina o número total de *pixels* da região.

Por fim, temos o reconhecimento e interpretação, a última etapa do sistema, que, tal como o nome indica, atribui um rótulo a um objeto, baseado nas características que foram traduzidas pelos seus descritores. Acresce ter em conta que todas estas etapas envolvem um conhecimento aprofundado sobre o problema a solucionar, dependendo aquelas da complexidade do problema [3].

2.1.2 Elementos de um sistema de visão por computador

Imagem digital

A imagem digital pode ser vista como uma matriz, cujas linhas e colunas identificam um ponto na imagem caraterizada como um *pixel*. Um *pixel* significa elemento de imagem, cujo valor corresponde o nível de cinza da imagem naquele ponto. A figura 2.2 apresenta uma matriz, que corresponde a uma foto, com uma intensidade de 86 e uma resolução de 8x12 com o *pixel* da posição [2,8].

Entre muitos outros, há modelos de cores, tais como o RGB, o CMYK,, que possuem mais do que uma componente matricial. No que concerne a este projeto, serão focadas imagens monocromáticas, que são discretizadas por apenas uma componente matricial de intensidades. O modelo é definido pela característica de que cada pixel da imagem é processado no intervalo [0,255], ou seja uma escala de cinzentos e o nível de intensidade é de 8 bits, isto é, 2^8=256 níveis, sendo o valor 0 como resultado preto e o 255 como branco [4].

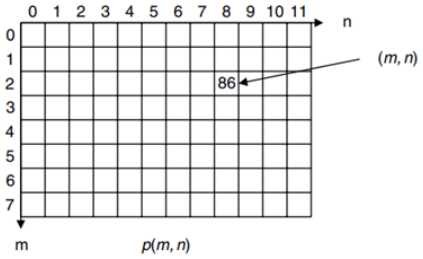


Figura 2.2: Imagem discretizada no *pixel* da posição [2,8]. Fonte: [4]

Binarização e Histograma

O histograma de uma imagem é nada mais do que um conjunto de números, indicando a porcentagem de *pixels* naquela imagem, que representa um determinado nível de cinza. Através da visualização do histograma é possível obter uma indicação da sua qualidade ao nível de contraste e brilho.

O princípio da binarização ou *threshholding* consiste na bipartição do histograma da imagem, em que os valores de *pixels* são convertidos para 0 ou 1, consoante o tom de cinza seja maior ou igual a um determinado valor limiar (T) que resulta em branco (1) e os restantes em preto (0). Na figura 2.3 podemos observar um exemplo da distribuição dos níveis de cinzento dos *pixels* por cada nível de intensidade da imagem do histograma onde se faz a escolha do valor de limiar na binarização da imagem [3].

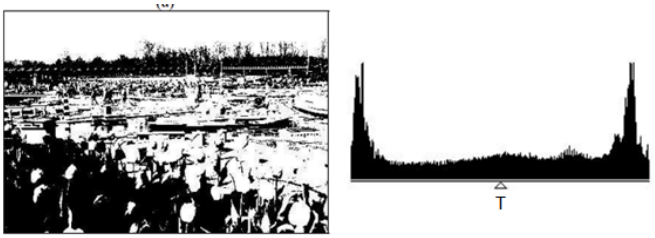


Figura 2.3: Efeito da escolha do limiar (T) na binarização de uma imagem. Fonte: [3]

Template Matching

O *template matching* é uma técnica bastante usada no processamento de imagem. É um método que determina os componentes de uma figura que corresponde a um modelo predefinido. A sua metodologia consiste em identificar componentes de uma figura que correspondam a uma imagem modelo como demonstra a figura 2.4.

Existem diferentes técnicas do *template matching*, algumas como a correlação cruzada ou a soma das diferenças absolutas, em que é procurada a secção da imagem que é mais idêntico ao do modelo procurado [5].

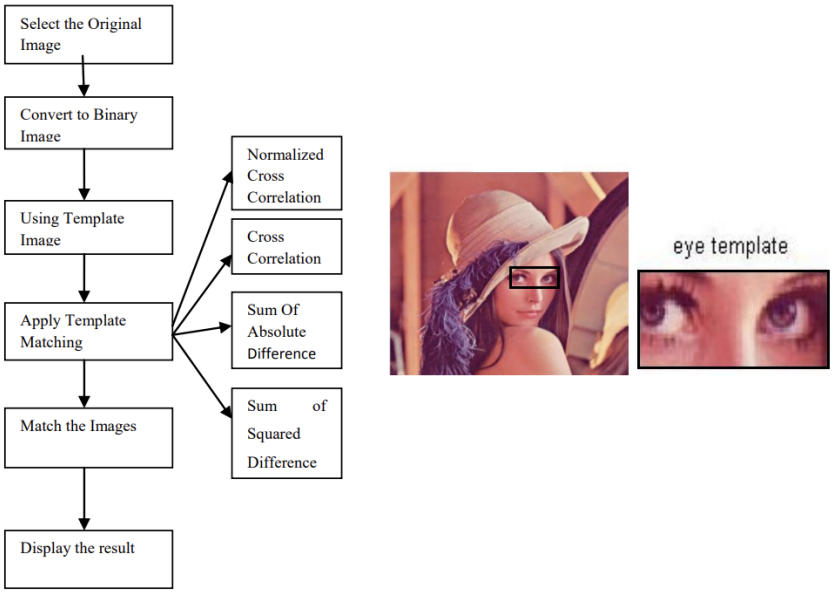


Figura 2.4: Diagrama do *template matching* (esquerda) e o seu exemplo (direita). Fonte: [5]

2.2 Machine Learning

Machine Learning (ML) é uma área da inteligência artificial dedicada à compreensão e construção de métodos de aprendizagem, através da utilização de dados, para melhorar o seu desempenho na tarefa [15].

Os algoritmos de *machine learning* (ML) são construídos com base num modelo com vários dados de amostra, utilizados para fazer previsões ou decisões sem serem declaradamente programados para isso. Estes algoritmos podem ser usados tanto na medicina, como na agricultura, como também no reconhecimento facial ou de voz, ou até mesmo na visão computacional [16].

2.2.1 Tipos de técnicas de Machine Learning

Existem técnicas de ML que envolvem o desenvolvimento e a implementação de algoritmos que, em vez de serem programados para atribuir determinados resultados (ações) em resposta a entradas específicas do ambiente, analisam os dados e as suas propriedades e determinam a ação, usando ferramentas estatísticas [6]. Portanto, os algoritmos de ML podem ser classificados em três categorias como apresenta a figura 2.5.

Aprendizagem supervisionada

A aprendizagem supervisionada depende de tarefas de ML para aprender uma função que mapeia uma entrada para uma saída, com base em exemplos de pares de entrada-saída, baseando na comparação entre a saída calculada e a saída prevista, isto é, calculando o erro e ajustá-lo para atingir a saída esperada.

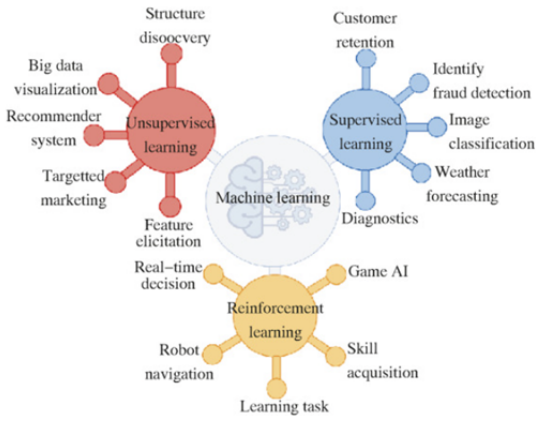


Figura 2.5: Principais técnicas de machine learning. Fonte: [6]

Existem alguns algoritmos de classificação como o *Support Vector Machine* (SVM) que têm como objetivo localizar um hiper-plano num espaço n-dimensional capaz de distinguir os espaços de classificação [17]. *Decision Trees*, possui características semelhantes em termos de objetivos, sendo este baseado num modelo, cuja decisão obedece a uma hierarquia, e que pode ser usado quer na solução de problemas de regressão, como também na solução de problemas de classificação [18]. O *Artificial Neural Networks*, que possui uma arquitetura multicamada, tais como a camada de convolução, de subamostragem e conexão completa, usadas para reduzir gradualmente dados e cálculos a um conjunto mais relevante, comparado com dados conhecidos para identificar ou classificar a entrada de dados [19].

Estando previsto na presente dissertação o uso do algoritmo SVM, serão abordados alguns pontos importantes. Neste algoritmo, nenhuma hipersuperfície não linear é obtida, mas o espaço das características (*features*) é transformado num espaço de maior dimensão, de modo que as características se tornem linearmente separáveis. Pelo que os vetores de características (*feature vectors*) podem ser classificados com um classificador linear. Por exemplo, a figura 2.6 apresenta duas classes num espaço de recursos 2D, onde são ilustrados quadrados pretos e brancos respetivamente. Num espaço em 2D, a linha que separa as classes pode ser encontrada. Ao adicionar uma terceira dimensão, deformando o plano construído pela Feature 1 e Feature 2, as classes tornam-se separáveis por um plano.

Para evitar problemas da dimensionalidade para o algoritmo de classificação, não as características, mas um kernel é transformado. Como tal, o desafio é encontrar o kernel adequado para transformar o espaço das características numa dimensão superior para que os quadrados pretos na figura subam e os brancos fiquem fixos. *Kernels* comuns são, por exemplo, o *kernel* polinomial não homogéneo ou o *kernel* da função de base radial gaussiana.

Com o SVM, a hipersuperfície de separação para as duas classes é construída por forma que a margem entre as duas classes se torne o maior possível. A margem é definida como a distância mais próxima entre o hiperplano de separação e qualquer amostra

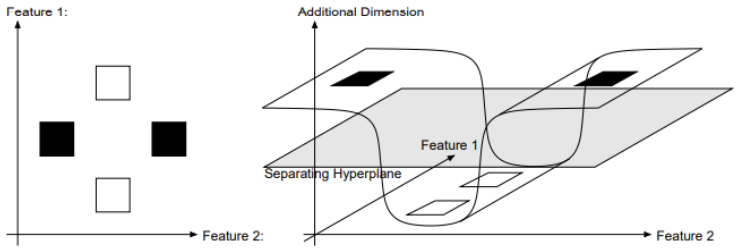


Figura 2.6: Duas classes separadas (branco e preto). No espaço das features em 2D (esquerdo) não podem ser separados por uma linha reta. A adição de uma dimensão (direita), as classes podem ser linearmente separadas. Fonte: [7]

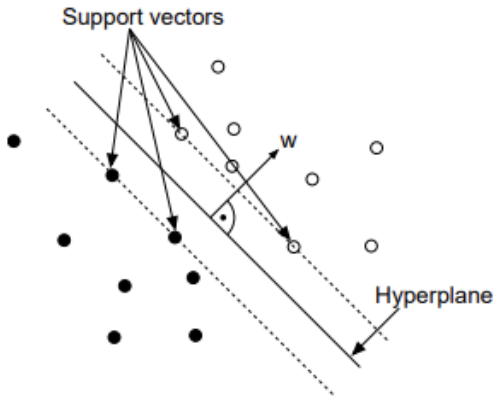


Figura 2.7: Vetores suporte que têm a distância mais próxima do hiperplano. Fonte: [7]

de treino. Ou seja, várias possíveis hipersuperfícies separadoras são testadas e a superfície com maior margem é selecionada. As amostras de treino de ambas as classes que têm exatamente a distância mais próxima da hipersuperfície (Figura 2.7) são chamados de vetores de suporte (*support vectors*) [7]. No entanto, é importante observar que nem sempre é possível encontrar um hiperplano de separação que separe perfeitamente todas as amostras das classes de treino. Isso pode acontecer em casos onde os dados são intrinsecamente não lineares ou quando as classes sobrepõem-se significativamente. Nesses cenários, dizemos que os dados são linearmente não separáveis, utilizando abordagens como o Kernel Trick, Margem Suave e o SVM não linear.

Aprendizagem não supervisionada

A aprendizagem não supervisionada baseia-se num conjunto de dados não rotulados, que são analisados sem interferência humana. Este tipo de aprendizagem utiliza dados que não contêm a resposta desejada, levando o algoritmo a ter a capacidade localizar estruturas e padrões capazes de atribuir uma classificação às classes desejadas.

Aprendizagem por reforço

A aprendizagem por reforço mostra um conjunto específico de algoritmos, cujos valores de entrada são obtidos com o intuito de recolher uma resposta do sistema, assim como

avaliar e decidir qual o passo seguinte.

Esta técnica utilizada existe o "agente", que atua e prevê as características numa etapa futura com base nas passadas. Consoante este resultado, é gerada uma recompensa ou penalidade, conforme o caso, que é atribuída com base na previsão e no estado em que o ambiente ou o agente se encontram, e toma essa decisão [8]. A figura 2.8 representa a forma como o agente executa a ação perante o ambiente e, por sua vez, é devolvida ao agente, da parte deste, uma recompensa e o estado em que atualmente se encontra no ambiente.

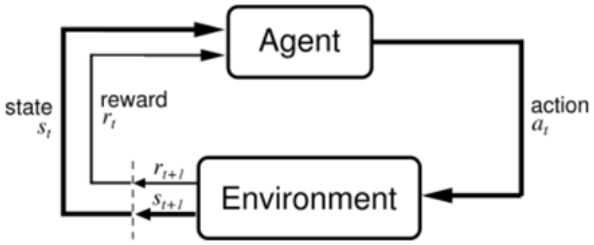


Figura 2.8: Relação entre o ambiente e o agente. Fonte: [8]

2.3 Trabalhos desenvolvidos em aplicações de Template Macthing

As aplicações de algoritmos através do *template matching* tem sido um forte potencial para as áreas de visão computacional, robótica e realidade aumentada. o principal foco desta técnica consiste na identificação e localização aproximada de uma imagem de referência (template) dentro de uma imagem de busca.

Um algoritmo rápido e robusto, proposto em [9], foi baseado por um método de gradiente descendente que atualiza iterativamente a localização do modelo até à convergência. O objetivo consiste no uso de outro método eficaz para além do uso do modelo tradicional. O termo "estimadores M"indica que o estudo envolve o uso de métodos estatísticos de estimativa M para melhorar a robustez e a precisão da correspondência de modelo. Os estimadores M são uma classe de estimadores estatísticos que são robustos a outliers e podem lidar com ruído e variações nos dados. Este algoritmo pode lidar com *outliers* e ruído na imagem usando o *M-estimators*, uma vez que estes são estimadores estatísticos e robustos que diminuem a influência de *outliers* nos dados. O algoritmo avalia um conjunto de dados das imagens com vários tipos de ruído e *outliers*. Utilizaram uma sequência de imagens, segmentada num conjunto de blocos, e tentaram encontrar o vetor de movimento para cada bloco (16x16 e 32x32). Os resultados mostram que as taxas de consumo de tempo e as taxas de contagem de operações (Figura 2.9) demonstram que o algoritmo é bastante preciso e robusto, comparado com outros algoritmos com diferentes estimadores (Geman, McClure, Tukey ou Huber), tornando-o adequado para aplicações em tempo real.

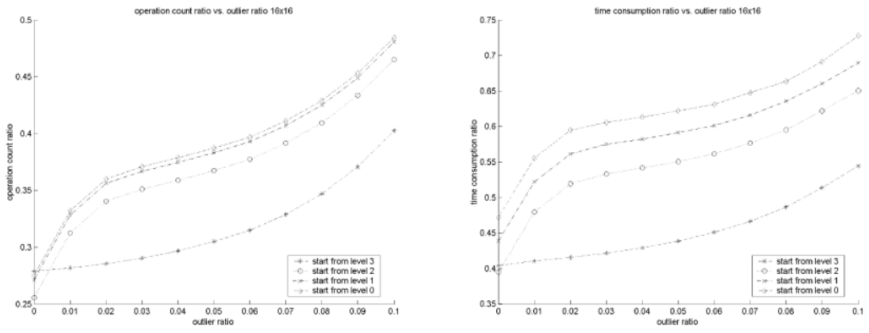


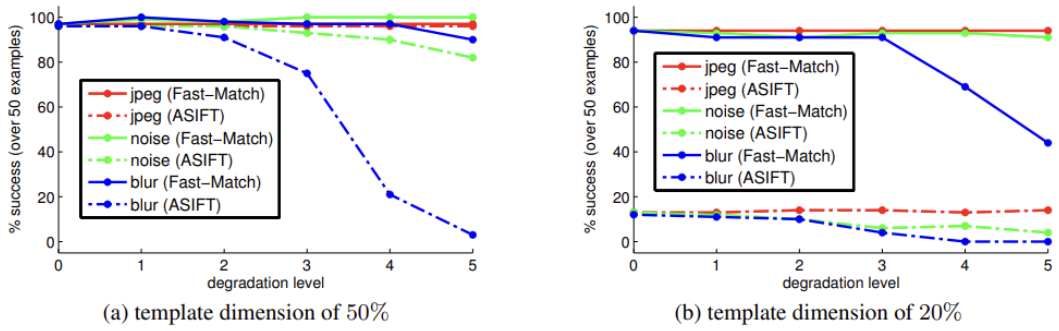
Figura 2.9: Taxas de consumo de tempo (esquerda) e as taxas de contagens de operações (direita) para blocos de tamanho 16x16. Fonte: [9]

Outro algoritmo rápido e preciso chamado Fast-Match [1] é baseado numa abordagem hierárquica que estima, em primeiro, os parâmetros de transformação, usando um pequeno número de pontos-chave e, em seguida, usa esses parâmetros para realizar um *template matching* denso das regiões da imagem. Este algoritmo pode lidar com grandes mudanças de escala, rotação e perspetiva.

Uma das experiências feitas com base em várias condições de imagem, testou este algoritmo com outro algoritmo (ASIFT), com vista à avaliação do tempo de execução médio acima de 100 instâncias para cada dimensão do modelo, entre 10% e 50% (Tabela 2.2). Ou, ainda numa perspetiva de avaliação dos diferentes níveis de degradação de imagem (Figura 2.10) de 0 (sem degradação) para 5 (mais alto).

Tabela 2.1: Tempos de execução em diferentes tamanhos do modelo. Fonte: [1]

Template Dimension	90%	70%	50%	30%	10%
ASIFT	12.2s	9.9 s	8.1 s	7.1 s	NA
Fast-Match	12.5s	2.4 s	2.8 s	6.4 s	25.2 s



(a) template dimension of 50% (b) template dimension of 20%

Figura 2.10: Desempenho sob diferentes tamanhos do modelo e degradações de imagem. Fonte: [1]

Poder-se-á concluir que o algoritmo Fast-Match é bastante rápido para tempos de execução e robusto para as diferentes degradações de imagem, tendo uma robustez ainda maior para altos níveis de imagem desfocada.

2.4 Trabalhos desenvolvidos na área de aplicações de redes neuronais para inspeção de defeitos de furos

Relativamente à utilização de redes neuronais para a identificação de furos, foram localizados vários artigos que abordavam o tema, contudo, as áreas de aplicação eram distintas. Apesar disso, era a identificação de furos o assunto em foco. Assim sendo, da pesquisa realizada, apurou-se que um deles abordava uma arquitetura, usando as modificações do codificador-decodificador *U-Net* [10, 12], métodos de limite desenvolvidos em *Python* através da biblioteca *OpenCV* para medir os fatores de diâmetro e delaminação para cada furo [13].

O modelo de segmentação em [10] consiste na extração de características da imagem e na reconstrução do mapa de segmentação de furos em painéis de móveis de madeira com textura, como demonstra na figura 2.11. Neste modelo foram utilizados vários métodos e fizeram-se comparações entre eles de acordo com a melhor pontuação Dice, isto é, se a tarefa de segmentação em nível de píxel propõe um leve aumento de desempenho computacional. Os resultados obtidos demonstram que cada arquitetura de rede são escolhidas de acordo com a melhor pontuação de Dice. A melhor pontuação foi obtida com os métodos de conexões residuais, *pooling* da pirâmide espacial atraso, blocos de compressão e excitação (pontuação 3,4%).

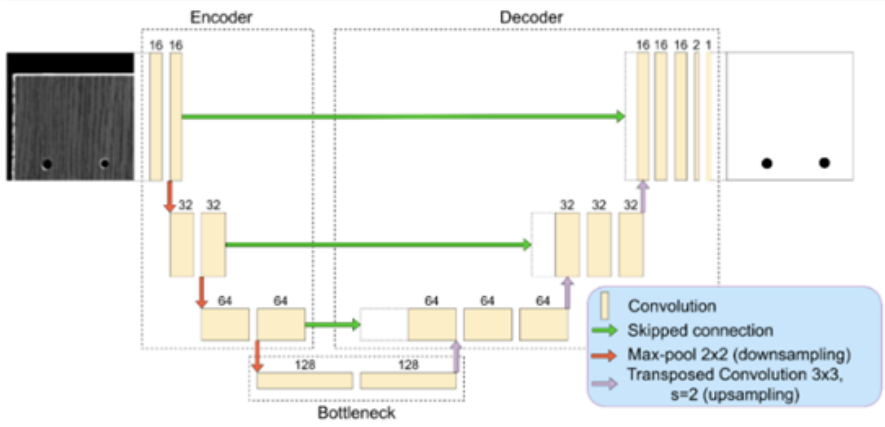


Figura 2.11: Estrutura principal de uma linha base U-Net. Fonte: [10]

Em [11] trata-se da inspeção de defeitos de forma para uma haste de ânodo. Foi adquirido um conjunto de dados de imagem, compostos por multiclasse de defeitos de haste de ânodo. Os algoritmos de deteção de objetos foram desenvolvidos para desenhar uma caixa delimitadora em torno do objeto de interesse para o localizar na imagem. Portanto, um algoritmo como *Faster R-CNN* (Figura 2.12) foi desenvolvido para detetar incidentes perante uma eventual existência de várias caixas delimitadoras, que representem diferentes objetos de interesse na imagem e localizá-las rapidamente.

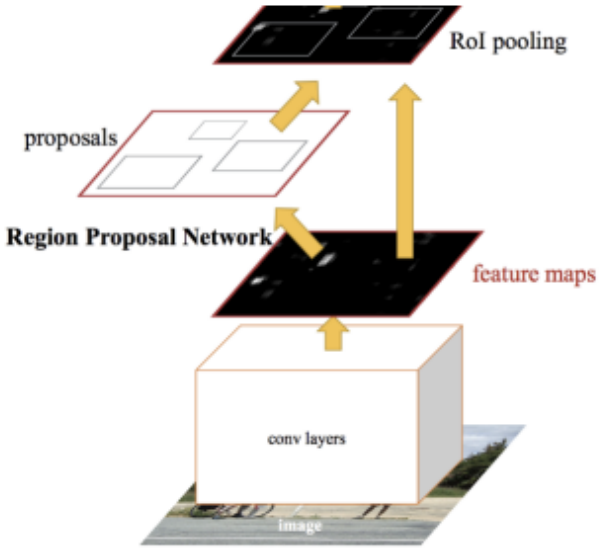


Figura 2.12: Algoritmo *Faster R-CNN*. Fonte: [11]

Por último, em [12] foi desenvolvido um novo sistema autónomo para a deteção e segmentação precisas de danos e fissuras ao redor de furos em compósitos de polímeros reforçados com fibra de carbono, através da arquitetura *U-Net* e com recurso a uma câmara. O sistema compreende três módulos que, de uma forma muito resumida, consiste em obter 4 imagens de diferentes direções de iluminação e de seguida são fundidas para suprimir o fundo e aumentar a visibilidade dos danos e fissuras. E finalmente processar a imagem através de várias etapas para obter o perfil do furo, a área do dano e as linhas de fissura (Figura 2.13). Dentro desses módulos, uma Rede Totalmente Convolucional. com a arquitetura *U-Net*, foi projetada para a segmentação semântica pixel a pixel de imagens de furos. Os testes experimentais mostram que o modelo *U-Net* pôde fornecer uma avaliação em tempo real e totalmente automatizado com um erro máximo de 5,4%.

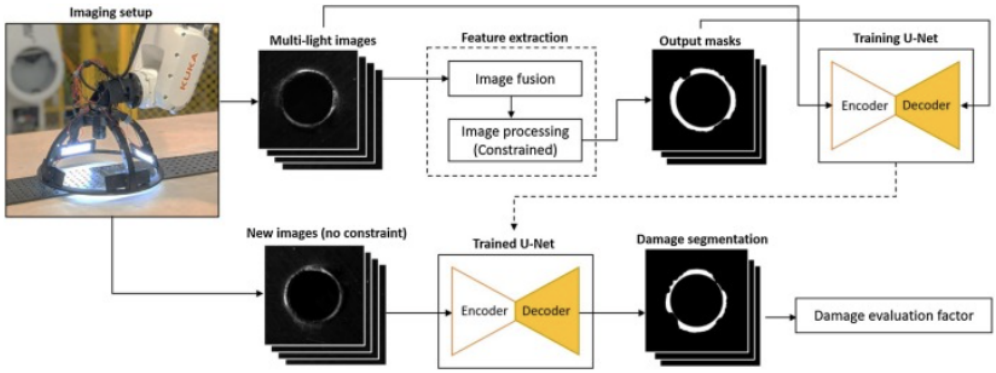


Figura 2.13: Um esquema do método de inspeção baseado em visão proposto com base na arquitetura *U-Net* [12]

2.5 Trabalhos desenvolvidos em aplicações de Machine Learning na identificação e inspeção de defeitos

Nas aplicações de *Machine Learning*, nomeadamente na inspeção e identificação de defeitos, foram encontrados alguns artigos que abordavam temas bastante semelhantes ao projeto a desenvolver, uma vez que os autores [2, 20] utilizaram métodos semelhantes no que toca à inspeção automática de defeitos na rosca.

No artigo [2], o autor pretende obter um método de inspeção mais eficaz para a deteção de roscas M5, com menor tempo de ciclo que o método de inspeção (Figura 2.14) através de um medidor passa/não passa (*gage go/No-go*). Uma ferramenta usada pelos operadores que compara mecanicamente as interferências entre a peça fabricada e o próprio dispositivo. Utilizou um método não invasivo (*non-invasive media*), que poderia ser usado para apoiar o diagnóstico da produção, garantindo a característica sem danificar a peça (Tabela 2.2).

Tabela 2.2: Comparação entre medidores passa/não passa e meios não invasivos. Fonte: [2]

Criteria	Use of gage go No-go	Use of non-invasive media
Cycle Time	12 seconds minimum	0.5 seconds maximum
Operators	Dedicated	Shared
Damages	Highly possible	Non-Possible
Repeatability	High	Low
Reproducibility	High	Low

Cerca de 300 imagens foram utilizadas para amostras da produção em linha, para que possam ser comparadas à imagem da câmara do tipo *CMOS* (*Complementary Metal Oxide Semiconductor*). Os resultados adquiridos mostram que em 300 amostras, cerca de 185 peças estão bem processadas. Quanto às restantes não apresentam roscas (75), ou nem mesmo furos (40). As inspeções feitas nas peças apresentaram uma eficiência de 78%. Estes resultados foram obtidos através de uma ferramenta estatística incluída no software de simulação de sensores de visão *Keyence IV*.

No artigo [20] o autor utiliza uma câmara *CCD* (*Charge-Coupled Device*) com resolução 3272x2469 *pixels*, que propõe duas abordagens de apenas um método na inspeção de defeitos de furo central fechado da peça, apesar de existirem mais dois métodos de inspeção, em que estes consistem na ausência de rosca no parafuso e rebarbas nas conexões soldadas. Porém, estes métodos são pouco importantes para este projeto. O algoritmo foi implementado em C++ e utilizaram a biblioteca *OpenCV* (Figura 2.15). A primeira abordagem consistia na aplicação da transformada circular de *Hough*, onde encontra vários círculos, sendo o correto determinado pelo centro e diâmetro do furo. Já na segunda abordagem utilizaram uma caixa delimitadora de contornos para extrair os contornos da peça metálica e calcular o retângulo com a área mínima que circunscreve o componente (*BoundingBox*). Reconhecido o centro do retângulo, que é o centro do furo, é verificado se o seu nível de cinza é igual ao fundo.

Os resultados revelaram-se positivos para ambas as abordagens, porém a transfor-

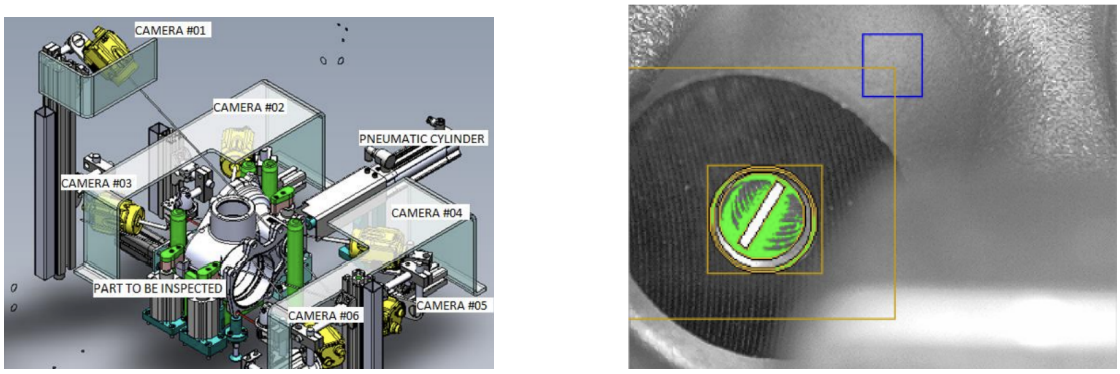


Figura 2.14: Sistema de inspeção para a deteção de presença/ausência de roscas laminadas (esquerda) e a inspeção da presença de rosca M5 (direita). Fonte: [2]

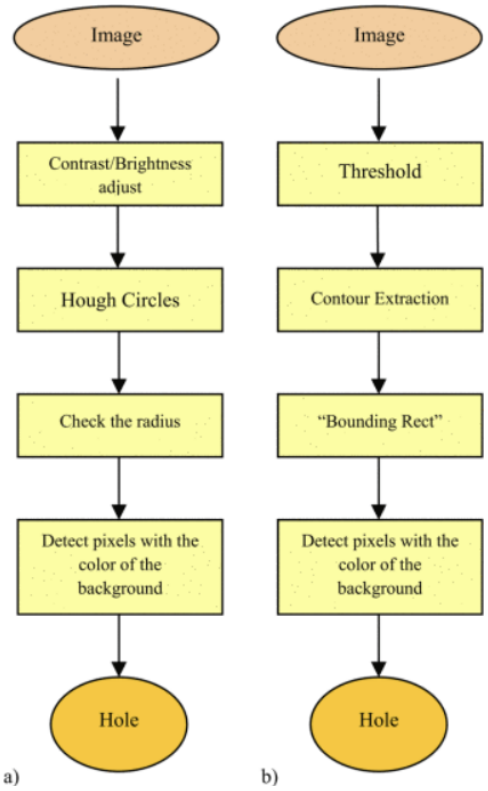
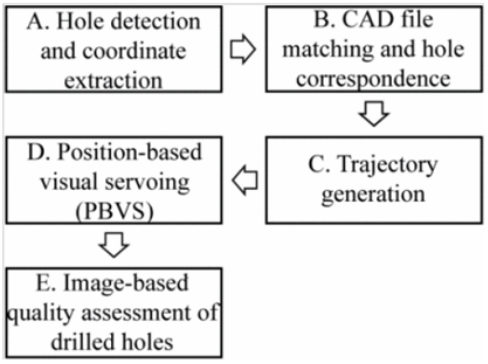


Figura 2.15: Diagramas de blocos das duas abordagens usadas para detetar o defeito do orifício fechado: a) Transformada circular de *Hough* b) *BoundingBox* . Fonte: [20]

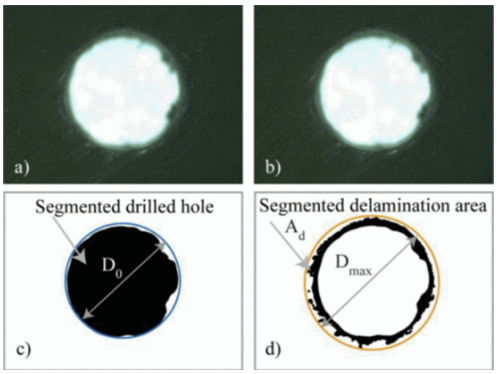
mada de *Hough* leva mais tempo a processar, muito embora mais precisa. A desvantagem deve-se aos diferentes níveis de iluminação ou às alterações nas propriedades metálicas da peça. Na abordagem da *BoundingBox*, a presença de uma rebarba significativa pode afetar o centro da caixa delimitadora. Para minimizar o efeito, aplicou-se um filtro morfológico, sendo a abordagem escolhida para integrar o sistema como um todo, e fizeram-se

os parâmetros de configuração.

Em [13], os métodos para a inspeção autónoma de furos em painéis de móveis de madeira com textura consistem na medição do diâmetro e a delaminação para cada furo (Figura D.1d) com recurso a duas câmaras. Uma de baixa resolução (*Logitech C525*) e outra de alta resolução (*Basler ACE ACA1440-220uc*). A sua estrutura para a inspeção autónoma de furos consistiu em cinco módulos (Figura 2.16a). Os resultados experimentais mostraram que o sistema conseguiu medir o diâmetro dos furos com um erro máximo de 10%, com uma repetibilidade de 99,5% e uma precisão de 90%. Uma melhor precisão poderia ter sido obtida através de uma câmara de melhor resolução.



(a) Diagrama de fluxo da estrutura baseada em visão



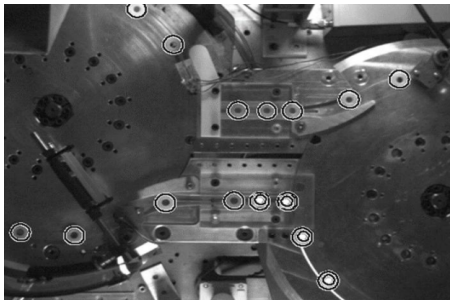
(b) Segmentação da área de delaminação

Figura 2.16: Métodos e algoritmos de processamento de imagem com base na biblioteca OpenCV. Fonte: [13]

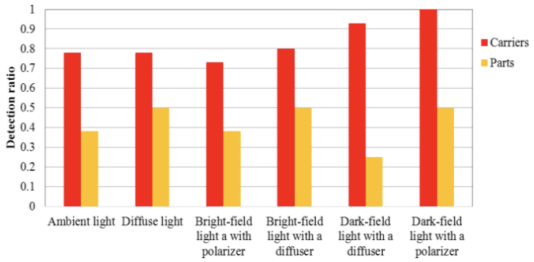
Os artigos [21, 22] demonstram modelos matemáticos específicos, através de princípios das onduletas (wavelets) e algoritmos de multiresolução com base em câmaras *CCD*. Estes modelos são usados para classificar os coeficientes de onda para diferentes níveis de resolução e depois fazer a distinção entre as classes, designadamente se apresenta ou não defeito [21]. Os modelos de mistura Gaussiana, usada para o objeto do sistema detetor de primeiro plano para monitorizar a máquina enquanto esta está em operação. Por sua vez classifica a condição de operação em uma operação normal tanto para o fluxo óptico como também para a média de execução [22] com recurso a *MatLab*.

Em [21] a determinação de ondas ortogonais garantem uma melhor descorrelação das informações contidas em diferentes níveis de resolução. A análise de componentes principais permite manter apenas as informações mais discriminantes. A fase de decisão do sistema de controle usa um algoritmo de classificação abaixo do ideal mas rápido. A taxa final de deteção de erros é inferior a 1% e a taxa de não deteção é praticamente nula.

Nos [22] diferentes algoritmos de modelos gaussianos, implementados em *MatLab*, requerem menos tempo de treino e processamento; têm uma capacidade de maior rapidez na deteção de eventuais falhas; um processamento superior em termos de qualidade num



(a) Imagem sob luz de campo escuro com um polarizador para detetar os portadores (círculos brilhantes) e as peças *O-rings* (círculos negros)



(b) Comparação da taxa de deteção com as condições de luminosidade

Figura 2.17: Algoritmo testado sob as condições de iluminação e os resultados obtidos. Fonte: [14]

curto espaço de tempo por frame; e um tempo de resposta mais rápido. O método do modelo de mistura Gaussiana e média de execução demonstraram um tempo de processamento de um vídeo de 10 segundos, que demonstra a inspeção dos produtos e seus portadores, em apenas cerca 5,88 segundos, e precisou de apenas 3 frames para detetar a falha introduzida.

Por último, mas não menos importante, e sendo um ponto fulcral para o projeto a desenvolver, com base na iluminação, o autor [14] (curiosamente autor também de [22]) explica os resultados de um estudo feito pela investigação das técnicas nos efeitos da luminosidade, cujo foco principal consiste na deteção de peças pequenas (*O-rings*) posicionadas num fundo que reflete luz, e os seus portadores com recurso ao software *MatLab* como se pode ver na figura 2.17a.

A câmara não vê um objeto, mas sim a luz refletida do objeto. O desenvolvimento e tempo de processamento para um sistema visão máquina depende muito da seleção das luzes adequadas. Perante isto, testaram um algoritmo sob as condições de iluminação e compararam resultados, chegando à conclusão de que uma luz de campo escuro e um polarizador instalado na frente da câmara, para a redução de brilho da imagem, é a mais eficaz, obtendo 100% na deteção dos portadores, e 50% para as peças *O-rings* como demonstra a figura a 2.17b. As peças foram detetadas com sucesso na região com luz de fundo colocada imediatamente após a faixa de transferência, debaixo da roda secundária da máquina. A luz de fundo cobria apenas a porção limitada do campo de visão. Daí a taxa de deteção para as peças ser inferior à dos portadores.

2.6 Análise crítica

No que toca aos artigos realizados através de técnicas de *template matching*, a deteção de anomalias, onde envolve a comparação de uma amostra com um modelo de referência para determinar se o produto apresenta conformidade ou não, é um bom princípio. No entanto, o uso destas tecnologias pode não atender aos requisitos industriais de precisão

e eficiência de identificação. É uma ferramenta poderosa para detetar anomalias, mas o seu desempenho pode ser limitado devido a vários fatores tais como a seleção do modelo, o ruído e as variações nos dados. Como tal, é importante avaliar cuidadosamente os métodos de *template matching* específicos, levando em consideração as características dos dados e as métricas de desempenho desejadas.

Relativamente às aplicações de redes neuronais para a inspeção de furos, estas apresentam algumas arquiteturas semelhantes com métodos diferentes. O artigo [10] revelou que módulos e camadas mais avançados aumentaram a precisão da segmentação do modelo. As diferenças podem ser mais distinguíveis em amostras mais complicadas através de vários métodos. Por último, em [12] apresenta uma arquitetura com um nível de desenvolvimento muito superior que em [11], com um sistema de inspeção mais simplificado, onde se podem obter imagens dos furos a partir de diferentes perspetivas, e assim obter resultados com maior precisão e menor erro. No entanto, a complexidade das arquiteturas apresentadas, comparando a dimensão do projeto, a utilização de um algoritmo menos complexo será suficiente para garantir a identificação da presença de rosca no furo do cárter.

Assim, atendendo aos resultados obtidos pelos autores [10, 12] no que confere à utilização de arquiteturas e algoritmos pré-treinados, o uso destes no presente projeto, como parte na identificação de furos nas peças maquinadas, pode ser a solução.

Por outro lado, tendo em conta as aplicações de *Machine Learning* na inspeção e identificação de defeitos, é possível observar que, dos artigos citados [2, 20–22] apresentam bastantes métodos no que confere ao tratamento dos dados onde seriam aplicados os algoritmos de ML, revelando-se de extrema importância. Apesar dos artigos [21, 22] demonstrarem algoritmos bastante complexos para decifrar, os desenvolvidos em [22] apresenta melhor destaque pelo facto destes possuírem um tempo de ciclo muito inferior e uma deteção de defeito muito mais precisa. Já em [21] não demonstra que tipo de software foi utilizado para obter os resultados obtidos. Para [2], os métodos mostram-se bastante úteis e conferem valores precisos, contudo para [2] o software utilizado é específico (*Keyance IV*) para a câmara que foi usada. No artigo [14] as condições de iluminação são fatores importantes para que a inspeção seja feita com a maior precisão possível. Portanto, o algoritmo desenvolvido demonstra grande importância, apesar da taxa de deteção das peças *O-rings* não ser a esperada. Por fim, os resultados experimentais em [13] são bastante positivos no que toca à sua taxa de precisão e de erro. Logo, os métodos desenvolvidos na biblioteca *OpenCV* para a medir os diâmetros dos furos e a sua delaminação, podem ser um recurso necessário para o projeto, no que à classificação do furo disser respeito.

Assim sendo, é possível concluir que a identificação e classificação de defeitos se revela de elevada importância, tanto no uso de técnicas de Template Matching para um primeiro passo, como também para a performance final do algoritmo de *Machine Learning* e, ainda, para as câmaras usadas. No que diz respeito aos algoritmos usados, nos artigos [2, 13, 22] os resultados obtidos revelaram-se bastante satisfatórios, pelo que se esperam resultados igualmente satisfatórios dos algoritmos escolhidos. Estes algoritmos utilizados, com base nas condições de iluminação em [14], poderiam ser possíveis solu-

ções. Relativamente à câmara, a do tipo *CCD* será a mais favorável, com características semelhantes às do autor [14], apesar de possuir um menor consumo de energia, melhor correção da exposição de luz e uma menor sensibilidade à luz, é um tipo de câmara que é bastante utilizada nos dias de hoje e que é maioritariamente utilizada na empresa Renault.

Capítulo 3

Solução proposta e implementação

Neste capítulo a arquitetura hardware de duas soluções conceptuais será descrita ao pormenor, sendo seguida de uma demonstração da arquitetura do software proposta para cumprir com o funcionamento do mesmo. Por fim, é descrita a implementação de todos os componentes das duas arquiteturas propostas anteriormente, bem como os algoritmos desenvolvidos para o processamento da imagem.

3.1 Arquitetura *hardware* da solução proposta

Existem duas proposta de funcionamento da linha. A primeira consiste na utilização de um sensor para o aviso da presença do cárter na zona de inspeção. Um dispositivo presente na linha é acionado para elevar e fixar o cárter numa posição em que a peça não sofra a influência do movimento da linha e permita, assim, a captura de uma imagem com a maior eficiência possível. Por sua vez, um robot inicia o seu movimento com uma câmara acoplada, deslocando-se para as posições dos furos e a câmara inicia a sua captura de imagem para que, posteriormente, esta seja dirigida para o seu tratamento com recurso a um software de visão sofisticado para a avaliação da qualidade da peça. E a segunda, não muito diferente da primeira, apresenta duas câmaras que se encontram fixamente posicionadas para a captura de imagem dos dois furos da peça, cujo procedimento feito é semelhante à da primeira arquitetura.

De acordo com os objetivos propostos, a figura esquematizada (Figura 3.1) demonstra as duas arquitetura do sistema em desenvolvimento.

Para automatizar a linha de inspeção visual são necessários 3 componentes integrantes do projecto, designadamente a câmara, o computador industrial e o autómato, aliados a dois programas que serão igualmente desenvolvidos, nomeadamente o programa da inspeção e identificação, e o programa de comunicação.

A arquitetura 1 consiste numa comunicação por TCP/IP de um sensor, e um elevador que se encontra na linha de produção, que poderão ser reaproveitadas, onde estarão ligadas a um PLC da marca Siemens do modelo S7-1200 cujo a sua comunicação será feita simultaneamente para a câmara e para um computador industrial. Ou seja, a passagem do cárter pelo sensor irá ativar uma saída do PLC para o elevador que fará elevar

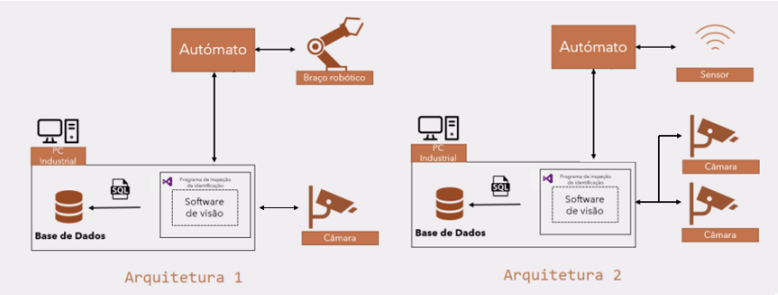


Figura 3.1: Arquitetura hardware de duas soluções propostas

o cárter, libertando este do movimento da linha e fixando a sua posição para inspeção.

Seguidamente o PLC irá transmitir um sinal de saída para o braço robótico com vista a ativar o início do programa deste. Estando o robô em posição para a captura de imagem, este envia um sinal para o autómato, transmitindo dois sinais de saída em simultâneo. Um para a câmara, para que esta inicie o processo das capturas de imagens, e outra para dar início a execução do programa de visão. Enquanto isto, o programa continua a classificar a imagem da primeira posição. Um sinal é enviado do computador industrial que passa pelo PLC e é transmitido para o robot com vista a alterar a sua posição para a captura de uma segunda imagem, e o processo decorre de igual forma na sua plenitude. Quando terminado o programa de visão, este dispõe uma interface, possibilitando uma visualização da foto de inspeção obtida. Acresce referir que o programa estará interligado a uma base de dados para proceder ao registo do cárter que acabou de ser inspecionado.

No que toca à arquitetura 2, não difere muito da arquitetura 1, a não ser a utilização de duas câmaras já posicionadas para a captura, substituindo o braço robótico. De forma sucinta, poder-se-á referir que, até à etapa em que o cárter já se encontra fixo no elevador, a comunicação é idêntica à da arquitetura 1. O PLC recebe sinal do elevador que depois transmite para as duas câmaras e para o computador industrial para, por sua vez, executar o programa de visão, passando depois para etapa final anteriormente descrita.

3.2 Arquitetura de software do programa

Relativamente à arquitetura geral do software desenvolvido, no âmbito do presente projeto, esta é ilustrada na figura 3.2. O módulo de aquisição de imagem recolhe, em tempo real, a imagem proveniente da câmara disponível que esteja ligada ao computador e que se encontre a correr, na câmara, o software desenvolvido. O módulo de processamento de imagem implica a aplicação de todos os parâmetros previamente especificados (exposição e ganho) e guardados em memória, filtrando a imagem proveniente da obtenção pela câmara, de forma a facilitar a diferenciação entre peças "conforme"e "não conforme". O módulo de deteção de defeitos, como o nome indica, identifica os defeitos da peça a analisar, tendo por base os parâmetros previamente especificados. Essa identificação baseia-se na análise de uma região da imagem adquirida, delimitada por uma forma ge-

ométrica de forma a extrair os principais recursos da imagem. Finalmente, o módulo de armazenamento dos resultados é responsável por guardar toda a informação referente ao teste efetuado na peça.

Figura 3.2: Arquitetura geral do software

3.3 Armazenamento de dados propostos

A organização da base de dados é fundamental e de extrema importância, uma vez que existirão um conjunto de dados diferentes para cada peça durante o processo de produção. A ideia seria introduzir mais uma informação que, por sua vez, se insere no histórico de registos do cárter. Pensou-se em usar a base de dados PostGreSQL, uma vez que do conhecimento adquirido foi a melhor opção para economizar tempo e minimizar o risco de erros durante o desenvolvimento e implementação do projeto.

Nesta altura está prevista a criação de uma base de dados no *pgAdmin4*, uma ferramaneta de administração para o PostGreSQL. Para este projeto foi tida em conta uma tabela que contém toda a informação referente à presença de rosca do cárter. Na figura 3.3, é apresentada a tabela *regist_hist* com as colunas julgadas necessárias para satisfazer a arquitetura, contendo a data de registo, a referência de cada peça, os resultados da conformidade da peça (OK ou NOK), os resultados da conformidade para cada furo e a localização das imagens dos furos onde foram guardadas.

3.4 Protótipo inspeção por visão

Como o protótipo foi feito a curto prazo, utilizou-se um robot UR10 para posicionar e manipular uma câmara Dalsa com uma Ring-light acoplada; um computador industrial com o programa de visão, um PLC S7-1200 como sistema de comunicação, por meio I/O entre autómato e UR10 e uma conexão TCP/IP entre autómato e o programa. O desempenho do sistema foi avaliado por meio de uma série de experimentos, demonstrando a capacidade de detetar as roscas de forma fiável e eficiente. A figura 3.4 demonstra uma parte do sistema implementado onde se testou as posições do braço robótico com a câmara e iluminária acoplada.

	id [PK] integer	timer timestamp with time zon	modelo character varying	referencia character varying	resultado character varying	furo_240 character varying	furo_312 character varying	localizacao_furo240 character varying	localizacao_furo312 character varying
1	30	2023-05-19 11:49:56...	HR10	158R5619	NOK	NOK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
2	36	2023-05-19 13:50:28...	HR10	158R2960	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
3	37	2023-05-19 13:58:28...	HR10	158R4788	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
4	38	2023-05-19 13:59:25...	HR10	158R6147	NOK	NOK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
5	39	2023-05-19 13:59:57...	HR10	158R6310	NOK	OK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
6	40	2023-05-19 14:00:19...	HR10	158R4262	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
7	41	2023-05-19 14:00:32...	HR10	158R1938	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
8	42	2023-05-19 14:00:58...	HR10	158R2854	NOK	NOK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
9	43	2023-05-19 14:01:11...	HR10	158R9097	NOK	OK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
10	44	2023-05-19 14:01:29...	HR10	158R2642	NOK	NOK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
11	45	2023-05-19 14:01:51...	HR10	158R6424	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
12	46	2023-05-19 14:02:02...	HR10	158R8986	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
13	47	2023-05-19 14:02:30...	HR10	158R6084	NOK	NOK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
14	48	2023-05-19 14:02:48...	HR10	158R3358	NOK	NOK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
15	49	2023-05-19 14:03:11...	HR10	158R8808	NOK	OK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
16	50	2023-05-19 14:03:33...	HR10	158R9321	NOK	OK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
17	51	2023-05-19 14:03:49...	HR10	158R7764	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
18	52	2023-05-19 14:04:21...	HR10	158R5442	NOK	NOK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
19	53	2023-05-19 14:04:38...	HR10	158R8015	NOK	NOK	NOK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...
20	54	2023-05-19 14:04:59...	HR10	158R6997	OK	OK	OK	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...	C:\Users\Pombeiro\AppData\Roaming\MVTec\HALCON-22.11-Stead...

Figura 3.3: Tabela regist_hist

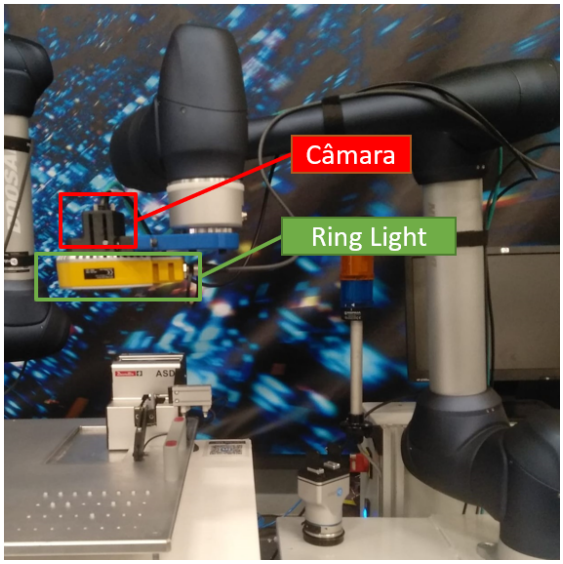


Figura 3.4: Braço robótico com uma câmara e iluminária acoplada

Após reconhecida a câmara no computador, procedeu-se as alterações e calibrações necessárias, utilizando as ferramentas necessárias para a manipulação de todas as funcionalidades da câmara usada. Estabeleceram-se as capturas de imagens através de um trigger da câmara, ou seja, o programa irá enviar um pedido de imagem através do *trigger* da câmara.

3.4.1 Câmara

A câmara utilizada é uma câmara industrial da marca Dalsa do modelo Genie Nano M2020. Esta câmara contém um sensor de imagem de alta qualidade com resolução de 2048x1536 pixels e suporta uma ampla gama de framerates e tempo de exposição. Esta comunica através da ligação do tipo Gigabit Ethernet e USB 3.1, que permitem uma transferência de dados rápida e confiável. De referir, ainda, que a câmara está equipada com recursos avançados, como exposição automática, controlo automático de ganho e seleção de região de interesse (ROI), permitindo aos utilizadores capturar imagens de alta

qualidade sob vários níveis de luminosidade. Refira-se, ainda, que esta possui entradas digitais, através das quais será dada ordem de captura de imagens (*triggers*).
A referida câmara vem dotada com um kit de desenvolvimento de software (SDK) Sapera CamExpert abrangente para fácil integração em plataformas de software de terceiros e aplicativos personalizados com as câmaras Teledyne DALSA.

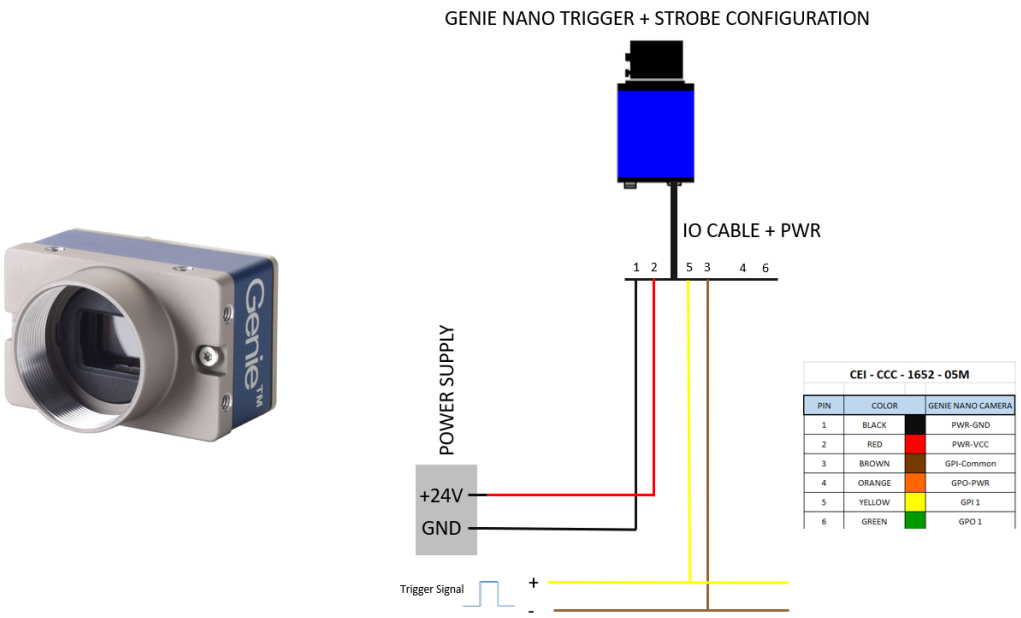


Figura 3.5: Câmara Teledyne Dalsa Genie Nano M2020 Mono e o seu esquema elétrico correspondente

A figura 3.5 demonstra o modelo da câmara usada e a ligação como foi feita ao autómato, sendo os pinos 1 e 2 ligados à fonte da alimentação. Os pinos 5 e 3 dão o sinal *trigger* da câmara, sendo que o pino 5 estará ligado a um *output* do autómato e o pino 3 ligado a 0V.

3.4.2 Lente

As lentes utilizadas no projeto são escolhidas de acordo com a câmara a ser utilizada, uma vez que esta possui características específicas para a escolha de cada tipo de lente. Das características disponíveis podemos destacar o tipo de montagem, em que, para esta câmara será do tipo C-Mount; o tamanho da lente; e o tamanho da câmara em relação ao seu sensor.

Os factores em que assentou a escolha da lente, baseou-se no tamanho relativo do sensor da câmara, tendo este como base o cálculo da distância focal (f). A abordagem é semelhante em [23]. A formula seguinte representa o cálculo de f:

$$f = s \times \frac{WD}{FOV}$$

Sendo:

- s - A dimensão horizontal do sensor escolhido, para a câmara escolhida este apresenta um valor de 7,12 mm.
- WD - A distancia de trabalho, sendo 240 mm para o furo 240 e 180mm para o furo 312.
- FOV - O campo de visão

O cálculo do campo de visão é feita através da fórmula abaixo apresentada:

$$FOV_{\min} = D_M + T_p + M$$

Sendo:

- D_m - A dimensão máxima da captura, sendo um valor de 107mm.
- T_p - A tolerância do posicionamento, tendo essa tolerância um valor de 2 mm.
- M - A margem, optou-se por colocar uma margem de segurança de 10 mm.

Atráves dos cálculos efetuados relativos para a escolha da lente, apresentam-se no apêndice A. A escolha desta incidiu sobre uma lente GM6HR31614MCN, com uma distância focal de 16mm.

3.4.3 Iluminação

Um dos grandes desafios que ocorreu durante o estágio foi sem dúvida a iluminação adequada para a captura de imagem. Apesar da elevada evolução tecnológica, a sensibilidade e versatilidade das câmaras atuais ainda não atingiu os níveis da visão humana. O olho humano, ao contrário das câmaras, consegue adaptar- se facilmente ao meio envolvente, mesmo em condições pouco favoráveis. Para que uma câmara possa capturar uma imagem que o olho humano conseguiria obter sem dificuldade, as condições de iluminação devem ser muito cuidadas. A dificuldade para obter a imagem desejada torna-se clara em objetos que apresentam formas muito complexas e superfícies muito reflexivas. Neste caso, a inspeção de um furo revela-se bastante mais difícil em comparação à inspeção de uma superfície.

Neste momento, um sistema de iluminação para aplicações industriais não se restringe apenas numa lâmpada, ou a um conjunto de LEDs, mas sim um conjunto complexo de vários componentes, tais como fontes de luz, difusores, filtros, etc. Todos estes recursos são fundamentais para desenvolver um sistema capaz de resistir às condições adversas, em termos de luminosidade,presentes num ambiente industrial. No que diz respeito às técnicas de iluminação, existem diferentes abordagens, incluindo iluminação frontal, lateral e coaxial. Para este projeto em particular, optou-se por uma iluminação frontal utilizando um emissor de luz LED *EFFI-Ring* na cor verde (Figura 3.6), uma luz anelar luminosa ajustável que oferece múltiplas configurações possíveis, com uma montagem fácil na extremidade do braço robótico.

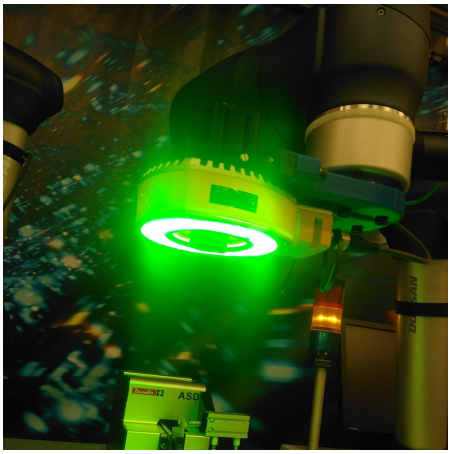


Figura 3.6: Emissor de luz LED *EFFI-Ring* LDR2-120RD2-WD

3.5 Tratamento de imagem

O software utilizado e disponibilizado pela empresa para a visualização por computador foi o Halcon. Este sistema foi desenvolvido pela Machine Vision Technology (MVTec). Trata-se de uma excelente ferramenta profissional, tendo sido adquirida pela empresa Renault CACIA no âmbito deste projecto. Esta ferramenta inclui uma biblioteca com uma vasta variedade de operadores para processamento de imagem, bem como toda a interface entre dispositivos de visão artificial. As soluções desenvolvidas no software possuem uma linguagem script suportada pelo seu próprio IDE gráfico denominado HDevelop. Uma das mais valias da referida ferramenta, é a possibilidade de exportar o projeto desenvolvido para outras linguagens de programação, tais como C, C++, C# ou VisualBasic. O próprio Halcon possui vários programas exemplos a explorar que são um excelente ponto de partida para uma rápida aprendizagem das soluções e algoritmos existentes, que podem ser úteis para o projeto. As etapas para instalação do software encontram-se no apêndice B

3.5.1 Incorporação do algoritmo de inspeção com obtenção de imagens através da câmara

Com vista a um ensaio da integração do sistema de avaliação de qualidade desenvolvido no software Halcon, e o software *Sapera CamExpert* desenvolvido para aquisição de imagem a partir da câmara, foi desenvolvido um programa em C#.

Na figura 3.7 é demonstrada a execução do programa C#. Para iniciar a captura de uma imagem, o programa necessita de receber um *input* da câmara. No passo seguinte, a conexão com o dispositivo é indispensável, sendo que, em caso de falha, é gerida uma mensagem de erro e, consequentemente, o sistema volta à fase inicial do programa. Caso contrário são configurados os parâmetros definidos da câmara, como a exposição e o ganho, e procede-se à captura da imagem. Uma vez obtida a imagem, esta é convertida em formato que viabilize uma leitura correta e o seu armazenamento numa pasta e, uma vez chegado ao seu fim de ciclo, a conexão inicial é desfeita.

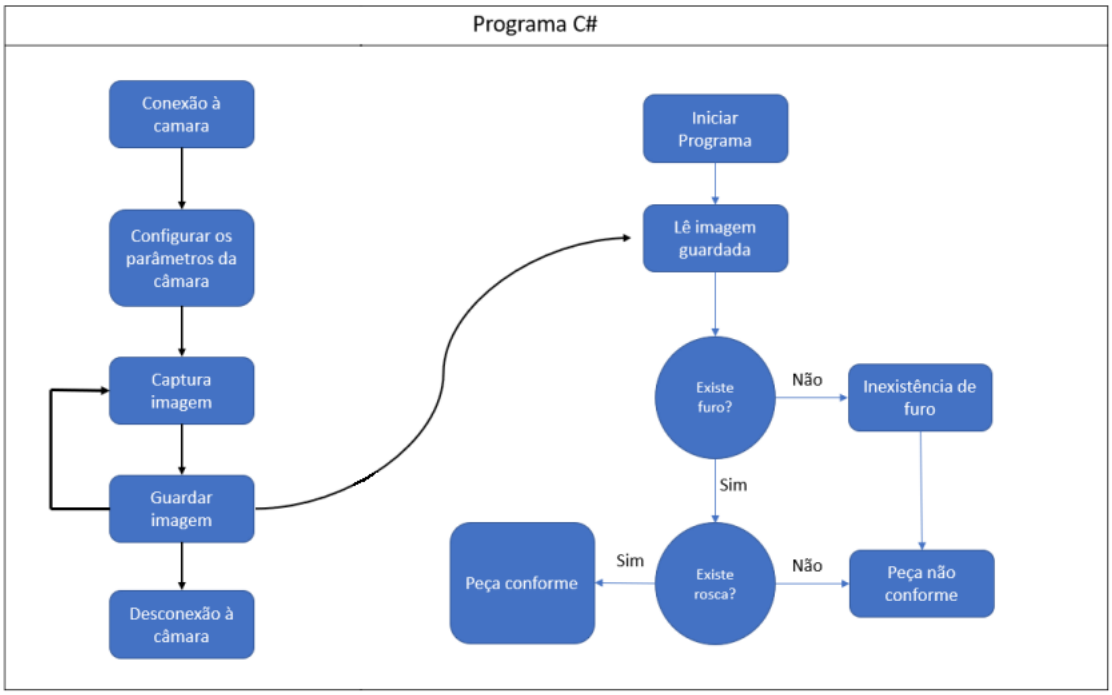


Figura 3.7: Funcionamento geral do programa

O processo de avaliação tem o seu início com a importação da imagem. A imagem é guardada numa pasta descrita no processo de captura, sendo depois utilizada para importação e submetida ao processo de identificação de furo, e só depois para a deteção de rosca. Iniciado o processo de identificação, caso não se verifique a presença de rosca na imagem, o ciclo do programa é encerrado com uma mensagem, indicando que não foi identificado nenhum módulo idêntico ao da imagem capturada.

3.5.2 Desenvolvimento do algoritmo *Template Matching Shape-Based* no Halcon

Com as ferramentas disponíveis no Halcon, foi utilizado um template matching que foi julgado relevante para o projeto nesta fase do protótipo. Foi conseguida a obtenção, através da imagem modelo do furo roscado, de imagens que mais se assemelham ao modelo obtido através das capturas que a câmara efetuou, escolhendo no momento qual o grau de confiança de todos os carateres reconhecidos. Existem alguns tipos de algoritmos de *template matching* que o Halcon disponibiliza.

O programa desenvolvido foi com base no algoritmo *template matching Shape-based*, um algoritmo que combina a forma do objeto com o modelo, em vez dos valores de cinza do pixel. Este método permite a descrição de um modelo, usando formas de contorno e outras restrições que podem ser adicionadas às arestas vizinhas, para aumentar a precisão do modelo. O conceito geral deste método (Figura 3.8) consiste nestas etapas principais:

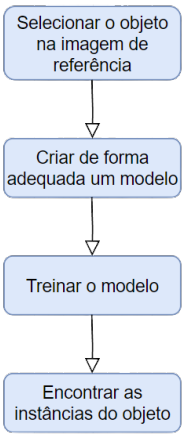


Figura 3.8: Etapas principais do *template matching Shape-Based*

Etapa 1 - Selecionar o objeto na imagem referência

A primeira etapa envolveu a captura de imagem de referência (cárter) e a criação de uma região que contenha o objeto de interesse (região do furo). O objeto servirá como modelo de treino para a etapa 3 do processo (Listagem 3.1).

```
* Matching 01: Obtain the model image
read_image (Image_Cima, 'C:/Users/Pombeiro/Desktop/halcon/Inspecao_sensor/Modelo/
    roscacima.png')

* Matching 01: Build the ROI from basic regions
gen_circle (ModelRegion_Cima, 438.401, 875.462, 59.0809)

* Matching 01: Reduce the model template
reduce_domain (Image_Cima, ModelRegion_Cima, TemplateImage_Cima)
```

Listagem 3.1: Criação da região onde contém o objeto de interesse

Etapa 2 - Criar de forma adequada um modelo

O modelo do *template matching* é criado através do operador "*create_generic_shape_model*", que gera um identificador para o modelo recém criado (ModelID). Nesta etapa, os parâmetros que modificam o ModelID devem ser definidos usando a função parâmetro "*set_generic_shape_model_param*" (Listagem 3.2). Caso se altere um valor correspondente, terá que se requerer um novo treino do modelo ajustado antes da correspondência.

```
create_generic_shape_model (ModelID_Cima)
* Matching 01: set the model parameters
set_generic_shape_model_param (ModelID_Cima, 'contrast_high', 78)
set_generic_shape_model_param (ModelID_Cima, 'contrast_low', 11)
set_generic_shape_model_param (ModelID_Cima, 'metric', 'use_polarity')
```

Listagem 3.2: Criação do modelo

Etapa 3 - Treinar o modelo

Para detetar o objeto de interesse (furo roscado) com base no modelo fornecido, o modelo criado teve de ser treinado usando o operador "*train_generic_shape_model*". Esta etapa permitiu que o modelo aprendesse os recursos e características específicas do padrão de treino.

Etapa 4 - Encontrar as instâncias do objeto

Para localizar as instâncias do modelo treinado em uma imagem foi utilizado o operador "*find_generic_shape_model*" (Listagem 3.3). Foi possível otimizar a busca por uma instância do modelo ajustando os parâmetros usando a função parâmetro "*set_generic_shape_model_param*", não sendo, para tal, requerido treino do modelo. Porém, se esses parâmetros fossem definidos por um valor não estimado para um valor que leve à estimativa automática durante o "*train_generic_shape_model*", seria necessário um novo treino do modelo.

```
grab_image_start (AcqHandle1, -1)
grab_image_async (Image_Cima_Foto, AcqHandle1, -1)
dev_display (Image_Cima_Foto)

* Matching 01: Set the search paramaters
set_generic_shape_model_param (ModelID_Cima, 'border_shape_models', 'false')
* Matching 01: The following operations are usually moved into
* Matching 01: that loop where the acquired images are processed

* Matching 01: Find the model
find_generic_shape_model (Image_Cima_Foto, ModelID_Cima, MatchResultID_Cima,
    NumMatchResult_Cima)
```

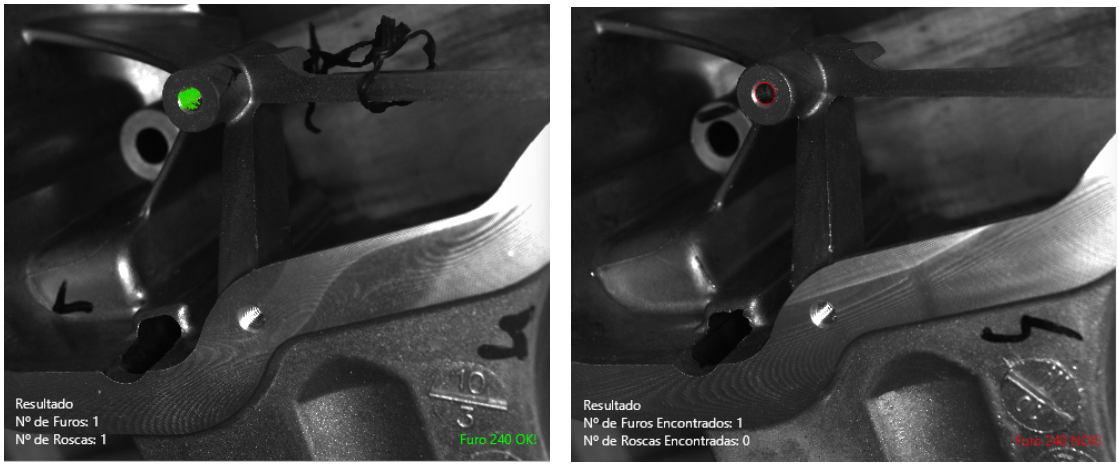
Listagem 3.3: Criação do modelo

Com o programa em funcionamento, uma vez que o software possui uma linguagem própria (HDevelop), exportou-se o programa, convertando-a para a linguagem C# para depois ser introduzida no programa do *Visual Studio*. A figura 3.9 demonstra as instâncias encontradas do modelo treinado tanto para o furo roscado, como também para o furo sem rosca. As etapas para a criação dos modelos do uso deste algoritmo encontram-se apresentadas na apêndice C.

3.6 Comunicação

Nesta secção, é apresentada a implementação de um programa de comunicação com o robot UR10 e o programa de visão. Para a realização deste, optou-se pelo que seria útil pensar e planear, de forma autónoma, aprofundando assim o conhecimento e a perceção do funcionamento da comunicação por TCP/IP entre autómato e o programa de visão, e a comunicação I/O entre o autómato e o robot UR10.

O software Halcon possui algumas ferramentas de comunicação por via TCP/IP através de sockets com o autómato, sendo o programa como servidor e o autómato como cliente. Inicialmente foi testado o desenvolvimento deste tipo de comunicação entre programa Halcon e autómato abdicando da necessidade de exportar o programa Halcon e



(a) Furo 240 "conforme" (b) Furo 240 "não conforme"

Figura 3.9: Resultado do programa desenvolvido

colocá-lo no programa em *Visual Studio* com as suas bibliotecas de comunicação. Todo esse processo apresenta-se no apêndice B, onde foi utilizado o software *Hercules* para testar uma comunicação separada do programa Halcon com o autómato inserindo o software.

3.6.1 Programa autómato

O programa foi desenvolvido para um autómato da Siemens, do modelo S7-1200, com o software da Siemens TIA Portal v15. Este autómato foi responsável pelo controlo das posições do robot, cujo programa inclui lógica para que o robot se mova com segurança e eficiência. Para além desta função, também controla as mensagens provenientes do programa em *Visual Studio*.

Nas funções e zonas de memória reservadas, foram definidas as entradas e saídas digitais necessárias para a realização das funções, sendo estas nada mais do que os *inputs* e os *outputs* do robot UR10 (*tag*), *trigger* da câmara e da *ring light*. Desenvolveu-se a criação de uma tabela de *Tags*, onde é atribuída a denominação das entradas e saídas (Figura 3.10) que se pretendem, e as variáveis definidas no *Data Block* (Figura 3.11) para leitura.

Posicao_robot

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...
1	In_Camera	Bool	%Q0.0	☐	☑	☑	☑
2	Pos_Inic_Robot	Bool	%I0.1	☐	☑	☑	☑
3	Pos_Final_Robot	Bool	%I0.2	☐	☑	☑	☑
4	Pos_origem	Bool	%I0.0	☐	☑	☑	☑
5	Ring_Light	Bool	%Q0.1	☐	☑	☑	☑
6	posicao_1	Bool	%Q0.2	☐	☑	☑	☑
7	posicao_2	Bool	%Q0.3	☐	☑	☑	☑
8	posicao_origem	Bool	%Q0.4	☐	☑	☑	☑
9	Start	Bool	%M60.0	☐	☑	☑	☑

Figura 3.10: Variáveis de entradas e saídas

DB_Snap7

	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...
1	Permissao_Origem	String	768.0	''	☐	☑	☑	☑
2	Permissao_1	String	1024.0	''	☐	☑	☑	☑
3	Permissao_2	String	1280.0	''	☐	☑	☑	☑

Figura 3.11: Variáveis do Data Block

Seguidamente procedeu-se ao desenvolvimento das funções e as suas interligações com as variáveis criadas no *Data Block*. O grafcet da figura 3.12, explica o envio de mensagens na fase em que o robot está em posição. O programa fica à espera das entradas digitais vindas do robot UR10 (I0.0 I0.1 e I0.2), para que depois as variáveis do tipo *string* do *DataBlock Snap7* (Permissao_1 e Permissao_2) sejam escritas com mensagens específicas para depois se dar o procedimento da captura e processamento de imagem. Finalmente será ativada uma saída digital para o UR10 para prosseguir para a próxima etapa e simultaneamente limpa a *string* da variável.

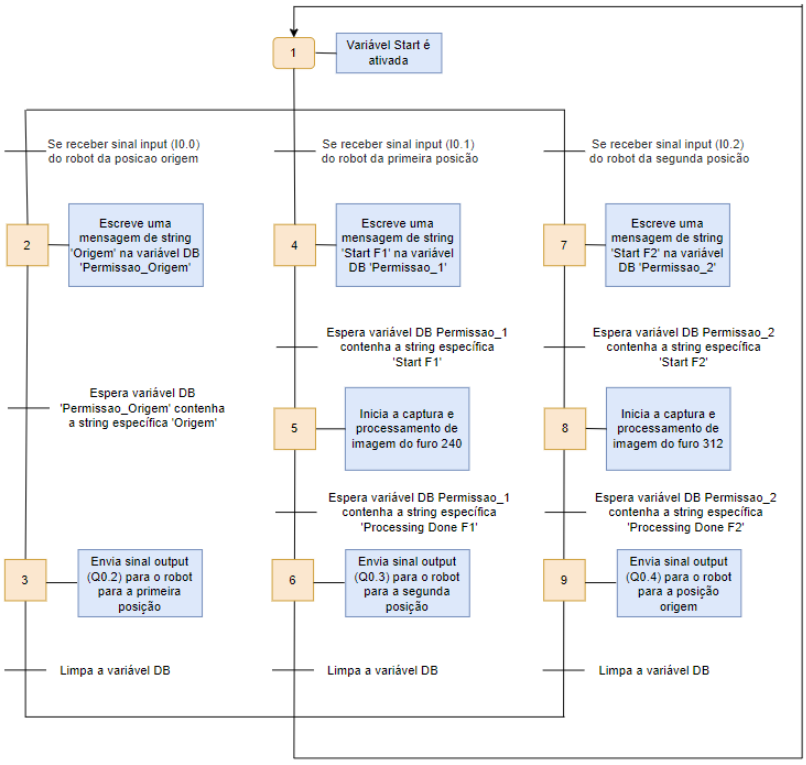


Figura 3.12: Grafcet relativamente ao envio de mensagens dos inputs da posição do robot para dar ao procedimento de captura e processamento de imagem e depois o envio de sinal *output* de volta para o robot

O Grafcet a seguir (Figura 3.13), demonstra o processo da captura de imagem, isto é, o acionar do *trigger* da câmara, sendo que a ativação da *ring light* só funcionam quando o robot se encontra nas posições 1 e 2. Ou seja, o programa fica à espera da ativação das variáveis inputs (I0.0 e I0.1) vindas do robot.

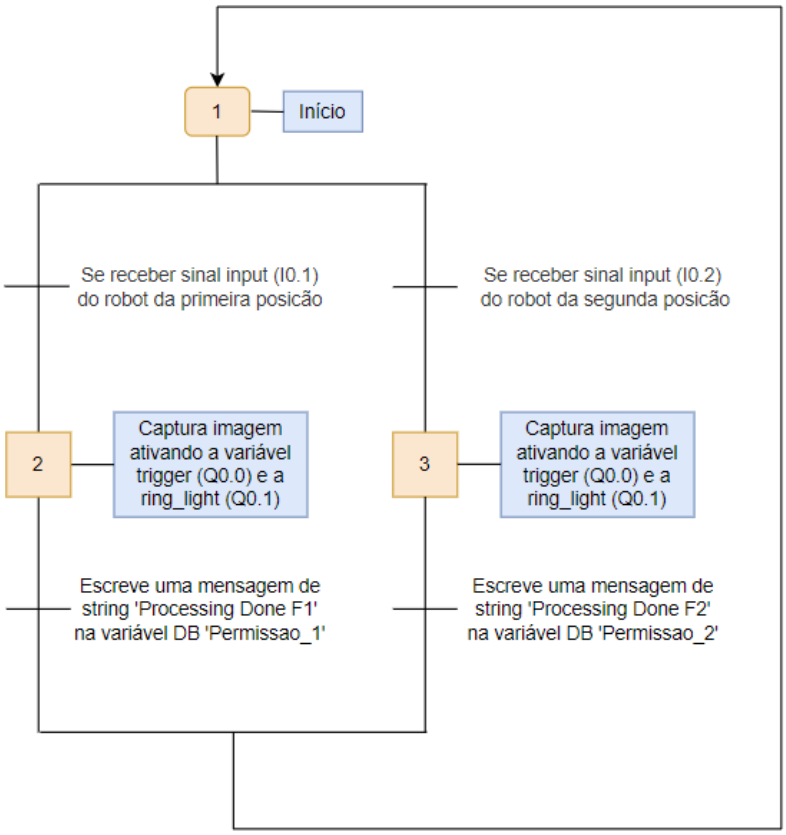


Figura 3.13: Grafcet da captura de imagem

3.6.2 Componente robótico

Um dos aspetos importante no projeto é a posição em que a câmara deve ser instalada para garantir uma boa imagem para a avaliação do furo do cárter. Com o material disponível da empresa, optou-se por utilizar um robot UR10, uma ferramenta versátil, eficiente para a automação industrial e projetado para controlar as suas posições e interagir com o autómato.

O seu software de programação, denominado "*Polyscope*", fornece uma interface intuitiva e amigável, que permite manualmente ao utilizador criar movimentos complexos e integrá-lo a dispositivos por I/O como o autómato.

A figura 3.14 demonstra o programa desenvolvido no software do robot, usaram-se as funções *Waypoints*, que representam a posicão em que o robot irá ficar. Os sinais de saída *outputs* do robot irão conectar-se às entradas digitais *inputs* do autómato (I0.0, I0.1 e I0.2) e vice-versa (Q0.0, Q0.1 e Q0.2). Ou seja, estas tarefas só serão executadas pelo robot de acordo com as instruções recebidas do PLC S7-1200.

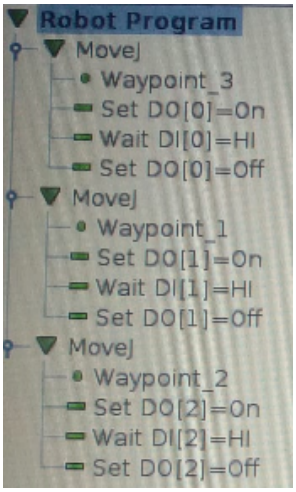


Figura 3.14: Programa desenvolvido no robot UR10

3.6.3 Programa *Visual Studio* e protocolo S7

Integrar o código do programa Halcon no Visual Studio foi uma grande vantagem para ter uma comunicação aprimorada com o PLC. Existem bibliotecas específicas para a comunicação PLC necessárias, tais como a comunicação em tempo real, a transferência eficiente de dados e o tratamento de erros, levando a uma comunicação fiável e eficiente entre o programa e o PLC.

O protocolo S7 é comumente usado para sistemas de controlo como este, e a capacidade de ler e manipular variáveis é bastante útil. Usou-se uma biblioteca denominada *S7netplus* que fornece uma maneira amigável e eficiente de interagir com o autómato. Inicialmente fez-se a conexão do programa do *Visual Studio* ao autómato através do seu IP.

Seguidamente selecionaram-se as váriáveis para manipular, por forma a que, quando o tratamento de imagem fosse concluído, a variável manipulada, neste caso as variáveis *Tag_1* (M50.0), *Tag_2* (M50.1) e *Tag_3* (M50.2), iria dar passagem para a próxima posição do robot. Por exemplo, a listagem 3.4 apresenta como as variáveis são manipuladas através da função da biblioteca *plc.Write()*. A função *plc.Read()* faz a leitura do estado da variável. No caso da listagem, fez-se a leitura das variáveis das posições a que o robot se encontra.

```
1  public Form1()
2  {
3      InitializeComponent();
4      plc=new Plc(CpuType.S71200, "169.254.9.1", Convert.ToInt16(0), Convert.ToInt16(1));
5      plc.Open();
6  }
7  private void timer1_tick(object sender, EventArgs e)
8  {
9      bool pos0 = (bool)plc.Read("I0.0");
10     bool pos1 = (bool)plc.Read("I0.1");
11     bool pos2 = (bool)plc.Read("I0.2");
12     if (pos0 == true & pos1== false & pos2 == false)
```

```
13      {
14          plc.Write("M50.0", true);
15      }
16      if (pos0== false & pos1 == true & pos2 == false)
17      {
18          plc.Write("M50.1", true);
19      }
20      if (pos0== false & pos1 == false & pos2 == true)
21      {
22          plc.Write("M50.2", true);
23      }
24  }
```

Listagem 3.4: Leitura e manipulação de variáveis do autómato

3.6.4 Interface do programa

No que diz respeito à interface, o programa desenvolvido possui uma janela (Figura 3.15), através da qual o utilizador dá início à inspeção dos furos no botão "*Start*", ordenando a deslocação do UR10 para a posição 1. Por sua vez dá início à captura de imagem do primeiro furo (Furo 240) para ser avaliado mais tarde e prossegue para a posição 2 para efetuar o mesmo processo. O utilizador tem a possibilidade, não só de visualizar as imagens resultantes da identificação de rosca e os seus dados, como também observar o registo histórico de peças que foram controladas e que ficam guardadas num diretório do computador industrial na base de dados.

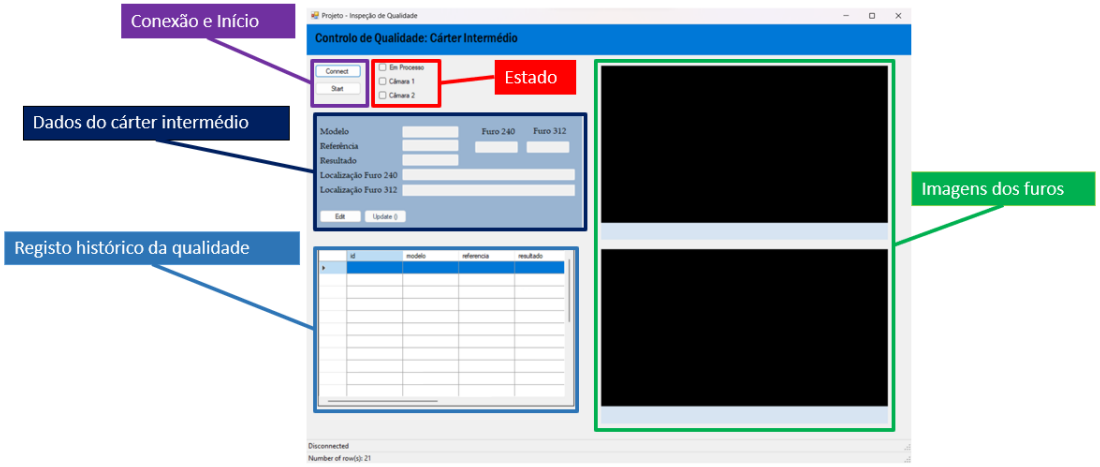


Figura 3.15: Interface visual do programa no VisualStudio

3.6.5 Ligação da base de dados *PostGreSQL* ao programa *Visual Studio*

Para fazer a ligação da base de dados *PostGreSQL* para o *Visual Studio*, em primeiro lugar fez-se a configuração da base de dados. Seguidamente, no *Visual Studio*, criou-se uma classe "*CRUD*"(Listagem 3.5) e colocou-se o endereço do servidor, número da porta,

nome da base de dados e o nome do utilizador e senha fornecidos na configuração (Linha 13-17).

```
1  using Npgsql;
2
3  namespace PostGres_demonstracao
4  {
5      class CRUD
6      {
7          public static NpgsqlConnection con = new NpgsqlConnection(getConnectionString())
       ;
8          public static NpgsqlCommand cmd = default(NpgsqlCommand);
9          public static string sql = string.Empty;
10
11         private static string getConnectionString()
12         {
13             string host = "Host=localhost;";
14             string port = "Port=5432;";
15             string db = "Database=Teste_Carter;";
16             string user = "Username=postgres;";
17             string pass = "Password=1234;";
18
19             string conString = string.Format("{0}{1}{2}{3}{4}", host, port, db, user,
       pass);
20             return conString;
21         }
22     }
23 }
```

Listagem 3.5: Classe *CRUD* para a conexão na base de dados *PostGreSQL*

3.7 Conclusões obtidas na realização do protótipo

Feito o teste do protótipo entre câmara, autómato e o programa de avaliação foi possível concluir que a câmara utilizada para a realização do mesmo, possuía as características necessárias para a obtenção de imagens com a qualidade desejada para a deteção da ausência de furos roscados das peças.

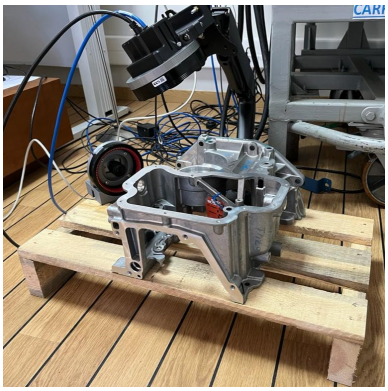
No que toca à parte da comunicação entre programa-autómato e autómato-robot revelou-se uma ótima solução a ser implementada no chão de fábrica, no entanto, as restrições e regras a serem respeitadas na empresa, no que toca à utilização do robot são bastante restritas e para um projeto com estas dimensões são pouco favoráveis. Caso fosse necessário inspecionar outras partes da peça para além dos dois furos, esta solução seria bastante eficaz com a participação do robot.

Durante o período de estágio, ocorreu um incidente com robot, por motivos de mau uso, e a comunicação do uso da biblioteca *S7netplus* deixou de funcionar, o que impediu o progresso do projeto. Posto isto, consequentemente surgiu a necessidade de partir de imediato para a segunda solução proposta desdita no capítulo anterior, ou seja, o uso de duas câmara fixas, não obstante a necessidade de se desenvolver outro programa na identificação de furos roscados, desta vez no uso de ML com o algoritmo SVM, embora o algoritmo desenvolvido na proposta anterior se revele bastante útil para combinar

objetos que possuam deformações, oclusões ou até mesmo nas variações das condições de iluminação. A implementação deste segundo sistema é economicamente viável e menos complexo de compreender o funcionamento e o tempo de ciclo de inspeção muito menor.

3.8 Solução final

Substituiu-se a configuração existente por um sistema mais simples e avançado que incorpora as duas câmaras fixas (Figura 3.16a). Uma vez que a empresa só tinha disponível uma *EFFI-Ring*, utilizou-se outra iluminação de cor vermelha para acoplar às duas câmaras a qual são as LDR2-120RD-WD (Figura 3.16b), projetadas para fornecer iluminação uniforme e sem sombras ao redor da lente da câmara, com níveis de intensidade ajustáveis para controlar o brilho da luz do anel.



(a) Novo *setup* experimental

(b) Emissor de luz LDR2-120RD-WD

Figura 3.16: Novo sistema de visão

3.8.1 Implementação de uma estrutura caixa negra

No chão de fábrica, a linha encontra-se numa área onde a peça está exposta à luz ambiente. Perante este fator, foi necessário planear uma estrutura de caixa negra capaz de impedir a interferência da luz solar, uma vez que será excluída a instalação de um braço robótico, de forma a que as câmaras consigam capturar imagens com uma iluminação fixa vinda dos *LED's*. Através do uso do *software SolidWorks*, conseguiu-se desenvolver um CAD de uma estrutura simples de calhas com placas, cuja cor transparente, presente na figura 3.17, na realidade apresenta uma cor escura para impedir a passagem de luz na zona da inspeção.

3.8.2 Reestruturação da comunicação

A exclusão do robot permitiu uma comunicação mais eficiente e menos complexa entre o computador e o autómato. Necessitou apenas das variáveis de saída (Figura 3.18), *trigger_ camara* e *Ring_ Light*, e das variáveis definidas no Data Block (Figura 3.19). Ou seja, reduziu-se para duas variáveis booleanas (*Inicio* e *Processo*) para dar início ao processo, e um inteiro (*Camara*), para dar o *trigger* da câmara correspondida.

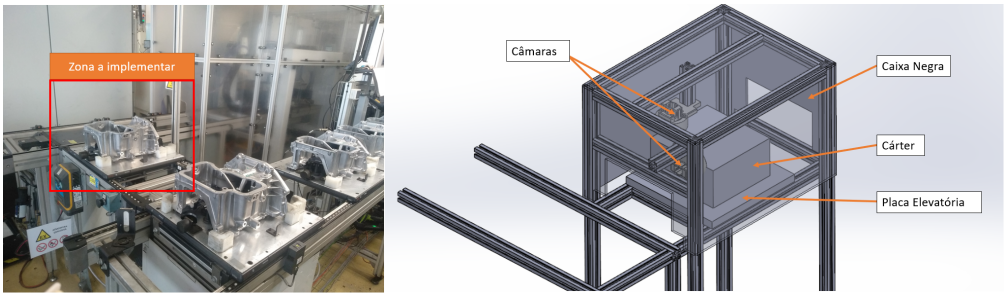


Figura 3.17: Estrutura CAD em *SolidWorks* a implementar no atelier 6 na linha 3

Posicao_robot

	Name	Data type	Address	Retain	Acces...	Write...	Visibl...
1	In_Camera	Bool	%Q0.0	☐	☑	☑	☑
2	Pos_Inic_Robot	Bool	%I0.1	☐	☑	☑	☑
3	Pos_Final_Robot	Bool	%I0.2	☐	☑	☑	☑
4	Pos_origem	Bool	%I0.0	☐	☑	☑	☑
5	Ring_Light	Bool	%Q0.1	☐	☑	☑	☑
6	posicao_1	Bool	%Q0.2	☐	☑	☑	☑
7	posicao_2	Bool	%Q0.3	☐	☑	☑	☑
8	posicao_origem	Bool	%Q0.4	☐	☑	☑	☑
9	Start	Bool	%M60.0	☐	☑	☑	☑

Figura 3.18: Variáveis de entradas e saídas

Sharp7

	Name	Data type	Offset	Start value	Retain	Accessible f...	Write...	Visible in ...	Setpoint
1	Static				☐	☐	☐	☐	☐
2	Start	Bool	0.0	false	☐	☑	☑	☑	☐
3	Em_Processo	Bool	0.1	false	☐	☑	☑	☑	☐
4	Trigger_Camara	Int	2.0	0	☐	☑	☑	☑	☐

Figura 3.19: Variáveis do *Data Block*

O *grafcet* a seguir (Figura 3.20), demonstra o processo de captura de imagem semelhante ao da figura 3.13, desta vez o uso da variável inteira *Camara* permitiu escolher a câmara que iria dar o *trigger* para a captura de imagem.

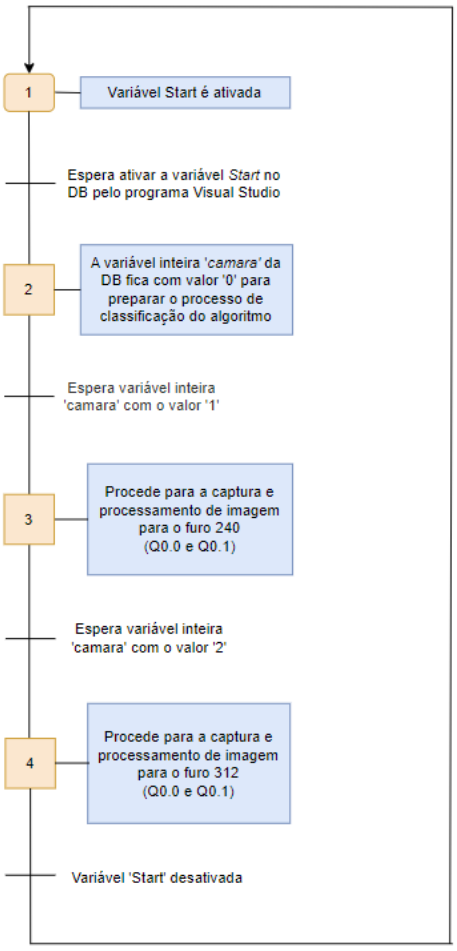


Figura 3.20: *Grafcet* da captura de imagem para a segunda solução

3.8.3 Biblioteca Sharp7

Não havendo solução para corrigir o problema da biblioteca anteriormente utilizada, teve-se que recorrer a uma outra biblioteca com os mesmos princípios, o *Sharp7*. Desenvolveu-se outro código para ler e gravar os dados no *Data Block "Sharp7"*. A listagem 3.6 demonstra uma parte do código que lê o conteúdo dos primeiros 4 bytes do *Data Block 1* ("*Sharp7*") e extrai o valor inteiro na posição 2 e o valor booleano do primeiro bit (*Start*) na posição 0. Dependendo dos valores dessas variáveis, o código modifica o conteúdo do *Data Block* (*plc.DBRead()*) e a escrita de volta no PLC (*plc.DBWrite()*).

Por exemplo, se o valor na posição 2 (variável *Trigger Camara*) for 0 e o primeiro *bit* na posição 0 (variável *Start*) for verdadeiro, o código define o primeiro *bit* como falso, define o valor na posição 2 como 1 e grava o *Data Block* de volta para o PLC.

```
1 private void timer1_Tick(object sender, EventArgs e)
2 {
3     var buffer1 = new byte[4];
4     int readresult = plc.DBRead(1, 0, buffer1.Length, buffer1);
5     int value = S7.GetIntAt(buffer1, 2);
```

```
6       bool start = S7.GetBitAt(buffer1, 0, 0);
7       if ((value == 0) && (start == true))
8       {
9           S7.SetBitAt(buffer1, 0, 0, false);
10          S7.SetIntAt(buffer1, 2, 1);
11          int writeresult = plc.DBWrite(1, 0, buffer1.Length, buffer1);
12      }
13      if (value == 1)
14      {
15          S7.SetIntAt(buffer1, 2, 2);
16          int writeresult = plc.DBWrite(1, 0, buffer1.Length, buffer1);
17      }
18      if (value == 2)
19      {
20          S7.SetIntAt(buffer1, 2, 0);
21          S7.SetBitAt(buffer1, 0, 1, false);
22          int writeresult = plc.DBWrite(1, 0, buffer1.Length, buffer1);
23      }
24  }
```

Listagem 3.6: Leitura e manipulação de variáveis utilzando a biblioteca *Sharp7*

3.8.4 Algoritmo SVM no Halcon

A decisão de utilizar este algoritmo foi com base num arquivo tutorial [7] que forneceu instruções abrangentes e acessíveis sobre a sua implementação. O tutorial revelou-se um recurso valioso para entender as complexidades do algoritmo e a sua adequação para tarefas de classificação de imagens.

Passando para a classificação de características, isto é, a criação de um classificador (Classifier) com as propriedades específicas. A figura 3.21 explica os passos gerais de uma classificação.

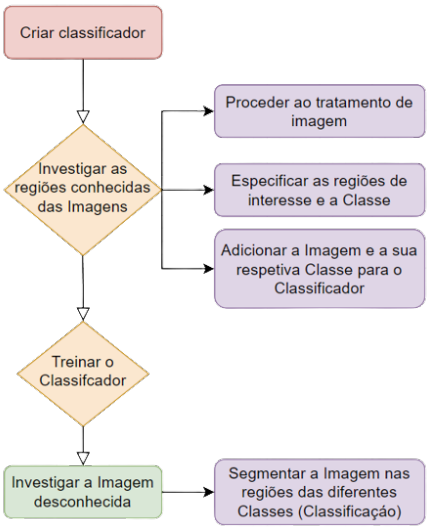


Figura 3.21: Diagrama de funcionamento da classificação SVM

A primeira etapa deste processo foi investigar objetos conhecidos, onde as características dos objetos com classes conhecidas são extraídas e adicionadas ao classificador como os vetores de características junto com os seus *IDs* das classes correspondentes. Por meio de treino, o classificador dirige as regras de classificação, capacitando-o a distinguir as diferentes classes.

Um classificador com base em duas classes através da regra da decisão seguinte, em que para cada instância z tem-se:

$$\dot{g}(\mathbf{z}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{z}) + b \Rightarrow \begin{array}{ll} g(\mathbf{z}) > 0 & y = 1 \\ g(\mathbf{z}) < 0 & y = -1 \end{array} . \tag{3.1}$$

Pelo treino, os parâmetros do classificador determinados são: *alpha*, vetores de suporte *xi* (elementos de conjunto de treino associados aos valores Langragianos de *alpha*) e as respetivas etiquetas *yi* e b. Na fase de testes, o classificador necessita do vetor suporte e dos valores Langragianos [24]. K(x,z) representa a função de kernel que são do tipo:

- **Kernel linear -** A superfície de separação no espaço multidimensional é um hiperplano. O kernel (Equação (3.2)) define o produto interno no espaço de entrada

$$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z} \tag{3.2}$$

- **Kernel RBF -** A superfície se separação no espaço multidimensional da entrada é não linear. A função de kernel (Equação (3.3)) define o produto interno no espaço determinado pela transformação.

$$K(\mathbf{x}, \mathbf{z}) = \Phi^T(\mathbf{x})\Phi(\mathbf{z}) = \exp\left(\frac{\|\mathbf{x} - \mathbf{z}\|^2}{-2\gamma}\right) \tag{3.3}$$

Para classificar os objetos desconhecidos, o mesmo conjunto das características que foram usadas para o treino é extraído e os vetores de características são classificados com base no classificador treinado.

O programa iniciou-se com a atribuição de classes disponíveis. Os cárteres intermédios são classificados nas classes "*good*" (cárter "conforme") e "*bad*" (cárter defeituoso) ou "*none*" (nenhum cárter encontrado).

Como primeiro passo da classificação real, um classificador SVM foi criado com o operador *"create_class_svm"* (Listagem 3.7). O parâmetro de saída *SVMHandle* foi essencial para todos os operadores específicos da classificação que serão utilizados posteriormente.

```
Nu_Cima := 0.05
KernelParam_Cima := 0.05

*Create a SVM classifier
create_class_svm (7, 'linear', KernelParam_Cima, Nu_Cima,
 |Classnames|, 'one-versus-one', 'principal_components', 5, SVMHandle_Cima)
```

Listagem 3.7: Valor de *Nu* e do *Kernel* usado e a criação do classificador SVM

No uso do SVM, o classificador foi treinado para a aplicação atual, isto é, as regras para classificação devem ser derivadas de um conjunto de amostras. Por exemplo, o treino determina os vetores de suporte ótimos que separam as classes umas das outras.

A classificação é uma tarefa crítica no processamento de imagens que envolve a determinação da associação de classe de vários objetos. Nesse processo, uma amostra é um objeto cuja associação de classe já é conhecida e a classificação requer que um objeto possa ser caraterizado por um conjunto de características ou valores de características. Objetos comuns para o processamento de imagem podem incluir píxeis, regiões, ou uma combinação de ambos. Para os cárteres, serve como uma ilustração, onde as regiões que representam os furos são os objetos que precisam ser treinados e classificados. A amostra de treino para cada objeto conhecido é composta por um vetor de características com valores derivados da região extraída e o nome da classe (conhecida) correspondente. Inicialmente tentou-se obter vetores de características na região do furo, mas as ligeiras mudanças de posição da peça iria perder a informação necessária para a classificação desta, isto é, como a posição da região de interesse é fixa na imagem, é possível que esta não demonstre a região do furo. Portanto, procedeu-se ao tratamento de toda a imagem do cárter para obter um melhor treino do classificador (Figura 3.22).

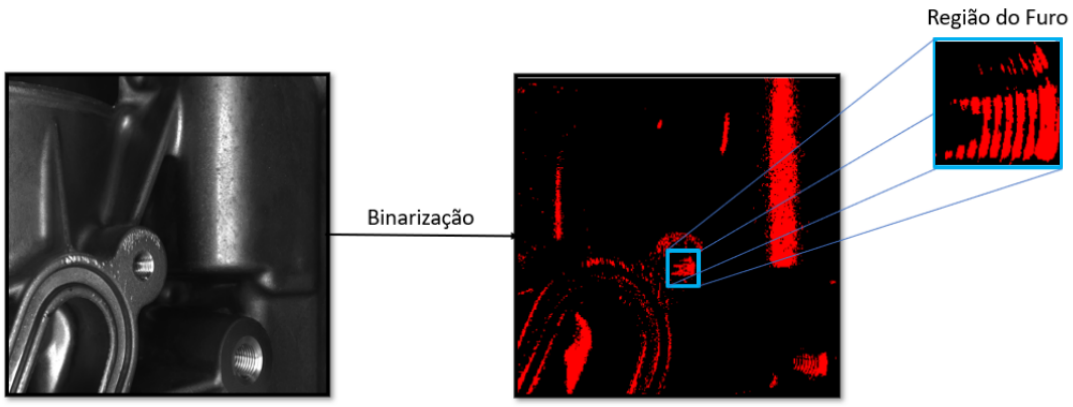


Figura 3.22: Binarização da imagem total do furo 312

Os limites de classe adequados foram estabelecidos, fornecendo um conjunto representativo de amostras de treino para cada classe. Foram adicionadas cerca de 50 amostras de peças conformes, 50 amostras de peças defeituosas e 3 amostras da não existência de peça, para dentro de um procedimento para o processamento de imagem (Listagem 3.8), tanto para o furo 240 (*add_samples_to_svm_lado*) como também para o furo 312

(*add_samples_to_svm_lado*).

```
for ClassNumber := 0 to |ClassNames| - 1 by 1
    list_files (ReadPath_Cima + ClassNames[ClassNumber], 'files', Files)
    Selection := regexp_select(Files,'.*[.]png')
    for Index := 0 to |Selection| - 1 by 1
        read_image (Image, Selection[Index])
        dev_display (Image)
        if (ClassNumber=0)
        gen_circle (ROI_0, 136.094, 129.277, 63.8202)
        reduce_domain (Image, ROI_0, ImageReduced)
        threshold (ImageReduced, Region, 165, 255)
        calculate_features (Region, Features)
        add_sample_class_svm (SVMHandle_Cima, Features, ClassNumber)
        endif
        if (ClassNumber=1)
        gen_circle (ROI_1, 136.094, 129.277, 63.8202)
        reduce_domain (Image, ROI_1, ImageReduced)
        threshold (ImageReduced, Regions, 165, 255)
        calculate_features (Region, Features)
        add_sample_class_svm (SVMHandle_Cima, Features, ClassNumber)
        endif
        if (ClassNumber=2)
        gen_circle (ROI_2, 136.094, 129.277, 63.8202)
        reduce_domain (Image, ROI_2, ImageReduced)
        threshold (ImageReduced, Region, 165, 255)
        calculate_features (Region, Features)
        add_sample_class_svm (SVMHandle_Cima, Features, ClassNumber)
        endif
    endfor
endfor
return ()
```

Listagem 3.8: Operador *"add_samples_to_svm_cima"*

Nesse procedimento, para cada classe são obtidas as imagens correspondentes. Foi utilizado um *tuple* para atribuir o nome da classe para cada imagem (['*good*' '*bad*' '*none*']). Assim, a sequência das imagens e a sequência dos elementos dentro da tuple tinham de corresponder. As imagens de cada classe foram armazenadas num diretório com o nome da classe. Assim, o procedimento *"add_samples_to_svm"* usou os nomes do diretório para atribuir os vetores às classes (Figura 3.8). Por exemplo, as imagens que contêm cárteres "conforme"foram armazenadas no diretório *"good"*. Então, para cada classe, todas as imagens foram lidas do diretório correspondente. Para cada imagem na região do furo do cárter, onde são aplicados os tratamentos de imagem, são extraídas para dentro do procedimento *"calculate_features"* e o vetor foi adicionado juntamente com o ID da classe correspondente ao classificador, utilizando o operador *"add_sample_class_svm"*.

Os vetores de características que foram usados para treinar o classificador e aqueles que foram classificados para os novos objetos, contêm o mesmo conjunto de características. As características (*features*) foram calculadas dentro do procedimento *calculate_features* (Listagem 3.9). Estas características foram predefinadas do programa Halcon. Existem muitas outras características que poderiam ser utilizadas mas requeria algum conhecimento e tempo para entender das quais seria a melhor opção para utilizá-las para obter um melhor resultado.

```
area_center (Region, Area, Row, Column)
compactness (Region, Compactness)
moments_region_central_invar (Region, PSI1, PSI2, PSI3, PSI4)
convexity (Region, Convexity)
Features := real([Area,Compactness,PSI1,PSI2,PSI3,PSI4,Convexity])
return ()
```

Listagem 3.9: operador *"add_samples_to_svm_cima"*

O cálculo das características foram feitas com base:

- **Área da região centro -** O operador *"area_center"* calcula a área e o centro das regiões de entrada. A área é definida como o número de *pixels* de uma região. O centro é calculado como o valor médio das coordenadas de linha ou coluna, respectivamente, de todos os *pixels*.

- **Compacidade -** O operador *"compactness"* calcula a compacidade das regiões de entrada com base na equação:

$$C' = \frac{\mathrm{L}^2}{4F\pi} \tag{3.4}$$

  Com L o tamanho do contorno e F a área da região do fator de forma C, sendo C=max(1,C')

- **Momento da região central -** O operador *"moment_region_central_invar"* calcula os momentos (PSI1, PSI2, PSI3, PSI4) que são invariantes sob translação e transformações lineares gerais.

- **Convexidade -** O operador *"convexity"* calcula a convexidade de cada região de entrada das regiões. O cálculo é feito com base na equação:

$$C = \frac{F_{\mathrm{O}}}{F_{\mathrm{C}}} \tag{3.5}$$

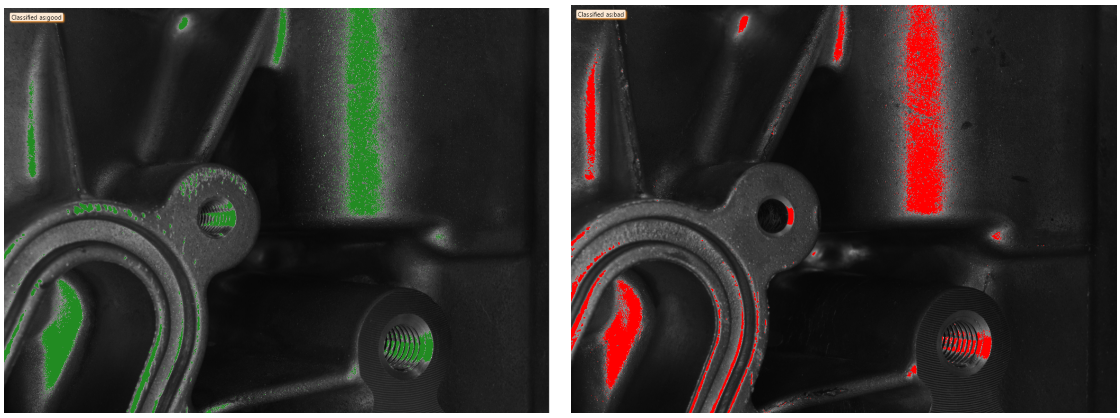
  Sendo F_O a área da convexidade e F_C a área original da região o fator de forma C

Depois de adicionar todas as amostras ao classificador com o procedimento *"add_samples_to_svm_cima"* para o furo 240 e *"add_samples_to_svm_lado"* para o furo 312, o treino é aplicado com o operador *"train_class_svm"* (Listagem 3.10) para ambos os furos. Nesta etapa, o classificador dirige as suas regras de classificação.

```
* Train the classifier
disp_message (WindowHandle_Cima, 'Training...', 'window', 12, 12, 'black', 'true')
train_class_svm (SVMHandle_Cima, 0.001, 'default')
disp_message (WindowHandle_Cima, 'Training completed', 'window', 12, 12, 'black', 'true'
    )
disp_continue_message (WindowHandle_Cima, 'black', 'true')
```

Listagem 3.10: Treino do classificador

Essas regras de classificação foram aplicadas dentro do procedimento "*classify_regions_with_svm_cima*" e "*classify_regions_with_svm_lado*" para as novas imagens captadas que ainda não foram classificadas. O procedimento funcionou de maneira semelhante ao procedimento para adicionar as amostras de treino. Nesse momento, as imagens desconhecidas foram lidas e nenhuma informação da classe estava disponível e, em vez de adicionar amostras ao classificador, o operador "*classify_class_svm*" foi aplicado para classificar a característica desconhecida do vetor com as regras de classificação. A figura 3.23 demonstra os resultados da classificação SVM das imagens desconhecidas, sendo o lado esquerdo da imagem representado a verde considerado como peça "conforme"e no lado direito representado a vermelho, a classificação considerada como "não conforme"da peça.



(a) Furo 312 "conforme" (b) Furo 312 "não conforme"

Figura 3.23: Resultado da classificação SVM

De facto, a implementação deste algoritmo difere da abordagem tradicional de separar explicitamente os conjuntos de treino e teste. O Halcon utiliza apenas os métodos de adicionar amostras e treiná-las para as tarefas de classificação.

Existe uma lista de operadores envolvidos apresentados no apêndice E que explica de forma mais aprofundada, o ajuste de parâmetros específicos que foram necessários para o algoritmo SVM.

Intentionally blank page.

Capítulo 4

Análise de desempenho

Na realização dos testes, foram executadas cerca de 200 fotos tanto para cárteres "conforme", como também para cárteres "não conforme". Foi feito todo o tipo de teste, designadamente com diferentes tons de luminosidade, assim como com ligeiras mudanças de posição do cárter, uma vez que a linha apresenta algumas vibrações, dando a possibilidade deste se encontrar numa posição ligeiramente diferente.

4.1 Resultado da implementação do programa

O funcionamento do programa desenvolvido e anteriormente pormenorizado será posto à prova nesta secção. Aqui se ilustram algumas imagens resultantes da execução do programa (Figura 4.1). Após o programa ser iniciado é aberta a interface visual e o programa conecta automaticamente ao autómato. Caso o programa não se conecte é dado o aviso do programa.

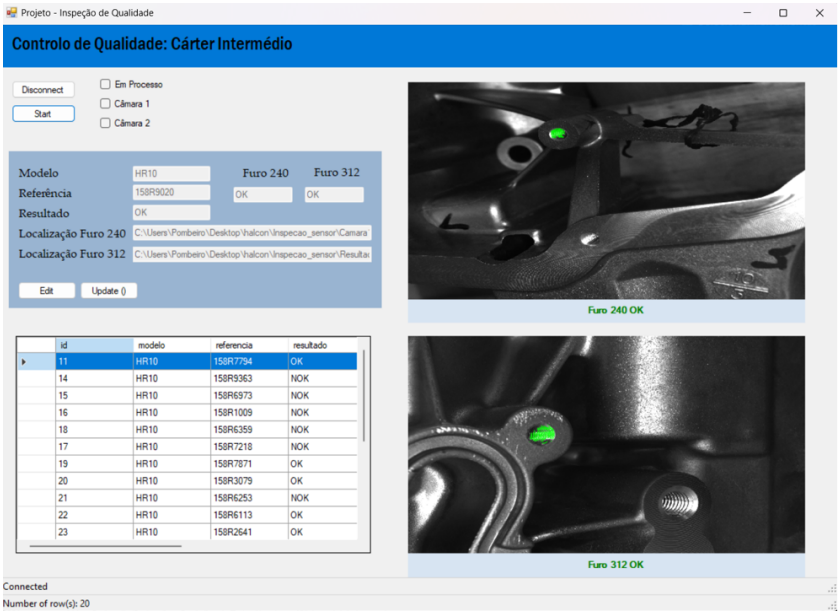


Figura 4.1: Interface do programa no VisualStudio

Devido ao pouco tempo despendido para implementação do protótipo na linha, não foi possível colocar o projeto em prática. A linha seria dotada de um sensor de presença para quando o cárter estivesse posicionado para a inspeção, sendo que aquele daria o sinal para dar início ao processo. Não tendo este material disponível no laboratório, foi substituído por um botão *Start* para simulação.

Feita a inspeção, automaticamente foi enviado para a base de dados PostGres, todas as informações relativas aos dois furos e a interface fica à espera de um novo *input* (botão *Start*) para repetir o processo. Caso o cárter apresente algum defeito (Figura 4.2), é avisado o utilizador para que seja feita uma verificação rápida do furo. Caso não apresente defeito, o utilizador tem a possibilidade de editar, no botão *Edit*, o registo apenas da última verificação feita (1) e alterar para OK (2) e clicar *Update* do registo (3). Caso a última verificação apresente conformidade, o utilizador não tem acesso à edição do registo.

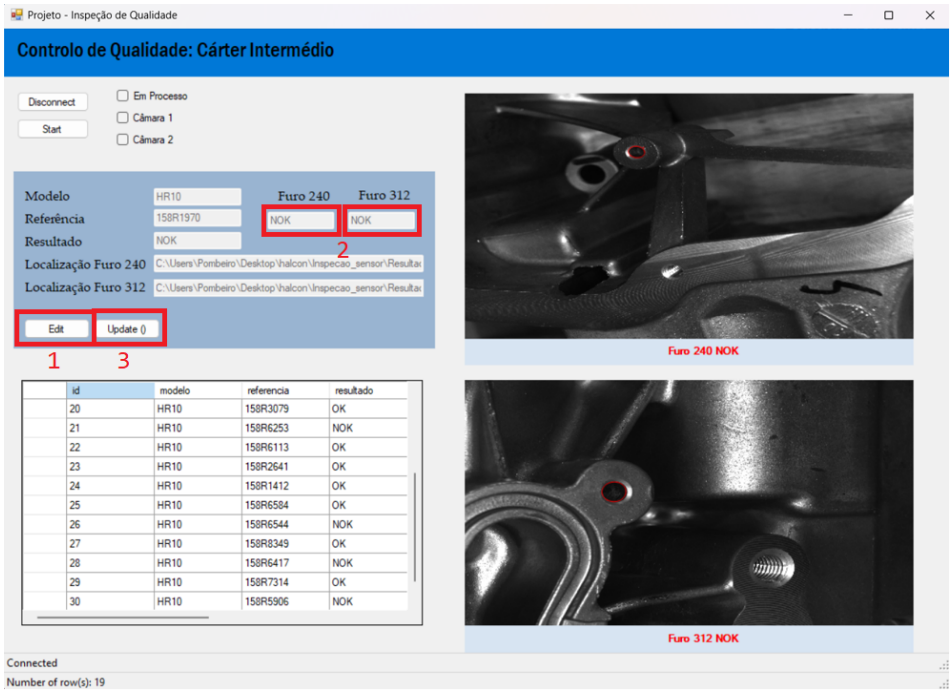


Figura 4.2: Inspeção do cárter com defeito e as etapas do utilizador para proceder à mudança do registo.

4.2 Resultados do algoritmo *Template Matching Shape-Based*

Com base no primeiro algoritmo utilizado, foram analisadas cerca de 200 imagens de peças "conforme", denominadas como imagens verdadeiros positivas (VP), e 200 imagens de peças "não conforme", denominadas como imagens verdadeiras negativas (VN). Durante as experiências, seria provável a existência de uma má classificação das imagens obtendo-se falsos positivos (FP) para cárteres "não conforme" e falsos negativos (FN) para cárteres "conforme". A Tabela 4.1 demonstra os resultados obtidos do algoritmo usado.

Tabela 4.1: Resultados do uso do algoritmo *template matching Shape-Based*

Região	VP	VN	FP	FN	Recuperação	Precisão	Pontuação F1
Furo 240	200	195	0	5	98%	100%	99%
Furo 312	200	182	0	18	92%	100%	96%

A precisão determina uma percentagem de 100%, onde se destaca a capacidade do algoritmo de minimizar os falsos positivos, garantindo a identificação confiável de defeitos (Equação(4.1)). Relativamente à recuperação (Equação (4.1)) é de cerca de 98% e 92% para o furo 240 e 312 respetivamente, o que indica a capacidade do algoritmo de capturar uma parte significativa de verdadeiros positivos (Equação (4.2)).

$$Precisao = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{4.1}$$

$$Recuperacao = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{4.2}$$

Portanto, algoritmo demonstrou um desempenho notável com uma pontuação F1 de 99% geral dos cárteres, sendo 99% para o furo 240 e 96% para o furo 312, mostrando a sua eficácia na deteção dos defeitos com precisão (Equação (4.3)).

$$PontuacaoF1 = 2 \times \frac{Precisao \times Recuperacao}{Precisao + Recuperacao} \tag{4.3}$$

4.3 Resultados do algoritmo *Support Vector Machine*

Foi utilizado um conjunto de dados, designadamente cerca de 50 imagens de peças conforme e 50 imagens de peças defeituosas para treino. Este conjunto de dados foi abrangido pelas variações nas condições de iluminação, orientações e possíveis anomalias. O algoritmo foi treinado para este conjunto e utilizou as técnicas apropriadas da extração e seleção das características, envolvendo a otimização dos parâmetros para o furo 240 (Nu=0,13 e Kernel=0,10) e para o furo 312 (Nu=0,06 e Kernel=0,13) para alcançar o melhor desempenho. Os resultados experimentais demonstraram uma enorme precisão, nomeadamente um valor de 97% para o furo 240 e 96% para o furo 312, tendo como resultado geral para o cárter 96%. Adicionalmente, o resultado da recuperação foi de 97% para o furo 240 e 97% para o furo 312 (Tabela 4.2).

Ou seja, um grande conjunto de dados para o treino facilita a capacidade de aprendizagem do algoritmo em generalizar bem os padrões de defeitos invisíveis e distinguir entre as instâncias defeituosas e não defeituosas. A seleção cuidada e a extração das características relevantes das imagens influenciou muito o desempenho do algoritmo, sendo capaz de classificar com precisão os defeitos, mesmo na presença de variações complexas

Tabela 4.2: Resultados do uso do algoritmo *template matching Shape-Based*

Região	VP	VN	FP	FN	Recuperação	Precisão	Pontuação F1
Furo 240	193	194	7	6	97%	97%	97%
Furo 312	192	190	8	10	95%	96%	96%

de fundo ou ruído.

O ajuste dos parâmetros desempenhou um papel crucial na otimização do desempenho do algoritmo. Ao selecionar a função do *kernel* apropriada (Kernel linear) e os hiperparâmetros de ajuste, permitiu obtido uma melhor precisão para a deteção de defeitos.

Capítulo 5

Conclusão

5.1 Considerações finais

O presente projeto teve como objetivo apresentar uma solução, desenvolvida para o controlo de qualidade automática, em ambiente industrial, de peças maquinadas, obtidas através do processo de maquinação, recorrendo, para tal, a técnicas de visão artificial e processamento de imagem. A classificação em causa teve como objetivo a distinção entre aquilo que eram consideradas peças "conformes" e "não conformes", por forma a validar uma determinada peça e armazenar um histórico de informação para posterior averiguação das causas que se encontravam na origem do defeito detetado.

No que toca ao *hardware* desenvolvido, foram identificadas algumas das condicionantes subjacentes a qualquer projeto que envolva a visão artificial, nomeadamente a iluminação e a posição do dispositivo de aquisição de imagem utilizado. No entanto, a substituição do braço robótico UR10 para suportes fixos foi uma mudança repentina, e a solução que se revelava economicamente mais viável para a aplicação na linha, uma vez que a instalação do robô implicaria a sua submissão a vários testes de aprovação e de certificação para que fosse validada a sua entrada no chão da fábrica. Não obstante, as limitações já referidas, também os incidentes ocorridos com o braço robótico, a dificuldade em satisfazer os pedidos de material para desenvolver uma segunda solução revelaram-se impeditivos de colocar o projeto em prática. Ainda assim, para além das oposições já descritas e não menos impeditivo de concluir o projeto, já na parte final do período em que decorreu o estágio, surgiu a opção, por parte da empresa, de avançar para um projeto bastante mais abrangente. O referido projeto inclui várias alterações às várias funcionalidades da linha, designadamente a integração de múltiplas funções de inspeção de qualidade das peças, entre as quais a funcionalidade proposta por este projeto, mantendo o mesmo objetivo e com a sua implementação no lugar ora proposto.

Em relação ao software, este permitiu a classificação de uma determinada peça como "conforme"ou "não conforme", tanto no uso do *Template Matching Shape-Based* como também o SVM. Ambos os algoritmos apresentraram excelentes resultados, demonstrando a sua eficácia na aplicação específica.

O algoritmo *template matching Shape-Based* exibiu uma precisão excecional, uma vez que a sua capacidade de capturar características e padrões geométricos específicos,

associados tanto a furos roscados como a furos sem rosca, permitiu uma deteção precisa e um mínimo de falsos negativos.

Por outro lado, o algoritmo SVM também se mostrou altamente eficaz. Na abordagem do *machine learning*, utilizou um modelo treinado para classificar os furos roscados com base nos recursos extraídos das imagens. Por isso, demonstrou a sua capacidade de generalizar bem as instâncias não vistas, distinguindo, com eficiência, entre furos roscados e furos sem rosca, com o mínimo de erros.

Ambos os algoritmos têm os seus pontos fortes e podem fornecer resultados confiáveis na deteção da presença de rosca. No entanto, considerando os requisitos específicos para esta aplicação, o algoritmo *template matching Shape-based* pode ser a escolha preferencial. A sua pontuação F1 garante que os furos roscados identificados sejam altamente precisos, minimizando a probabilidade de existência de falsos positivos. Além disso, a sua abordagem alinha-se diretamente com as características geométricas do furo roscado, tornando-o adequado para essa tarefa.

Embora o algoritmo SVM também tenha um desempenho admirável, a sua pontuação F1 é inferior, uma vez que indica uma probabilidade marginalmente maior da existência de falsos positivos. No entanto, a capacidade do algoritmo de uma boa divulgação das novas instâncias e o seu potencial de lidar com as variações nas aparências do furo roscado, fazem dele uma boa alternativa.

5.2 Trabalhos futuros

Relativamente a trabalhos futuros, o estado final do último protótipo e as considerações tomadas anteriormente, são apontados alguns trabalhos futuros no que toca à melhoria do trabalho desenvolvido neste documento.

- Integração do *deep learning*. As redes neuronais convolucionais (CNN) demonstram um desempenho notável nas tarefas de reconhecimento de imagem e podem melhorar a precisão e a robustez da deteção de furos roscados.
- Combinar os pontos fortes dos diferentes algoritmos utilizados para um melhor desempenho. A exploração de técnicas de fusão de recursos baseados nos dois algoritmos pode aumentar a precisão e a confiabilidade da deteção de furos roscados com resultados mais robustos.
- Passagem para uma inspeção pormenorizada na classificação de defeitos ao nível da presença de limalhas e rosca dupla.

Referências

[1] Simon Korman, Daniel Reichman, Gilad Tsur, and Shai Avidan. Fast-match: Fast affine template matching. In *CVPR*, pages 2331–2338. IEEE Computer Society, 2013.

[2] Luis Alcántara, Carlos Toledo, and Irma Carrillo. Automated fault detection and diagnostics for aluminum threads using statistical computer vision. page 29–38, Oct 2018.

[3] Ogê Marques Filho and Hugo Vieira Neto. *Processamento Digital de Imagens*. Editora Brasport, Rio de Janeiro, Brazil, 1999.

[4] L. Tan and J. Jiang. *Digital Signal Processing: Fundamentals and Applications*. Elsevier Science, 2018.

[5] P. R. S. Swaroop and Neelam Sharma. An overview of various template matching methodologies in image processing. *International Journal of Computer Applications*, 153:8–14, 2016.

[6] Raffaele Pugliese, Stefano Regondi, and Riccardo Marini. Machine learning-based approach: global trends, research directions, and regulatory standpoints. *Data Science and Management*, 4:19–29, 2021.

[7] Solution guide ii-d classification halcon 22.11 progress, 2008.

[8] 3.1 The Agent-Environment Interface — incompleteideas.net. `http://www.incompleteideas.net/book/ebook/node28.html`. [Accessed 09-May-2023].

[9] Jiun-Hung Chen, Chu-Song Chen, and Yong-Sheng Chen. Fast algorithm for robust template matching with m-estimators. *IEEE Transactions on Signal Processing*, 51(1):230–243, 2003.

[10] Rytis Augustauskas, Arūnas Lipnickas, and Tadas Surgailis. Segmentation of drilled holes in texture wooden furniture panels using deep neural network. *Sensors*, 21(11):3633, may 2021.

[11] Hamou Chehri, Abdellah Chehri, Laszlo Kiss, and Alfred Zimmerman. Automatic anode rod inspection in aluminum smelters using deep-learning techniques: A case study. *Procedia Computer Science*, 176:3536–3544, 2020.

[12] Ali Maghami, Meshkat Salehi, and Matt Khoshdarregi. Automated vision-based inspection of drilled CFRP composites using multi-light imaging and deep learning. *CIRP Journal of Manufacturing Science and Technology*, 35:441–453, nov 2021.

[13] Alejandro Hernandez, Ali Maghami, and Matt Khoshdarregi. A machine vision framework for autonomous inspection of drilled holes in cfrp panels. In *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, pages 669–675, 2020.

[14] Vedang Chauhan, Heshan Fernando, and Brian Surgenor. Effect of illumination techniques on machine vision inspection for automated assembly machines. 06 2014.

[15] Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[16] Junyan Hu, Hanlin Niu, Joaquin Carrasco, Barry Lennox, and Farshad Arvin. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(12):14413–14423, 2020.

[17] Premanand S. The A-Z guide to Support Vector Machine — analyticsvidhya.com. `https://www.analyticsvidhya.com/blog/2021/06/support-vector-machine-better-understanding/`. [Accessed 09-May-2023].

[18] Classification based on decision tree algorithm for machine learning. 2.

[19] Danfeng Xie, Lei Zhang, and Li Bai. Deep learning in visual computing and signal processing. *Applied Computational Intelligence and Soft Computing*, 2017:1–13, 2017.

[20] Mario Campos, Teresa Martins, Manuel Ferreira, and Cristina Santos. Detection of defects in automotive metal components through computer vision. In *2008 IEEE International Symposium on Industrial Electronics*. IEEE, jun 2008.

[21] O. Laligant, F. Truchetet, and E. Fauvet. Wavelets transform in artificial vision inspection of threading. volume 1, page 513 – 518, 1993. Cited by: 6.

[22] Vedang Chauhan and Brian Surgenor. A comparative study of machine vision based methods for fault detection in an automated assembly machine. *Procedia Manufacturing*, 1:416–428, 2015.

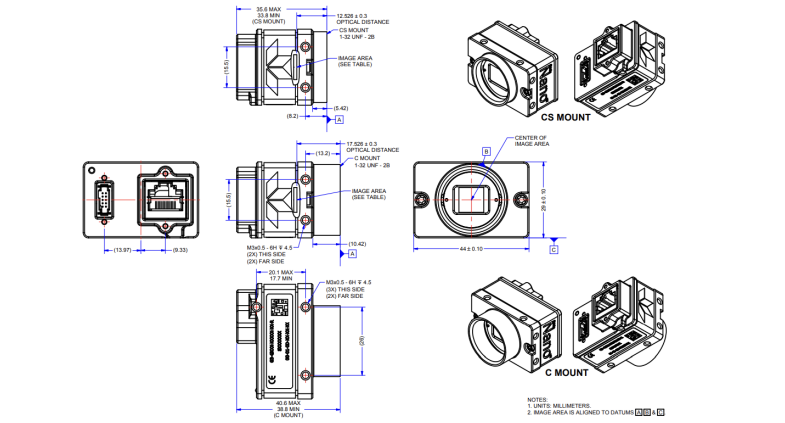
[23] João António Castro Esteves. Inspectionadaptmark sistema inteligente de inspeção e correção de parâmetros de marcações a laser, 2022.

[24] Ana Rosa Marques Gonçalves. Análise de perfis de utilização em sistemas de autoaprendizagem, 2016.

Apêndice A

Cálculo para a escolha da lente

Neste apêndice será descrito as etapas feitas para a escolha da lente. Na figura A.1 estão descritas as características da câmara para proceder ao cálculo da distância focal.



Genie Nano M2020 Mono	2064 x 1544	Mono	GigE Vision	3.45 µm	Standard: 38 fps
Part number: G3-GM11-M2020	*Sony IMX265*				TurboDrive: 53 fps

Total Pixels	Interface Speed	Bit Depth	Operating Temperature
~ 3.2 MP	1 Gbps	8 bit, 12 bit	-20—65 °C
Dynamic Range	**Supported Lens Options**	**Shutter Type**	**Power Requirement**
76 dB	C-Mount, CS-Mount	Global Shutter	PoE or 10-36 VDC
Synchronisation	**Exposure Control**	**General Purpose I/O**	**Dimensions (W x H x L)**
software trigger, free-run, hardware trigger, PTP	hardware trigger, programmable via camera API, automatic	2 digital input, 2 digital output	21 x 29 x 44 mm

Figura A.1: Características específicas da câmara Dalsa Genie Nano M2020

O cálculo de s foi feita da com base no valor do tamanho do píxel (*pixel size*) e a sua resolução (2064x1544), então:

$$s = pixelsize \times resolution = 3,45 \times 2064 = 7,12mm$$

De seguida, fez-se os cálculos para o campo de visão (FOV)

$$FOV = 107 + 10 + 2 = 119mm$$

Finalmente, procedeu-se ao cálculo da distância focal para o furo 240 (WD=220mm) e para o furo 312 (WD=180mm):

$$f_{240} = 7,12 \times \frac{240}{119} = 14,36mm$$

$$f_{312} = 7,12 \times \frac{180}{119} = 10,75mm$$

Foram obtidos resultados de 14.36 mm para o furo 240 e 10,75mm para o furo 312. Concluindo-se que uma lente com uma distância focal de 16 é o suficiente.

Apêndice B

Instalação do Halcon

No apêndice presente encontra-se descrito e ilustrado todo o procedimento de instalação do software Halcon, com a licença da MVTec. O primeiro passo a fazer é criar uma nova conta. Efetuado o registo e a sua confirmação de email, é apresentada a página de instalação onde existem diversos pacotes de instalação da MVTec. Deve-se descarregar o pacote Halcon Steady com a versão mais recente no sistema operativo em que se vai instalar. O pacote Halcon Progress também é opção, porém a licensa é limitada.

Após a instalação, abre-se o aplicativo executável som.exe (Software Manager), onde irá abrir um terminal que direciona um localhost da Porta 8188 numa janela de um browser onde poderá fazer a instalação do Halcon (Figuras B.1, B.2 e B.3).

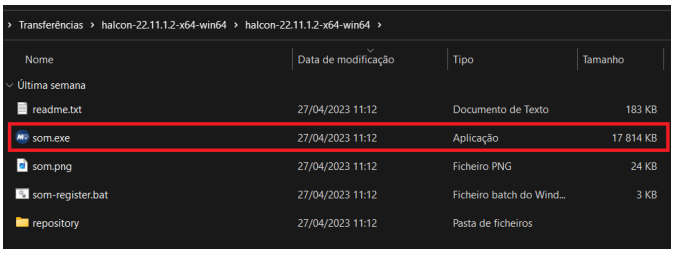


Figura B.1: Aplicativo executável som.exe

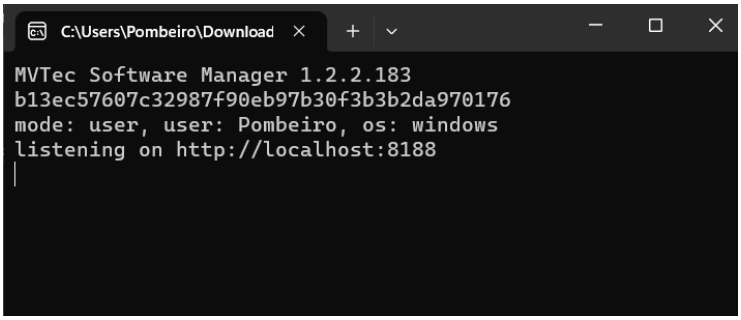


Figura B.2: Terminal

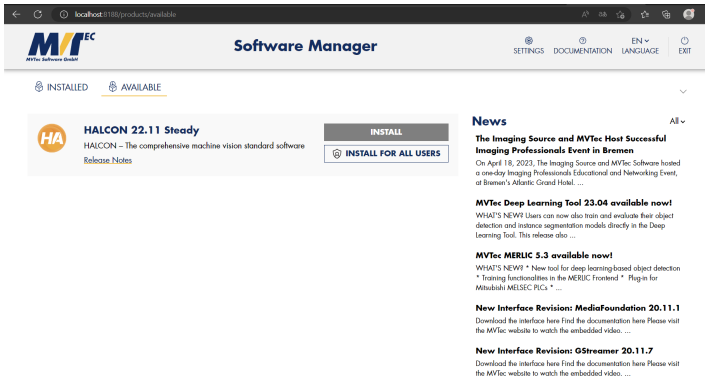


Figura B.3: Estrutura do aplicativo som

Com o Halcon instalado, o último passo será dar o upload nas licensas disponíveis que temos para o software (Figura B.4). As licensas apresentadas na figura B.5 são o *SDK*, *Runtime* e *Deep Learning*. Assim, o software está pronto para ser utilizado.

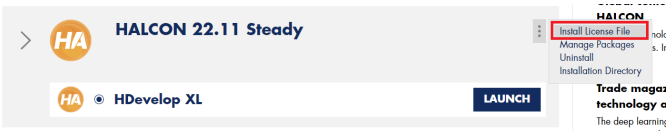


Figura B.4: Definições do software

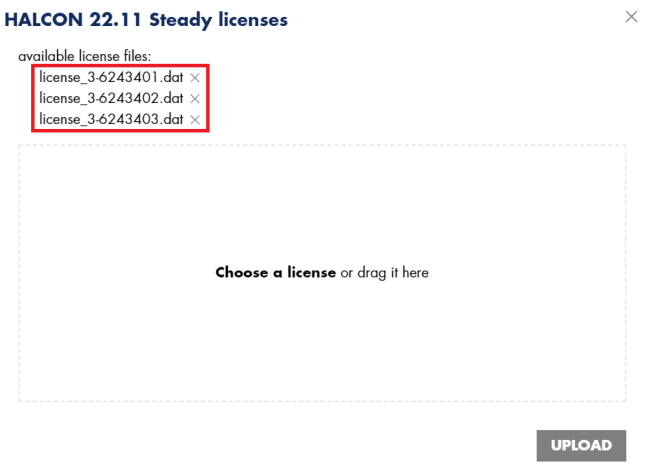


Figura B.5: Licensas instaladas no software Halcon

Apêndice C

Utilização do software Hercules

Neste apêndice será descrita a comunicação entre *Hercules* e o software Halcon, e entre hércules e autómato S7-1200 Siemens. Este aplicativo apenas foi feito para simular a comunicação entre Halcon e autómato. Inicialmente, o objetivo visava estabelecer a comunicação do programa com o autómato, estando o programa Halcon à escuta para receber pedidos do autómato, pedidos esses vindos inicialmente do robot quando este chega à posição. Recebido esse pedido, o programa iniciava a captura de imagem e prosseguia para o tratamento desta. No final da avaliação, o programa iria enviar uma mensagem para autómato, sendo encaminhada para o robot para que este prosseguisse para próxima posição, estando o programa à espera de receber uma próxima mensagem da posição em que o robot se encontra.

Para tal, foi necessária a utilização do software *Hercules* SETUP, sendo este normalmente usado para configurar e testar diferentes tipos de dispositivos que se comunicam em redes *TCP/IP*. É muito utilizado para testar e encontrar soluções de *troubleshooting* em equipamentos de rede. Perante estas características, foi utilizada esta ferramenta para os dois passos, isto é, usou-se para testar a comunicação entre *Hercules* e o programa Halcon e para comunicação entre *Hercules* e autómato. No primeiro caso, foi necessário desenvolver uma comunicação *TCP/IP* no software Halcon na utilização de sockets. A aplicação *Hercules* será o servidor, estando à escuta para receber pedidos do cliente, que será o programa Halcon. A Listagem C.1 demonstra o código desenvolvido com as funções utilizadas no software Halcon para conectar ao *Hercules* e a sua troca de mensagens. A figura C.3 representa o *software Hercules* e as suas mensagens recebidas e enviadas para o programa Halcon.

```
*Protocolo de comunicacao cliente
Protocol := 'TCP4'
Timeout := 5.0

open_socket_connect ('localhost', 2000, 'protocol', 'TCP4', Socket)
get_socket_param (Socket, 'address_info', Address)
send_data (Socket, 'z', 'Posicao 1', [])
        receive_data (Socket, ['c', 'z'], Message, From)

        if(Message == 'posicao 1')
            grab_image_async (Image, AcqHandle, -1)
            grab_image_async (Image, AcqHandle, -1)
```

```
13          dev_display(Image)
14          gen_circle (ROI_0, 1029.36, 972.244, 71.1381)
15          reduce_domain (Image, ROI_0, ImageReduced)
16          threshold (ImageReduced, Regions, 163, 255)
17          dilation_circle (Regions, RegionDilation, 3)
18          connection (RegionDilation, Connection)
19          select_shape (Connection, SelectedRegions, ['area','height'], 'and',
    [6016.64,102.96], [7661.74,119.96])
20          count_obj (SelectedRegions, Number)
21          send_data (Socket, 'z', 'PROCESSING DONE F1', [])
22       endif
23
24       if(Message == 'posicao 2')
25          grab_image_async (Image1, AcqHandle, -1)
26          dev_display(Image1)
27          grab_image_async (Image1, AcqHandle1, -1)
28          gen_circle (ROI_1, 978.09, 932.3, 87.6181)
29          reduce_domain (Image1, ROI_1, ImageReduced1)
30          threshold (ImageReduced1, Regions1, 232, 255)
31          dilation_circle (Regions1, RegionDilation1, 3)
32          connection (RegionDilation1, Connection1)
33          select_shape (Connection1, SelectedRegions1, ['area','height'], 'and',
    [5000,80], [10000,100])
34          count_obj (SelectedRegions1, Number1)
35          send_data (Socket, 'z', 'PROCESSING DONE F2', [])
36       endif
```

Listagem C.1: Conexão com o software Hercules e a troca de mensagens no Halcon

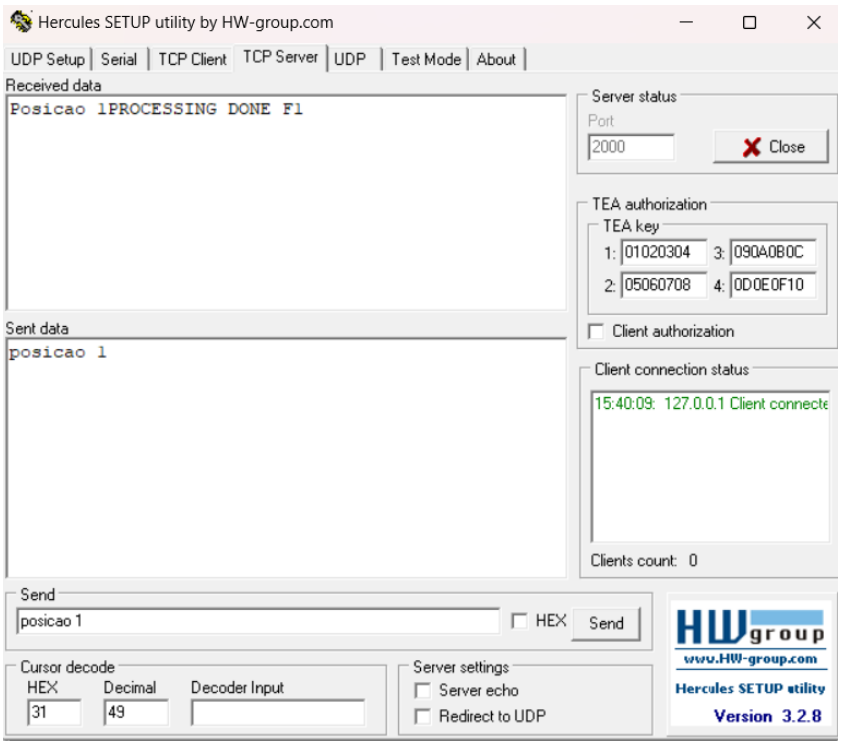


Figura C.1: Sofware Hercules como Servidor

De uma forma muito breve, na listagem C.1, a linha 5 permite ao Halcon conectar ao servidor Hercules na porta 2000 pelo protocolo 'TCP4'. De seguida, na linha 7, o Halcon irá mandar uma mensagem 'Posicao1' para o Hercules e ficará à espera de receber um pedido na linha 31. Ao receber o pedido por parte do Hércules, o programa aguarda a receção do pedido, com a mensagem específica '*posicao 1*', para que este ajuste a condição IF na linha 10. Caso a mensagem recebida seja a desejada, este irá percorrer o código para o tratamento de imagem e verifica se o furo contém rosca. Feito o tratamento, o Halcon envia uma mensagem 'PROCESSING DONE F1' para o hércules na linha 21, de que a foto do primeiro furo foi feita com sucesso, prosseguindo para a próxima etapa que será o furo 2 que contém outra mensagem específica 'posicao 2' que, por sua vez, fará outro tratamento de mensagem, até chegar à posição origem do robot UR10 '*posição origem*'.

No segundo caso, foi necessário desenvolver um programa no TIA Portal para receber e enviar pedidos de mensagens vindas do programa Halcon. Para isso foi necessário, em primeiro lugar, fazer a conecção por TCP/IP utilizando o bloco "TCON" (Figura C.2). Foram necessárias fazer alterações nas propriedades da função, colocando o endereço de IP e a porta ao qual queremos comunicar (Figura C.3).

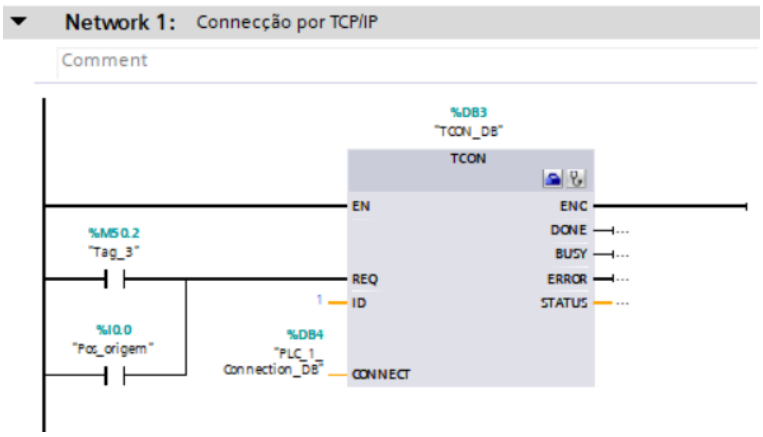


Figura C.2: Função TCON ativa quando a entrada digital "Pos_origem" é ativa

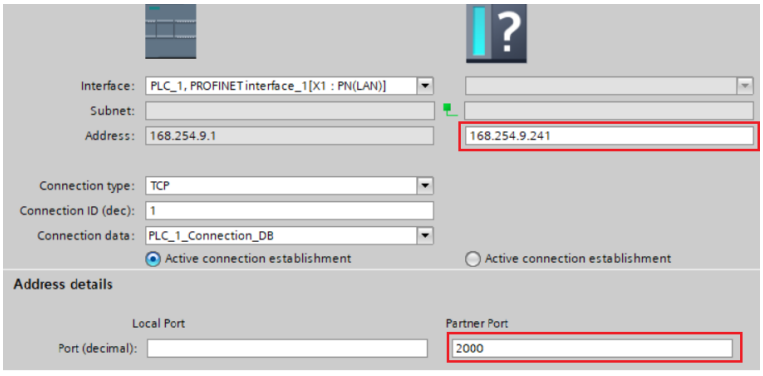
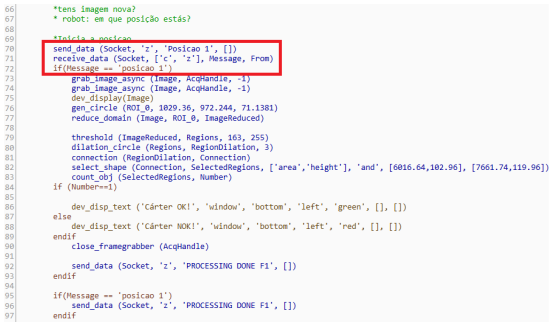
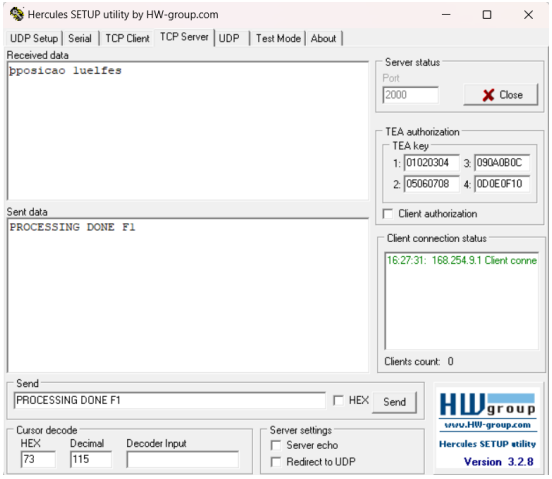


Figura C.3: Propriedades da função bloco TCON

De seguida, procedeu-se ao desenvolvimento do envio de mensagens do autómato para o Halcon utilizando a função bloco "TSEND", que fará o envio de mensagens específicas da posição em que o robot se encontra. O aplicativo *Hercules* serviu para substituir o software Halcon para analisar qual o tipo de mensagens específicas que eram recebidas. Na figura apresentada, do lado esquerdo demonstra a mensagem específica "posicao 1"que se pretende que seja recebida pelo *Hercules*. De facto, a imagem do lado direito apresenta os resultados da mensagem recebida do autómato e denotou-se que a mensagem foi corrompida do tipo "pposicao 1uelfes".



(a) Código Halcon da comunicação entre programa e autómato



(b) Mensagem corrompida recebida no *Hercules*

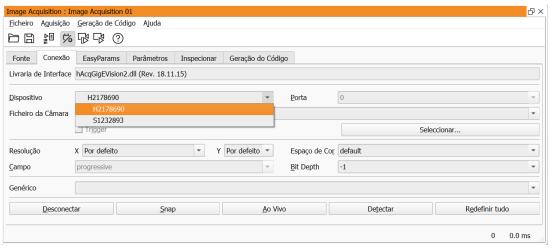
Figura C.4: Mensagens enviadas e recebidas por parte do Halcon e a sua substituição por parte do Hercules para análise das mensagens específicas recebidas pelo PLC

Podemos concluir que, existe uma comunicação cujas mensagens recebidas do programa Halcon pelo autómato apresentavam-se corrompidas o que impedia a passagem para a fase de classificação da imagem sem que este recebesse uma mensagem específica vinda do autómato. Como descrito anteriormente, o Halcon tem a possibilidade de exportar o programa desenvolvido noutras linguagens de programação. Assim, existe a possibilidade de utilização das bibliotecas necessárias para uma comunicação eficiente no uso do *Visual Studio*.

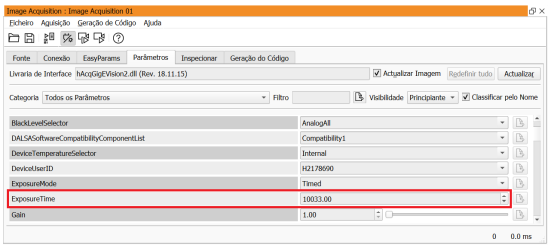
Apêndice D

Concepção dos modelos destinados ao uso no algoritmo *Template Matching Shape-Based*

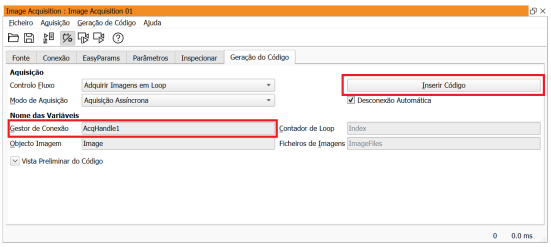
A criação dos modelos para o reconhecimento das roscas por via template macthing foi essencial para que se pudesse avaliar a conformidade do furo. Para a criação dos modelos utilizaram-se imagens guardadas numa pasta, peças conformes e não conformes. Inicialmente, foi utilizada a ferramenta "Image acquisition" para inserir código da conexão das duas câmaras. Foi testada a conexão das duas câmaras (Figura D.1a), parametrizou-se a sua exposição (Figura D.4b) e posteriormente inseriu-se o código ao programa (Figura D.5), fazendo as alterações de posição de código necessárias.



(a) Janela "Conexão"nas propriedades do Image Acquisition



(b) Janela "Parâmetros"nas propriedades do Image Acquisition



(c) Janela "Geração de Código"nas propriedades do Image Acquisition

```
* Image Acquisition 01: Code generated by Image Acquisition 01
open_framegrabber ('GigEVision2', 0, 0, 0, 0, 0, 0, 'progressive', -1, 'default', -1, 'false', 'default', 'H2178690',
* Image Acquisition 01: Attention: The parameterization may fail in case parameters need to
* Image Acquisition 01: be set in a specific order (e.g., image resolution vs. offset).
set_framegrabber_param (AcqHandle1, 'ExposureTime', 9003.0)

* Image Acquisition 02: Code generated by Image Acquisition 02
open_framegrabber ('GigEVision2', 0, 0, 0, 0, 0, 0, 'progressive', -1, 'default', -1, 'false', 'default', 'S1232893',
* Image Acquisition 02: Attention: The parameterization may fail in case parameters need to
* Image Acquisition 02: be set in a specific order (e.g., image resolution vs. offset).
set_framegrabber_param (AcqHandle2, 'ExposureTime', 8546.0)

grab_image_start (AcqHandle1, -1)
while (true)
    grab_image_async (Image, AcqHandle1, -1)
    * Image Acquisition 01: Do something
endwhile
close_framegrabber (AcqHandle1)

grab_image_start (AcqHandle2, -1)
while (true)
    grab_image_async (Image, AcqHandle2, -1)
    * Image Acquisition 02: Do something
endwhile
close_framegrabber (AcqHandle2)
```

(d) Código inserido para a conexão das duas câmaras

Figura D.1: Propriedades da ferramenta Image Acquisition

Seguidamente, na janela "Assistentes" foi utilizada a ferramenta "Abrir um novo matching" e selecionada a imagem da peça conforme, selecionou-se a região de interesse (*ROI*), neste caso a área roscada (Figuras D.2 e D.3). Este processo também foi feito para o modelo do furo sem rosca com objetivo de adquirir maior credibilidade do programa da não presença de rosca.

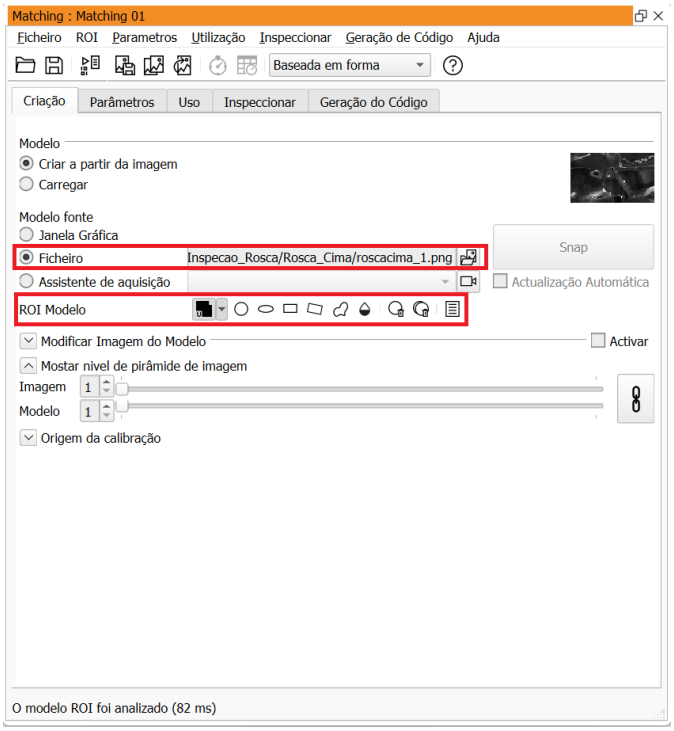


Figura D.2: Janela "Criação"nas propriedades do Template Matching

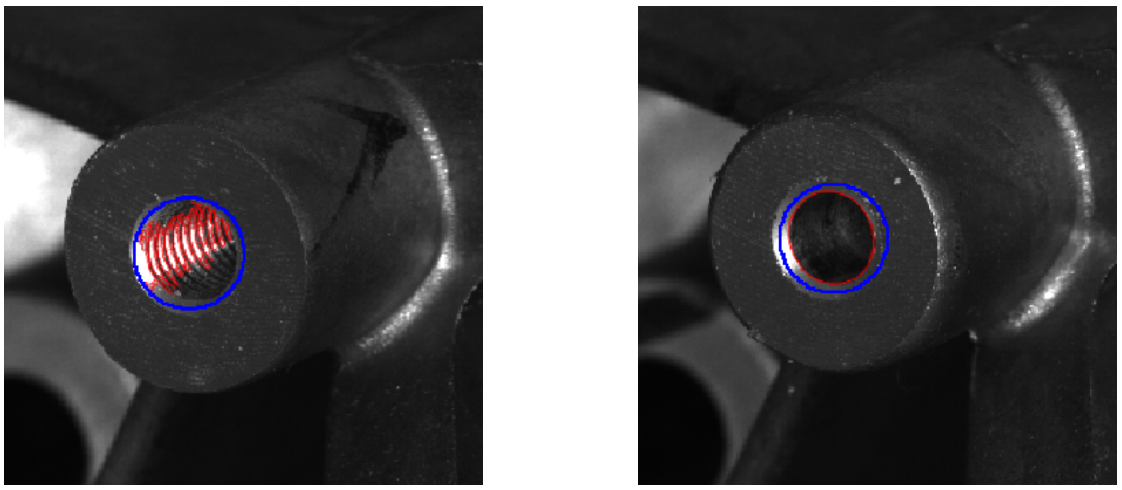
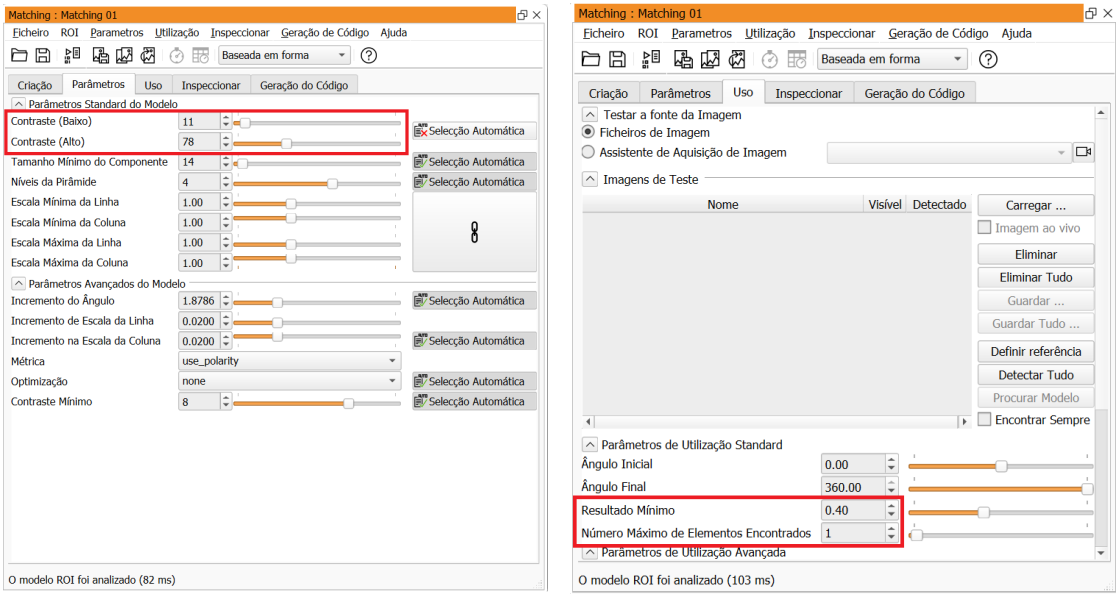


Figura D.3: ROI no furo do cárter conforme (esquerda) e não conforme (direita)

Na janela "Parâmetros"(Figura D.4a) foram feitas algumas alterações no contraste alto e baixo para obter o formato ideal da rosca. Na janela "uso" (Figura D.4) em opção "Parâmetros de Utilização Standard" foi introduzido um resultado mínimo de 0,4, isto é, a percentagem necessária para que o template seja idêntico ao template da imagem referente. Foi então, introduzido o valor de 1 no "Número Máximo de Elementos Encontrados", pois só é necessário identificar um furo roscado em cada imagem.



(a) Janela "Parâmetros"nas propriedades do *Template Matching*

(b) Janela "Uso"nas propriedades do *Template Matching*

Figura D.4: Propriedades da ferramenta *Template Matching*

Finalmente na janela "Geração de código" (Figura D.5), ao selecionar a opção "Nomes das Variáveis das correspondências" foram feitas mudanças em todos os nomes das variáveis com um final no nome "_Cima" e "_Lado" para o furo 240 e 312 respetivamente para não existirem conflitos no treino do modelo e foi inserido o código no programa.

Com o código necessário, foram feitas algumas alterações nas posições em que o código deveria estar inserido, colocando a primeira parte do código do *template matching*, que se refere à criação do modelo dos dois furos fora do ciclo WHILE, uma vez que só será necessário correr esse código uma única vez. Dentro do ciclo WHILE, inseriu-se a função TRY e CATCH para cada processo do tratamento de imagem de cada furo desde a obtenção da imagem "grab_image_async()" até desconectar a câmara selecionada "close_framegrabber", embora essa função não seja necessária quando exportado o programa para o Visual Studio.

O código final apresentado (Listagem D.1) demonstra apenas o processo feito para o furo 240 do programa no Halcon, sendo este idêntico para o furo 312.

```
1 * Matching 01: BEGIN of generated code for model initialization
2 * Matching 01: Obtain the model image
```

```
read_image (Image_Cima, 'C:/Users/Pombeiro/Desktop/halcon/Inspecao_sensor/Camara1/
    roscacima_1.png')

* Matching 01: Build the ROI from basic regions
gen_ellipse (ModelRegion_Cima, 617.922, 1260.97, rad(1.33367), 53.2535, 59.7455)
* Matching 01: Reduce the model template
reduce_domain (Image_Cima, ModelRegion_Cima, TemplateImage_Cima)
* Matching 01: Create and train the shape model
create_generic_shape_model (ModelID_Cima)
* Matching 01: set the model parameters
set_generic_shape_model_param (ModelID_Cima, 'contrast_high', 157)
set_generic_shape_model_param (ModelID_Cima, 'contrast_low', 13)
set_generic_shape_model_param (ModelID_Cima, 'metric', 'use_polarity')
train_generic_shape_model (TemplateImage_Cima, ModelID_Cima)
* Matching 01: Get the model contour for transforming it later into the image
get_shape_model_contours (ModelContours_Cima, ModelID_Cima, 1)
* Matching 01: Support for displaying the model
* Matching 01: Get the reference position
area_center (ModelRegion_Cima, ModelRegion_CimaArea, RefRow_Cima, RefColumn_Cima)
vector_angle_to_rigid (0, 0, 0, RefRow_Cima, RefColumn_Cima, 0, HomMat2D_Cima)
affine_trans_contour_xld (ModelContours_Cima, TransContours_Cima, HomMat2D_Cima)
* Matching 01: Display the model contours
dev_display (Image_Cima)
dev_set_color ('green')
dev_set_draw ('margin')
dev_display (ModelRegion_Cima)
dev_display (TransContours_Cima)

* Matching 04: BEGIN of generated code for model initialization
* Matching 04: Obtain the model image
read_image (Image_Furo_Cima, 'C:/Users/Pombeiro/Desktop/halcon/Inspecao_sensor/Camara1/
    roscacima_D_1.png')
* Matching 04: Build the ROI from basic regions
gen_ellipse (ModelRegion_Furo_Cima, 661.885, 1234.56, rad(16.2989), 54.8744, 63.9327)
* Matching 04: Reduce the model template
reduce_domain (Image_Furo_Cima, ModelRegion_Furo_Cima, TemplateImage_Furo_Cima)
* Matching 04: Create and train the shape model
create_generic_shape_model (ModelID_Furo_Cima)
* Matching 04: set the model parameters
set_generic_shape_model_param (ModelID_Furo_Cima, 'contrast_high', 62)
set_generic_shape_model_param (ModelID_Furo_Cima, 'contrast_low', 29)
set_generic_shape_model_param (ModelID_Furo_Cima, 'metric', 'use_polarity')
train_generic_shape_model (TemplateImage_Furo_Cima, ModelID_Furo_Cima)
* Matching 04: Get the model contour for transforming it later into the image
get_shape_model_contours (ModelContours_Furo_Cima, ModelID_Furo_Cima, 1)
* Matching 04: Support for displaying the model
* Matching 04: Get the reference position
area_center (ModelRegion_Furo_Cima, ModelRegion_Furo_CimaArea, RefRow_Furo_Cima,
    RefColumn_Furo_Cima)
vector_angle_to_rigid (0, 0, 0, RefRow_Furo_Cima, RefColumn_Furo_Cima, 0,
    HomMat2D_Furo_Cima)
affine_trans_contour_xld (ModelContours_Furo_Cima, TransContours_Furo_Cima,
    HomMat2D_Furo_Cima)
* Matching 04: Display the model contours
dev_display (Image_Furo_Cima)
dev_set_color ('green')
dev_set_draw ('margin')
dev_display (ModelRegion_Furo_Cima)
```

```
dev_display (TransContours_Furo_Cima)

open_framegrabber ('GigEVision2', 0, 0, 0, 0, 0, 0, 'progressive', -1, 'default', -1, '
    false', 'default', 'S1232893', 0, -1, AcqHandle2)
dev_set_window (WindowHandle1)
grab_image_start (AcqHandle1, -1)

try
    grab_image_async (Image_Cima_Foto, AcqHandle1, -1)
    dev_display (Image_Cima_Foto)
    * Matching 01: BEGIN of generated code for model application
    * Matching 01: Set the search paramaters
    set_generic_shape_model_param (ModelID_Cima, 'border_shape_models', 'false')
    * Matching 01: The following operations are usually moved into
    * Matching 01: that loop where the acquired images are processed
    * Matching 01: Find the model
    find_generic_shape_model (Image_Cima_Foto, ModelID_Cima, MatchResultID_Cima,
    NumMatchResult_Cima)
    * Matching 01: Retrieve results
    for I_Cima := 0 to NumMatchResult_Cima-1 by 1
        * Matching 01: Display the detected match
        dev_display (Image_Cima)
        get_generic_shape_model_result_object (MatchContour_Cima, MatchResultID_Cima,
    I_Cima, 'contours')
        dev_set_color ('green')
        dev_display (MatchContour_Cima)
        * Matching 01: Retrieve parameters of the detected match
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'row', Row_Cima)
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'column',
    Column_Cima)
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'angle', Angle_Cima)
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'scale_row',
    ScaleRow_Cima)
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'scale_column',
    ScaleColumn_Cima)
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'hom_mat_2d',
    HomMat2D_Cima)
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'score', Score_Cima)
        stop ()
    endfor
    * Matching 01: Code for alignment of e.g., measurements
    for I_Cima := 0 to NumMatchResult_Cima-1 by 1
        * Matching 01: Retrieve a hom_mat2d for each of the matching results
        * Matching 01: Retrieve the matching results
        get_generic_shape_model_result (MatchResultID_Cima, I_Cima, 'hom_mat_2d',
    HomMat2D_Cima)
        hom_mat2d_identity (AlignmentHomMat2D_Cima)
        hom_mat2d_translate (AlignmentHomMat2D_Cima, -RefRow_Cima, -RefColumn_Cima,
    AlignmentHomMat2D_Cima)
        hom_mat2d_compose (HomMat2D_Cima, AlignmentHomMat2D_Cima, AlignmentHomMat2D_Cima
    )
        * Matching 01: Insert your code using the alignment here, e.g., code generated
    by
        * Matching 01: the measure assistant with the code generation option
        * Matching 01: 'Alignment Method' set to 'Affine Transformation'.
    endfor

    * Matching 01: END of generated code for model application
```

```
103
104     * Matching 02: BEGIN of generated code for model application
105     * Matching 02: Set the search paramaters
106     set_generic_shape_model_param (ModelID_Furo_Cima, 'min_score', 0.4)
107     set_generic_shape_model_param (ModelID_Furo_Cima, 'num_matches', 1)
108     set_generic_shape_model_param (ModelID_Furo_Cima, 'border_shape_models', 'false')
109     * Matching 02: The following operations are usually moved into
110     * Matching 02: that loop where the acquired images are processed
111     *
112     * Matching 02: Find the model
113     find_generic_shape_model (Image_Cima_Foto, ModelID_Furo_Cima,
        MatchResultID_Furo_Cima, NumMatchResult_Furo_Cima)
114     *
115     * Matching 02: Retrieve results
116     for I_Furo_Cima := 0 to NumMatchResult_Furo_Cima-1 by 1
117         * Matching 02: Display the detected match
118         dev_display (Image_Furo_Cima)
119         get_generic_shape_model_result_object (MatchContour_Furo_Cima,
        MatchResultID_Furo_Cima, I_Furo_Cima, 'contours')
120         dev_set_color ('green')
121         dev_display (MatchContour_Furo_Cima)
122         * Matching 02: Retrieve parameters of the detected match
123         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, 'row',
        Row_Furo_Cima)
124         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, 'column',
        Column_Furo_Cima)
125         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, 'angle',
        Angle_Furo_Cima)
126         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, 'scale_row
        ', ScaleRow_Furo_Cima)
127         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, '
        scale_column', ScaleColumn_Furo_Cima)
128         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, '
        hom_mat_2d', HomMat2D_Furo_Cima)
129         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, 'score',
        Score_Furo_Cima)
130     endfor
131
132     * Matching 02: Code for alignment of e.g., measurements
133     for I_Furo_Cima := 0 to NumMatchResult_Furo_Cima-1 by 1
134         * Matching 04: Retrieve a hom_mat2d for each of the matching results
135         * Matching 04: Retrieve the matching results
136         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, '
        hom_mat_2d', HomMat2D_Furo_Cima)
137         hom_mat2d_identity (AlignmentHomMat2D_Furo_Cima)
138         hom_mat2d_translate (AlignmentHomMat2D_Furo_Cima, -RefRow_Furo_Cima, -
        RefColumn_Furo_Cima, AlignmentHomMat2D_Furo_Cima)
139         hom_mat2d_compose (HomMat2D_Furo_Cima, AlignmentHomMat2D_Furo_Cima,
        AlignmentHomMat2D_Furo_Cima)
140         * Matching 04: Insert your code using the alignment here, e.g., code generated
        by
141         * Matching 04: the measure assistant with the code generation option
142         * Matching 04: 'Alignment Method' set to 'Affine Transformation'.
143     endfor
144
145     * Matching 02: Code for rectification of the image
146     * Matching 02: Calculate the hom_mat2d for the model
147     hom_mat2d_identity (HomMat2DModel)
```

```
148     hom_mat2d_translate (HomMat2DModel, RefRow_Furo_Cima, RefColumn_Furo_Cima,
        HomMat2DModel)
149     for I_Furo_Cima := 0 to NumMatchResult_Furo_Cima-1 by 1
150         * Matching 02: Calculate an inverse hom_mat2d for each of the matching results
151         get_generic_shape_model_result (MatchResultID_Furo_Cima, I_Furo_Cima, '
        hom_mat_2d', HomMat2D_Furo_Cima)
152         hom_mat2d_invert (HomMat2D_Furo_Cima, HomMat2DMatchInvert)
153         hom_mat2d_compose (HomMat2DModel, HomMat2DMatchInvert, RectificationHomMat2D)
154         affine_trans_image (Image_Furo_Cima, RectifiedImage_Furo_Cima,
        RectificationHomMat2D, 'constant', 'false')
155         *
156         * Matching 02: Insert your code using the rectified image here
157     endfor
158
159     * Matching 02: END of generated code for model application
160 catch (Exception)
161 endtry
162     close_framegrabber (AcqHandle1)
```

Listagem D.1: Algoritmo *Template Matching Shape-Based* para o furo 240

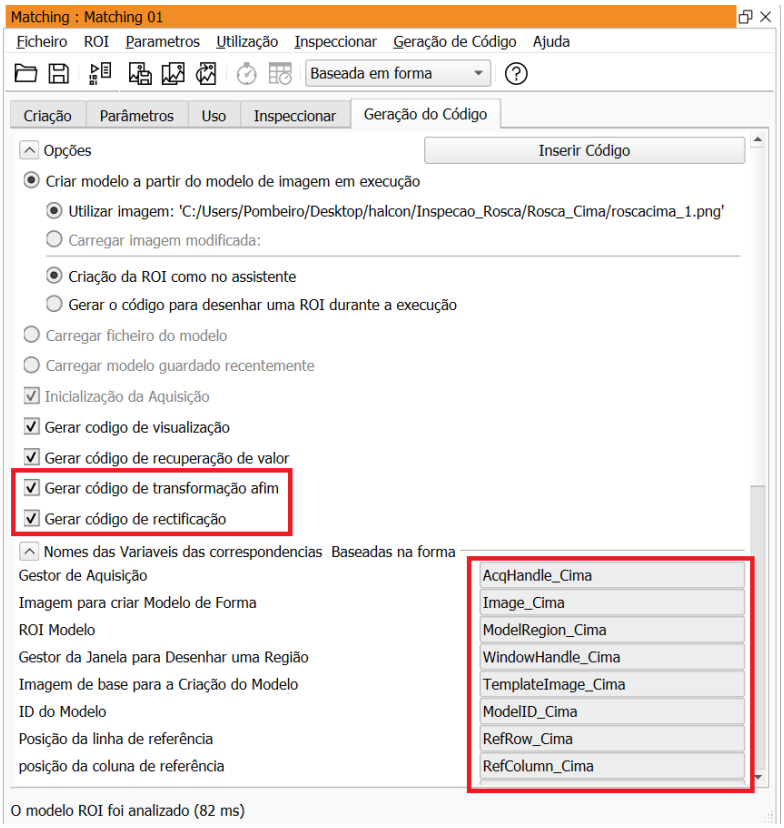


Figura D.5: Janela "Geração do Código" nas propriedades do *Template Matching*

Intentionally blank page.

Apêndice E

Parâmetros específicos para o algoritmo SVM no Halcon

Nesta secção será descrito o ajuste de parâmetros para uma classificação por SVM. Os parâmetros mais importantes que foram ajustados para que o classificador SVM funcionasse de maneira ideal foram os parâmetros de entrada *Nu* e *KernelParam* que se encontram no operador *create_ class_ svm*. E para que a classificação funcionasse, foi também necessário ajustar a velocidade e o parâmetro ajuste utilizado foi o *Preprocessing* no operador *create_ class_ svm*.

Serão apresentados os parâmetros para cada operador utilizado no projeto e a influência destes na classificação. São parâmetros necessários para a criação e treino do classificador.

E.1 Ajuste no operador *create_ class_ svm*

O classificador SVM é criado por este operador. Existem diversas propriedades definidas e importantes para as etapas de classificação que se apresentam como:

- **Parâmetro *NumFeatures*** – Parâmetro de entrada que especifica a dimensão do vetor característica usado para treino. No projeto apresenta o parâmetro de valor sete, que são eles a área, compacidade da região, a convexidade da região e os quatro momentos. Todos eles pertencentes ao cálculo das características do operador *calculate_ features*.
- **Parâmetro *KernelType*** – Define como o espaço da feature é mapeado para a dimensão superior. O mapeamento adequado e recomendado na maioria dos casos usa um kernel que é baseado na curva de distribuição do erro de Gauss e é chamada de kernel da função de base radial gaussiana ('rbf') demonstrada na equação X, citada no capítulo 2. Ao definir como 'rbf', este é usado para ajustar o γ da curva de erro (Figura E.1) e deve ser ajustado com muito cuidado. Se o valor de γ for muito alto, o número de vetores de suporte aumenta, o que resulta por um lado num *overfitting*, ou seja, por um lado a capacidade de generelização do classificador é perdida, e, por outro lado a velocidade é reduzida. Por outro lado, ainda, com um valor muito baixo para γ, ocorre um underfitting, ou seja, o número de vetores de suporte não é suficiente para obter um resultado de classificação satisfatório.

Inicialmente, foi introduzido um valor de γ baixo (0,02), sendo este aumentado progressivamente até ao valor chegado do projeto (0,10). Inicialmente foi utilizado este o *kernel* 'rbf', mas depois foi utilizado *kernel* do tipo 'linear' que transforma o espaço de características usando um produto escalar. Os resultados foram melhores que à do 'rbf' uma vez que as classes demonstram ser linearmente separáveis. Uma vez selecionada este tipo de *kernel*, o valor de γ deixa de ter sentido e é ignorado.

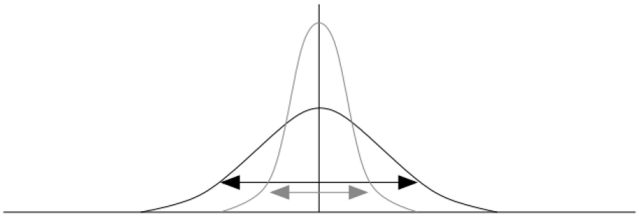


Figura E.1: A quantidade de influência do vetor suporte sobre ao seu redor descrito pelo valor γ

- **Parâmetro *Nu*** – Nestas classes que não são linearmente separáveis, os dados das diferentes classes podem sobrepor-se. Como tal, este parâmetro de entrada regulariza a separação das classes, isto é, com o Nu, o limite superior para erros de treino dentro das áreas sobrepostas entre as classes é ajustado e, ao mesmo tempo, o limite inferior para o número de vetores suporte é determinado. O valor está de entre 0 e 1. Se for muito próximo de 0, leva a números instáveis, ou seja, muitos vetores de características são classificados incorretamente. Se for muito próximo de 1, o treino pode ser abortado e uma mensagem de erro do tratamento é gerada. Inicialmente, foi então definido para a taxa de erro esperada, um valor de 5% (0,05), sendo esse valor acrescido até ao valor de 13% (0,13) com o treino dado no programa. Geralmente, recomenda-se a busca simultânea por um par Nu-γ adequado, pois juntos definem o quão complexa é a hipersuperfície de separação. Isto pode ser aplicado na validação cruzada.

- **Parâmetro *Mode*** – Este parâmetro define se o problema em causa é de duas classes (*one-versus-one*), ou de múltiplas classes (one-versus-all). Para este caso, o SVM lida apenas com problemas de classes. O classificador binário é criado e a classe que possui mais comparações é selecionado. Aqui, n classes resultam em $n(n-1)/2$ classificadores. Para este projeto, foram necessários 3 classificadores.

- **Parâmetro *Preprocessing*** – Parâmetro interno que define o pré-processamento aplicado ao vetor de características para o treino, bem como posteriormente para a classificação ou avaliação. O pré-processamento é usado para acelerar a formação, bem como a classificação. Tem disponível o "*none*", "*normalization*", "principal_components" e "canonical_variates". Foi usado o "*normalization*" que permitiu uma melhor obtenção de informação relevante com uma velocidade maior.

- **Parâmetro *NumComponents*** – É um parâmetro de entrada que define o número de componentes, aos quais o vetor de características é reduzido se for selecionado um pré-processamento que reduza a dimensão desse mesmo vetor. Em particular, este parâmetro só é ajustado apenas se o parâmetro *Preprocessing* for definido

como "principal_components", como foi definido antes. O valor utilizado foi 5. Esta transformação é utilizada para obter o valor da média e do desvio do vetor de característica de valores entre 0 a 1, respetivamente.

- **Parâmetro *SVMHandle*** – Parâmetro de saída que vai ser utilizado para todas as classificações de operadores específicos.

E.2 Ajuste no operador *add_sample_class_svm*

Para o treino, as amostras são adicionadas neste operador, sendo sucessivamente chamadas com amostras diferentes. Os parâmetros de ajuste são os seguintes:

- **Parâmetro *SVMHandle*** – O parâmetro externo do operador *create_class_svm* e agora parâmetro interno deste operador. Após aplicado para todas as amostras disponíveis, o identificador está preparado para o treino real do classificador.

- **Parâmetro *Features*** – Parâmetro de entrada que contém o vetor de características de uma amostra a ser adicionada ao classificador com o operador *add_sample_class_svm*. O vetor de característica consiste em números reais. Esse vetor foi desenvolvido através do operador criado calculate_features.

- **Parâmetro *Class*** – Parâmetro entrada que contém o ID da classe, à qual o vetor de características pertence. O ID contém um número entre o 0 e o "Número de classes – 1". Foi criado um "*tuple*"com os nomes das classes denominado "*Classnames*". Essas *classnames* apresentam com os nomes "*good*"que representa as peças conformes, "*bad*"como não conformes e "*none*"nenhuma presença dos cárteres.

E.3 Ajuste no operador *train_class_svm*

Este operador faz o treino do classificador SVM, cujo parâmetros de ajuste são os seguintes:

- **Parâmetro *SVMHandle* -** Este parâmetro é o identificador do classificador que foi criado no operador *create_class_svm* e para os quais as amostras foram armazenadas no operador *add_sample_class_svm*. Depois de ser aplicado no operador *train_class_svm*, o identificador fica preparado para a classificação real de dados desconhecidos, ficando com as informações de como separar as classes.

- **Parâmetro *Epilson* -**Ao treinar o SVM significa otimizar gradualmente a função que determina os limites da classe. A otimização pára se o gradiente da função cair abaixo de um certo limite, esse limite define este parâmetro. Foi colocado um valor mínimo de 0,01.

- **Parâmetro *TrainMode* -** Parâmetro de entrada que define o tipo de treino selecionado. Para a maioria dos casos, incluindo o projeto, é utilizado o modo '*default*'.

E.4 Ajuste no operador *classify_class_svm*

Este operador é usado para decidir a qual das classes treinadas do vetor de recurso desconhecido pertence. Os parâmetros de ajuste são os seguintes:

- **Parâmetro *SVMHandle* -** É o identificador que foi criado no operador *create_class_svm*, onde as amostras foram adicionadas no *add_sample_class_svm* e que agora foram treinadas no operador *train_class_svm*. Assim contém todas as informações que o classificador precisa para atribuir um vetor de características desconhecidas para uma das classes disponíveis.
- **Parâmetro *Features* -** Contém o vetor de características do objeto a ser classificado. O vetor de características consiste nas mesmas características usadas para as amostras de treino dentro do operador *add_sample_class_svm*.
- **Parâmetro *Num* -** Específica o número das melhores classes a serem procuradas. Para o caso do projeto, definiu-se o valor 1 apenas à classe com maior probabilidade de ser procurada.
- **Parâmetro *Class* -** Este parâmtero retorna o resultado da classificação do vetor de características com o classificador SVM treinado. Neste caso, definiu-se o modo "one-versus-one", uma vez que contém as classes ordenadas pelo número de votos dos subclassificadores.

Apêndice F

Impressão 3D de peças suporte para as câmaras industriais

Os suportes de material discutido neste apêndice são um componente impresso em 3D projetado especificamente para fornecer estabilidade e funcionalidade das câmaras industriais usadas para o projeto. O suporte foi criado através do processo de manufatura aditiva, onde se obteve 3 peças suporte que foram essenciais para as câmaras que se encontram nas páginas abaixo.

[pages=1]Appendix/CAD/Peca$_c$*amara.pdf*[*pages* = 1]*Appendix/CAD/suporte*$_b$*aixo.pdf*[*pages* = 1]*Appendix/CAD/suporte*$_r$*ing.pdf*