Universidade de Aveiro Departamento de Engenharia Mecânica
2024

Beatriz Magalhães 103406

Edward Silva 76923

José Duarte 103892

Rastreabilidade da linha de produção de bombas de calor com Robô

Turma P4 Tema 5

Universidade de Aveiro Departamento de Engenharia Mecânica
2024

Beatriz Magalhães 103406

Edward Silva 76923

José Duarte 103892

Rastreabilidade da linha de produção de bombas de calor com Robô

Turma P4 Tema 5

Relatório da Unidade curricular de Projeto em Sistemas de Automação do Mestrado (Integrado) em Engenharia Mecânica (MIEM/MEM), realizado sob orientação de João Paulo Santos

palavras-chave: registo de dados, visão por computador, braço mecânico

resumo:

A utilização de sistemas de gestão de qualidade baseados em *software* permite rastrear peças defeituosas e gerir processos de não conformidade de forma eficiente. Estes sistemas permitem automatizar a aquisição de dados, fornecer análises detalhadas sobre problemas de qualidade e facilitam a comunicação entre os departamentos envolvidos na resolução dos mesmos.

Nesta linha de montagem, vão ser analisadas as bombas de calor para verificar se estas têm fugas ou não. Para isso, foi utilizada uma câmara que deteta variações de temperatura e, caso esta detete fugas, o braço mecânico é acionado de maneira a retirar a peça da linha. Todos estes passos vão ser registados no MQTT, que por sua vez vão ser armazenados na base de dados (mySQL).

Índice

Índice 6
Introdução 8
Proposta geral de Solução 8
Desenvolvimento da Solução 9
Linha de Montagem 9
Estações de Trabalho 9
Monitorização e processamento de dados 10
Ligação à base de dados 10
Leitura do Arquivo CSV e Preparação para Inserção em Lote 11
Processamento de dados 11
Extração das Variáveis da Mensagem MQTT 12
Verificação e Limpeza da Tabela 13
Visualização no Dashboard 13
Deteção de fugas 15
Sistemas existentes 15
Proposta de funcionamento 15
Montagem de testes 15
Câmara utilizada 16
Possibilidades de aquisição de imagem 17
Processamento de imagem 19

Problemas encontrados 22
Acionamento do braço mecânico 23
Esquema de ligação dos sensores 23
Comunicação com o braço mecânico 24
Conclusões e trabalhos futuros 27
Aquisição de dados 27
Visão por computador 27
Acionamento do braço mecânico 28
Referências 29

Introdução

Atualmente, numa linha de montagem de bombas de calor, é verificada a qualidade das ligações de forma manual. O objetivo deste projeto é auxiliar na automatização deste processo tornando-o mais eficiente e com menos intervenção humana.

Para isto serão executadas 3 tarefas:

- Verificação de fugas através de visão com auxílio de uma câmara IR;
- Deslocação de componentes defeituosos através de um braço robótico;
- Controlo de informação a partir de uma base de dados SQL.

Como repositório de scripts e ficheiros foi utilizado o GitHub [1].

Proposta geral de Solução

De forma resumida, o funcionamento geral é o seguinte: o componente avança na linha de montagem e para em frente à câmara IR. Um operador liga um tubo de ar comprimido ao componente, e o sistema da câmara envia a informação adquirida para a base de dados. É retirado o tubo de ar comprimido e o componente avança para a próxima posição onde, caso seja detetada fuga, o braço robótico retira a peça para fora da linha de montagem para que esta seja reparada posteriormente. O funcionamento deste sistema encontra-se representado na Figura 1.

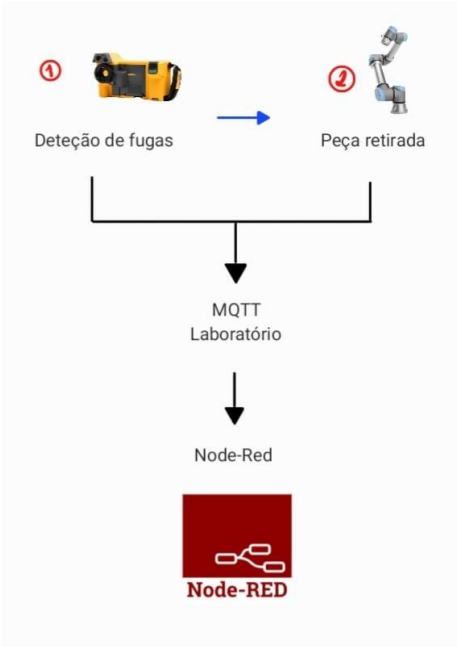


Figura 1- Esquema inicial do sistema

Desenvolvimento da Solução

Linha de Montagem

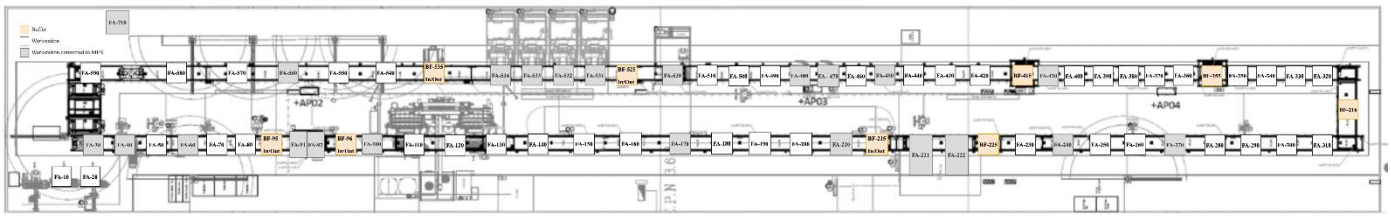


Figura 2 - Linha de montagem

A Figura 2 - Linha de montagemdescreve uma linha de produção de bombas de calor, proveniente da arquitetura de uma das linhas da *Bosch*. O *layout* inclui tanto as estações de trabalho conectadas, como não conectadas, ao Sistema MES *(Manufacturing Execution System)*.

A linha de produção é composta por estações de trabalho (FA) e buffers (BF) onde as estações conectadas ao MES são destacadas a cinzento. As estações estão numeradas sequencialmente e dispostas numa configuração linear e em forma de U, permitindo um fluxo contínuo do processo de montagem.

Os *buffers* são áreas de armazenamento temporário estrategicamente posicionados para gerir o fluxo de trabalho entre diferentes estações e evitar congestionamentos, ou *bottlenecking*. Estes são identificados pela sigla "BF" seguida de um número de identificação.

A linha de produção tem uma configuração que facilita a deslocação eficiente dos operadores.

Estações de Trabalho

As estações de trabalho são os locais onde ocorrem diferentes etapas do processo de montagem. Estas são identificadas pela sigla "FA" seguida de um número. A linha segue o seguinte formato:

- FA-10 a FA-30: Início da linha de produção.
- FA-40 a FA-180: Estações intermediárias antes da curva da linha.
- FA-190 a FA-310: Estações após a curva da linha.
- FA-320 a FA-710: Estações de finalização e extração do objeto.

As estações conectadas ao MES são essenciais para a integração digital da linha de produção, permitindo monitorizar e registar em tempo real.

Os *outputs* de dados do MES saem, em CSV, no seguinte formato, onde as *strings* dispõem de todas as informações consecutivamente, como apresentado na Figura 3.

```
LOCATION_RESULT_UID;UNIQUEPART_ID;LOCATION_ID;RESULT_DATE;PROCESS_NUMBER;PSTATINTERVAL;WORKCYCLE_COUNTER;RESULT_STATE;PART_ATTRIBUTE;TYPE_NUMBER;TYPE_VARIANT;TYPE_VE
EFBAE08574E049DEE0537FC2790AA082;837099912345699999999999;00000000001500010001100010001;22.12.14 13:56:03 EUROPE/LISBON;8881;;0;1;0;9999999999;0000;;;0;1;;23.03.14 13:56:03
EFBAE084CF6549DBE0537FC2790A5C22;837099912345699999999999;00000000001500010001100010001;22.12.14 13:58:42 EUROPE/LISBON;8888000;;0;1;0;9999999999;;;;0;1;;23.03.14 13:58:42
EFBAE084CF7549DBE0537FC2790A5C22;837099912345699999999999;00000000001500010001100010001;22.12.14 13:58:51 EUROPE/LISBON;8881;;1;1;0;9999999999;0000;;;0;1;;23.03.14 13:58:51
EFBAE084CF7849DBE0537FC2790A5C22;837099912345699999999999;00000000001500010001100010001;22.12.14 13:58:51 EUROPE/LISBON;8888000;;1;1;0;9999999999;;;;0;1;;23.03.14 13:58:51
EFBD987600C749E9E0537FC2790A83B5;837099912345699999999999;00000000001500010001100010001;22.12.14 14:18:12 EUROPE/LISBON;8881;;2;1;0;9999999999;0000;;;0;1;;23.03.14 14:18:12
EFBD987600C949E9E0537FC2790A83B5;837099912345699999999999;00000000001500010001100010001;22.12.14 14:18:13 EUROPE/LISBON;8888000;;2;1;0;9999999999;;;;0;1;;23.03.14 14:18:13
EFCD8B34530642CFE0537FC2790A0F62;837099912345699999999999;00000000001500010001100010001;22.12.14 16:51:17 EUROPE/LISBON;8888000;;3;2;3;9999999999;;;;0;1;;23.03.14 16:51:17
EFCDB36351436A76E0537FC2790A685D;837099912345699999999999;00000000001500010001100010001;22.12.14 17:05:36 EUROPE/LISBON;8888000;;4;2;3;9999999999;;;;0;1;;23.03.14 17:05:36
FAB13809048FB20AE05335C2790AC334;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:37:59 EUROPE/LISBON;3145;0;0;1;0;8738213464;0000;;;;1;;23.07.31 14:37:59
FAB15814F8D5B20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:44:08 EUROPE/LISBON;3145;0;1;2;0;8738213464;0000;;;;1;;23.07.31 14:44:08
FAB15814F8ECB20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010001;23.05.02 14:44:13 EUROPE/LISBON;3145000;0;0;2;3;8738213464;0000;;;;1;;23.07.31 14:44:13
FAB15814F98DB20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:46:15 EUROPE/LISBON;3145;0;2;1;0;8738213464;0000;;;;1;;23.07.31 14:46:15
FAB15814F9B4B20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:46:46 EUROPE/LISBON;3146;0;0;1;0;8738213464;0000;;;;1;;23.07.31 14:46:46
FAB15814F9B8B20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010001;23.05.02 14:46:48 EUROPE/LISBON;3145000;0;1;1;0;8738213464;0000;;;;1;;23.07.31 14:46:48
FAB15814FAB5B20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:50:33 EUROPE/LISBON;3145;0;3;1;0;8738213464;0000;;;;1;;23.07.31 14:50:33
FAB1380908ECB20AE05335C2790AC334;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:52:33 EUROPE/LISBON;3146;0;1;1;0;8738213464;0000;;;;1;;23.07.31 14:52:33
FAB1380908F0B20AE05335C2790AC334;837012312345687382134 64;00000000001500300001100010001;23.05.02 14:52:36 EUROPE/LISBON;3145000;0;2;1;0;8738213464;0000;;;;1;;23.07.31 14:52:36
FAB138090953B20AE05335C2790AC334;837012312345687382134 64;00000000001500300001100010001;23.05.02 14:53:40 EUROPE/LISBON;3145000;0;3;1;0;8738213464;0000;;;;1;;23.07.31 14:53:40
FAB138090954B20AE05335C2790AC334;837012312345687382134 64;00000000001500300001100010001;23.05.02 14:53:40 EUROPE/LISBON;3145000;0;4;1;0;8738213464;0000;;;;1;;23.07.31 14:53:40
FAB15814FB49B20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:54:56 EUROPE/LISBON;3145;0;4;1;0;8738213464;0000;;;;1;;23.07.31 14:54:56
FAB15814FB68B20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010011;23.05.02 14:55:47 EUROPE/LISBON;3146;0;2;1;0;8738213464;0000;;;;1;;23.07.31 14:55:47
FAB15814FB6CB20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010001;23.05.02 14:55:49 EUROPE/LISBON;3145000;0;5;1;0;8738213464;0000;;;;1;;23.07.31 14:55:49
FAB138090A7CB20AE05335C2790AC334;837012312345687382134 64;00000000001500300001100010001;23.05.02 14:57:26 EUROPE/LISBON;3145000;0;6;1;0;8738213464;0000;;;;1;;23.07.31 14:57:26
FAB138090BECB20AE05335C2790AC334;837012312345687382134 64;00000000001500300001100010011;23.05.02 15:02:29 EUROPE/LISBON;3145;0;5;1;0;8738213464;0000;;;;1;;23.07.31 15:02:29
FAB15814FD61B20CE05335C2790A9E96;837012312345687382134 64;00000000001500300001100010011;23.05.02 15:05:09 EUROPE/LISBON;3146;0;3;2;0;8738213464;0000;;;;1;;23.07.31 15:05:09
```

Figura 3 - Ficheiro output da linha de produção

Monitorização e processamento de dados

Ligação à base de dados

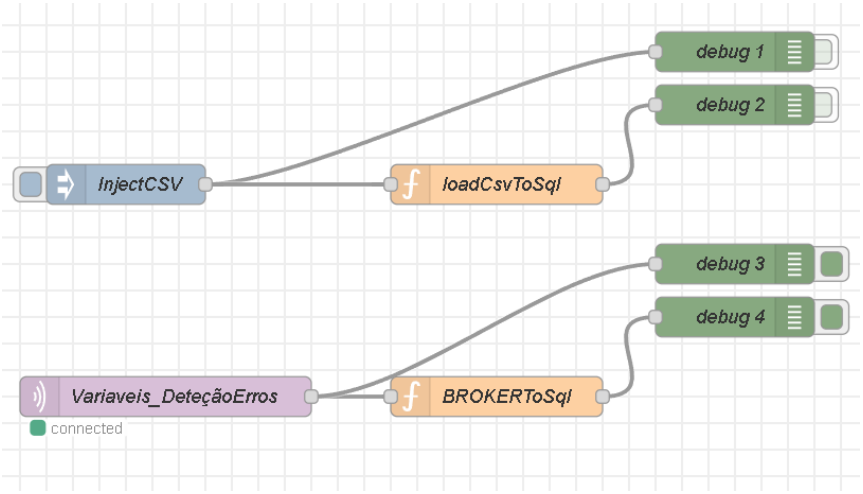


Figura 4 - Node-Red flow correspondente à aquisição de dados

Na Figura 4 está representado o *flow* desenvolvido em Node-Red com o objetivo de adquirir os dados do sistema desenvolvido.

O bloco função “loadCsvToSql”, contém um código em *Javascript,* realiza a leitura de um arquivo CSV, processa os seus dados e insere-os numa tabela MySQL. Devido à dimensão do ficheiro, este é inserido em lotes de 1000 registos.

Na secção que cria a conexão com a base de dados, os detalhes da conexão incluem o *host*, porta, utilizador, nome da base de dados e tempo limite de conexão de 60 segundos.

Leitura do Arquivo CSV e Preparação para Inserção em Lote

Inicialmente é feita a leitura do arquivo CSV, de *output* do MES. Caso ocorra um erro durante a leitura, este será registado e a execução será interrompida.

Os dados do CSV são depois divididos em linhas, onde cada linha é processada, removendo espaços em branco das extremidades, e armazenada no *array “rows”*.

Cada linha, exceto o cabeçalho, é dividida por colunas pelo delimitador “;” e os valores são armazenados no *array “values”*. Quando o número de valores atinge 1000 ou chega à última linha, a função *performBulkInsert* é invocada para inserir os dados na base de dados em lote. Após a inserção, o *array “values”* é reiniciado para o próximo lote.

O comando SQL INSERT INTO é depois utilizado para inserir os valores na tabela *LocationResults*.

Processamento de dados

O script lê os dados e insere-os na base de dados “linha_ed” em lotes, ou inserções de 1000 registos e utiliza leitura assíncrona de arquivos de forma a otimizar o desempenho. Este utiliza a biblioteca “MySQL” para conexão à base de dados e a biblioteca “fs” para leitura do arquivo CSV.

Após a injeção dos dados na base de dados, é feito o processamento destes de forma a simplificar a sua leitura e monitorização. Por exemplo, a porção da linha de dados correspondente à localização da peça (LOCATION_ID), na linha de produção foi simplificada, como se pode observar na Figura 5.

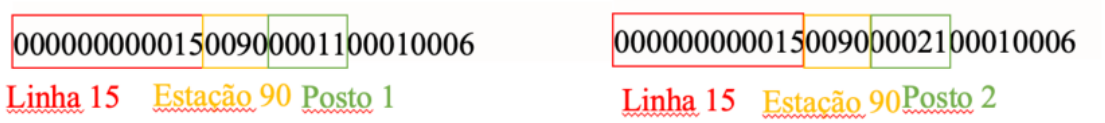


Figura 5 - Simplificação do string de localização da peça

A data e hora dos registos são posteriormente ordenados e são filtrados os valores em branco, permitindo reordenar os dados e visualizá-los de uma forma limpa.

Na Figura 6 encontra-se o percurso de uma peça aleatória (UNIQUEPART_ID), através das sucessivas estações de trabalho.

UNIQUEPART_ID	Linha	Estacao	Posto	RESULT_DATE
83703740000068738213471	000000000015	0020	00011	2023-06-06 10:42:59
83703740000068738213471	000000000015	0030	00011	2023-06-06 11:03:15
83703740000068738213471	000000000015	0040	00011	2023-06-06 11:12:18
83703740000068738213471	000000000015	0060	00011	2023-06-06 13:38:56
83703740000068738213471	000000000015	0090	00021	2023-06-06 14:20:13
83703740000068738213471	000000000015	0100	00011	2023-06-06 15:23:58
83703740000068738213471	000000000015	0170	00011	2023-06-07 10:19:19
83703740000068738213471	000000000015	0210	00011	2023-06-07 11:16:02
83703740000068738213471	000000000015	0220	00031	2023-06-12 08:01:12
83703740000068738213471	000000000015	0240	00011	2023-06-12 08:02:25
83703740000068738213471	000000000015	0270	00011	2023-06-12 08:11:12
83703740000068738213471	000000000015	0410	00011	2023-06-12 08:57:40
83703740000068738213471	000000000015	0450	00011	2023-06-12 09:46:47
83703740000068738213471	000000000015	0470	00011	2023-06-12 09:58:30
83703740000068738213471	000000000015	0480	00011	2023-06-12 10:02:33

Figura 6 - Percurso de uma peça na linha

Ainda relativamente à Figura 6, o bloco "*BROKERtoSql*" insere dados recebidos via MQTT, sob certas condições, na base de dados MySQL.

A secção inicial cria, novamente, uma conexão com a base de dados MySQL cujos detalhes da conexão são os mesmos que no bloco anterior. Em caso de falha, um erro é registado e a execução é interrompida. Se a conexão for bem-sucedida, uma mensagem de sucesso é exibida.

Extração das Variáveis da Mensagem MQTT

As variáveis são extraídas, do *payload* da mensagem MQTT, e os valores são registados para verificação.

O *querry* SQL para inserir os valores na tabela “fugas” é definido, contendo “?” como *placeholders* que serão substituídos pelos valores das variáveis. A consulta SQL é executada com os valores das variáveis. Em caso de erro, este é registado com *node.error*.

Verificação e Limpeza da Tabela

É imposta uma condição *if* que diz que, se o valor de N_Comp (contador de peças registadas no turno) for 0, ou *string* vazia, a função *clearTable* é chamada para limpar a tabela “fugas”. Esta função executa o comando “TRUNCATE TABLE fugas” para limpar a tabela.

A conexão com a base de dados é depois fechada e o script retorna a mensagem original para continuar o fluxo de processamento.

Visualização no Dashboard

Para permitir que a *interface* do *Node-Red* leia a base de dados, utilizou-se um bloco *mysql,* ligado a um bloco de tabela, do *dashboard*., representado na Figura 7.

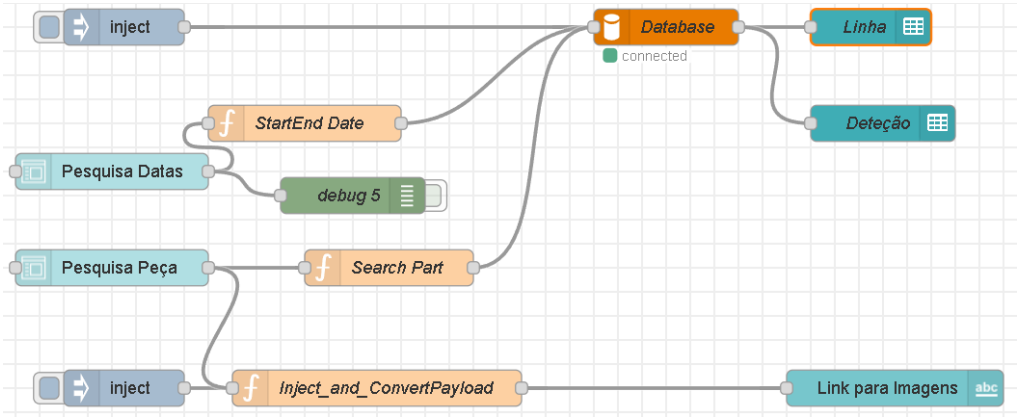


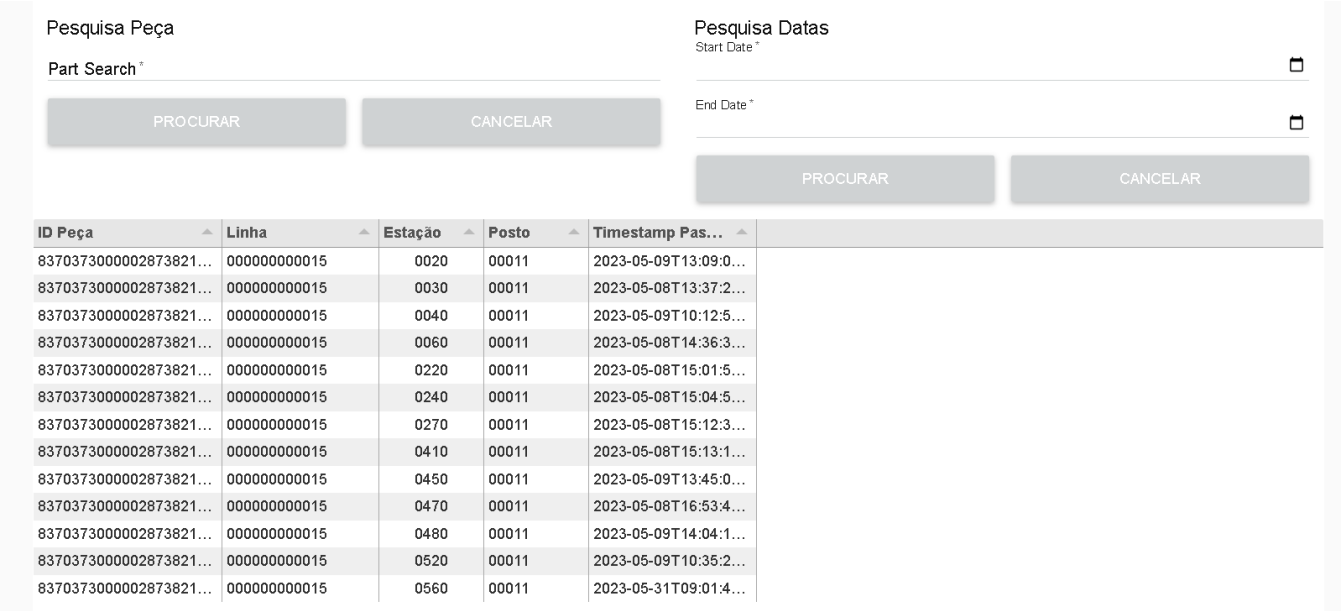
Figura 7 - Flow de visualização dos dados

O flow pode ser ativado manualmente para iniciar a leitura de dados, ou procurar dados de um intervalo de datas específico, de forma a verificar dados de turnos anteriores.

O utilizador preenche um formulário no nó *Form* cujos dados são enviados ao nó *StartEnd Date*. Este processa as datas de início e término do formulário, que são enviadas ao nó *Database*. Alternativamente, o utilizador pode também pesquisar o registo de uma peça específica.

O nó *Database* recebe dados tanto do *Search Part* como do *StartEnd Date* e executa a consulta à base de dados. Os resultados da consulta são enviados às tabelas Linha e Deteção.

Pode assim ser observado o registo no próprio browser, filtrando, por número de peça (p.e. 83703730000028738213464) ou data e/ou hora, como se pode observar na Figura 8.



ID Peça	Linha	Estação	Posto	Timestamp Pas…
8370737000002873821…	000000000015	0020	00011	2023-05-09T13:09:0…
8370737000002873821…	000000000015	0030	00011	2023-05-08T13:37:2…
8370737000002873821…	000000000015	0040	00011	2023-05-09T10:12:5…
8370737000002873821…	000000000015	0060	00011	2023-05-08T14:36:3…
8370737000002873821…	000000000015	0220	00011	2023-05-08T15:01:5…
8370737000002873821…	000000000015	0240	00011	2023-05-08T15:04:5…
8370737000002873821…	000000000015	0270	00011	2023-05-08T15:12:3…
8370737000002873821…	000000000015	0410	00011	2023-05-08T15:13:1…
8370737000002873821…	000000000015	0450	00011	2023-05-09T13:45:0…
8370737000002873821…	000000000015	0470	00011	2023-05-08T16:53:4…
8370737000002873821…	000000000015	0480	00011	2023-05-09T14:04:1…
8370737000002873821…	000000000015	0520	00011	2023-05-09T10:35:2…
8370737000002873821…	000000000015	0560	00011	2023-05-31T09:01:4…

Figura 8 - Dashboard da visualização de dados no Node-Red

Deteção de fugas

Sistemas existentes

Na linha de montagem, para a verificação de fugas, um tubo proveniente de um regulador de pressão é inserido no produto, pressurizando o sistema com ar comprimido. Após a pressurização do sistema, o fornecimento de ar é desligado e a pressão é verificada através de um dispositivo de verificação de fugas (ATEQ). Se a pressão se mantiver constante, não há fuga, se a pressão diminuir, existe uma fuga. Quando é determinado que existe fuga, os técnicos removem o produto da linha de montagem e todas as juntas são novamente soldadas. Este processo pode-se observar na Figura 9.

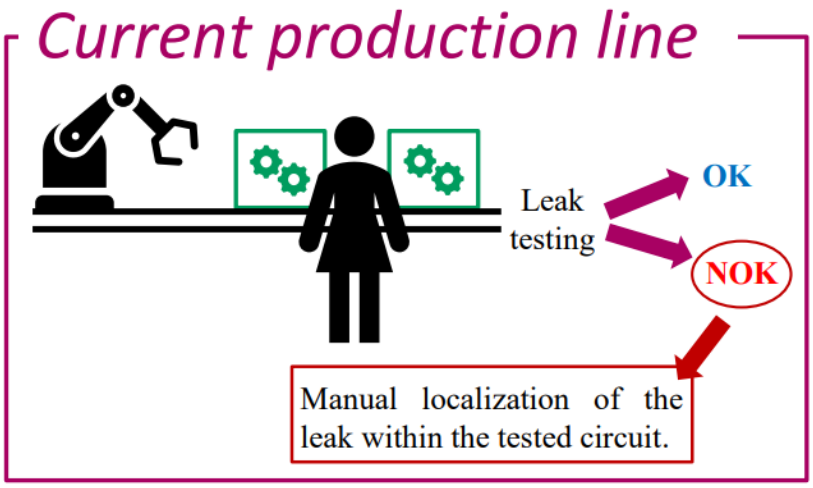


Figura 9 - Representação esquemática da atual linha de produção

Proposta de funcionamento

Através de uma câmara de infravermelhos e de um *software* de deteção, é possível não só determinar a existência de uma fuga, mas também localizar precisamente o ponto da fuga. Desta forma, evita-se a perda de tempo na revisão de todas as ligações, intervindo apenas no local específico da fuga, o que aumenta a eficiência do processo, o custo de reparação e a produtividade da fábrica.

Montagem de testes

Para a realização dos testes, foi montado o sistema ilustrado na Figura 10. Neste sistema, podemos observar o regulador de pressão, a câmara IR e uma peça especificamente desenhada para testes. Esta peça contém fugas de vários diâmetros diferentes (3 mm, 2,5 mm, 2 mm, 1 mm, 0,5 mm, 0,25 mm)

e é utilizada para simular as fugas que eventualmente surgem nas peças em chão de fábrica. Para alterar a fuga a ser simulada, basta rodar a peça.

O circuito está montado dentro de uma caixa para simular um ambiente fechado, sem interferências de luz ou calor do seu arredor. Embora para o olho humano isto seja suficiente, para um programa de computador pode gerar problemas, os quais serão discutidos posteriormente.

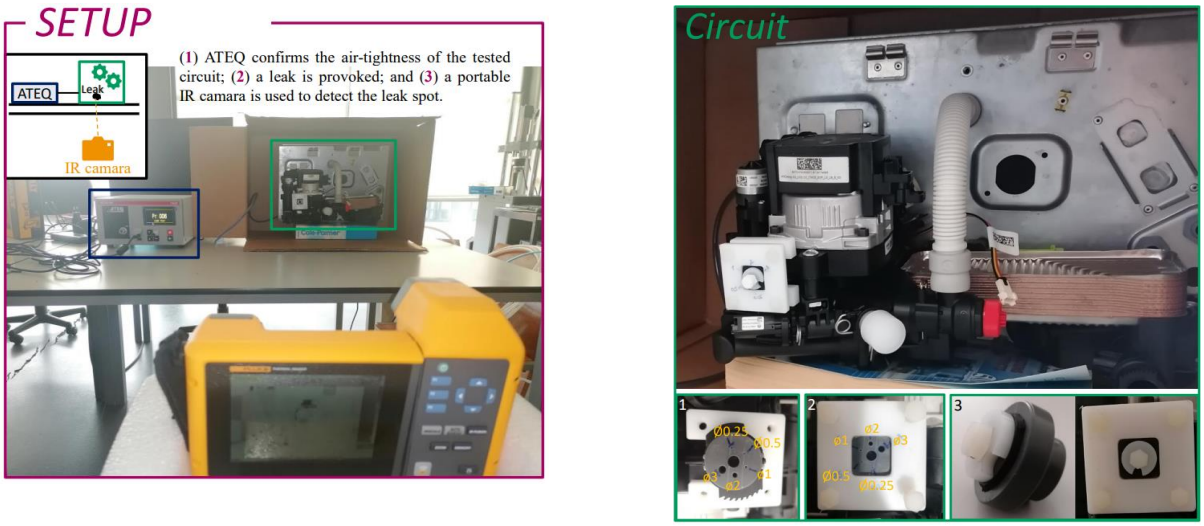


Figura 10 - Montagem de testes

Câmara utilizada

Para a visualização da fuga, foi utilizada uma câmara de infravermelhos, mais especificamente o modelo Expert Fluke TiX580, conforme mostrado na Figura 11.

Os termovisores da série Expert Fluke são câmaras termográficas portáteis adequadas para diversas aplicações, incluindo a solução de problemas de equipamentos, manutenção preditiva e preventiva programada, diagnósticos de edifícios, e pesquisa e desenvolvimento. O termovisor exibe imagens térmicas numa tela de toque LCD de alta visibilidade e qualidade industrial, podendo armazenar imagens na memória interna, num cartão de memória removível ou num dispositivo de armazenamento por USB. As imagens e os dados armazenados na memória interna ou no cartão de memória podem ser transferidos para um PC através de uma conexão USB direta ou por transferência sem fios para um PC ou dispositivo móvel.

O termovisor vem equipado com o *software* SmartView, um pacote profissional de alto desempenho que permite análises e geração de relatórios de qualidade, como demonstrado na Figura 12. No entanto, sendo este um *software* apenas de visualização, que não permite processamento de imagem em tempo real, foi necessário encontrar uma solução alternativa para este problema.

Devido ao facto de a câmara não comunicar com o computador de forma convencional (esta não surge como um dispositivo “câmara”, mas sim como “placa de rede”), não é possível obter o vídeo em tempo real de forma direta utilizando por exemplo a linguagem Python. No entanto, a câmara possui compatibilidade com o *software* MATLAB, estando disponibilizada uma *toolkit* para estes efeitos.

Figura 11 - Câmara IR (Infravermelho)

Figura 12 - Visualização de Imagem por SmartView

Possibilidades de aquisição de imagem

MatLab

A câmara Expert Fluke TiX580 é compatível com o MatLab, permitindo aceder às imagens de forma direta utilizando os scripts e a documentação disponibilizados no repositório GitHub [3].

Esta toolkit disponibiliza ao utilizador uma série de funções (ToolkitFunctions.m) que, através da biblioteca da Fluke para MATLAB, permitem adquirir as imagens em tempo real da câmara, obter valores de temperatura, etc. Todas estas funções estão implementadas num GUI (FlukeMatlabToolkit.mlapp) que facilita a utilização que pode ser visto na Figura 13.

Alguns cuidados a ter ao utilizar o MatLab incluem garantir que todos os ficheiros estão no diretório correto e que o *software* SmartView não está aberto, uma vez que a câmara só consegue enviar informação para um *software* de cada vez. Este foi o principal problema encontrado, pois a conexão entre a câmara e o computador é bastante instável. Por vezes, é necessário terminar as tarefas no gestor de tarefas e/ou reiniciar todos os *softwares* para que o sistema funcione corretamente.

Python com MatLab Engine

Visto que os integrantes do projeto estão mais familiarizados com Python, também foi estudada uma forma de aceder às imagens utilizando esta linguagem. Não sendo possível realizar este acesso por meios diretos, recorreu-se à utilização de um MatLab Engine, o qual permite utilizar os comandos e funções do MatLab fora do seu ambiente. Através deste processo, foi possível adquirir as imagens e

transformá-las para um formato que pode ser processado pelas bibliotecas de Python como podemos ver na Figura 14.

Mais uma vez, é necessário garantir que apenas um *software* de visualização esteja aberto, caso contrário, o sistema não funcionará. Além disso, é importante notar que o MatLab utiliza a convenção de cores RGB, enquanto algumas bibliotecas de Python, como o OpenCV, utilizam a convenção BGR.

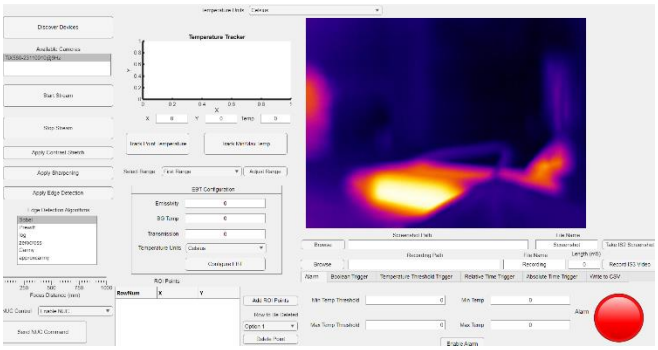


Figura 13 - Visualização de imagem através do MatLab- ToolKit GUI

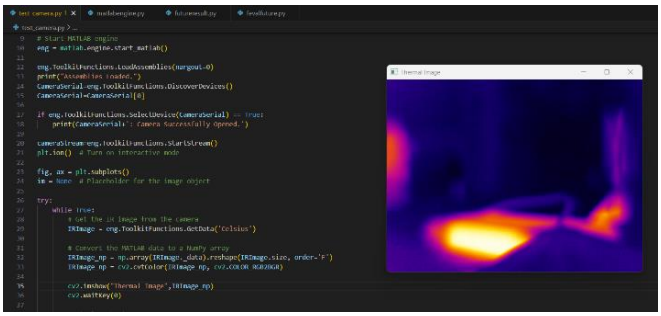


Figura 14 - Visualização de imagem através de Python

Monitor duplo

Com o sistema de monitor duplo, o vídeo será transmitido ao vivo para um monitor utilizando o *software* SmartView, visto que este é o mais fácil de interagir com a câmara. O computador vai gravar a tela desse monitor e fazer o processamento de imagem a partir dessa gravação, a sua montagem pode ser vista na Figura 15.

Para que este sistema funcione corretamente, é necessário ter alguns cuidados. Nomeadamente, é essencial garantir que o monitor tem uma resolução superior à do vídeo exportado para não perder qualidade. Além disso, é crucial assegurar que não haja um atraso percetível entre as imagens, de modo a que este sistema possa ser aplicado em tempo real.

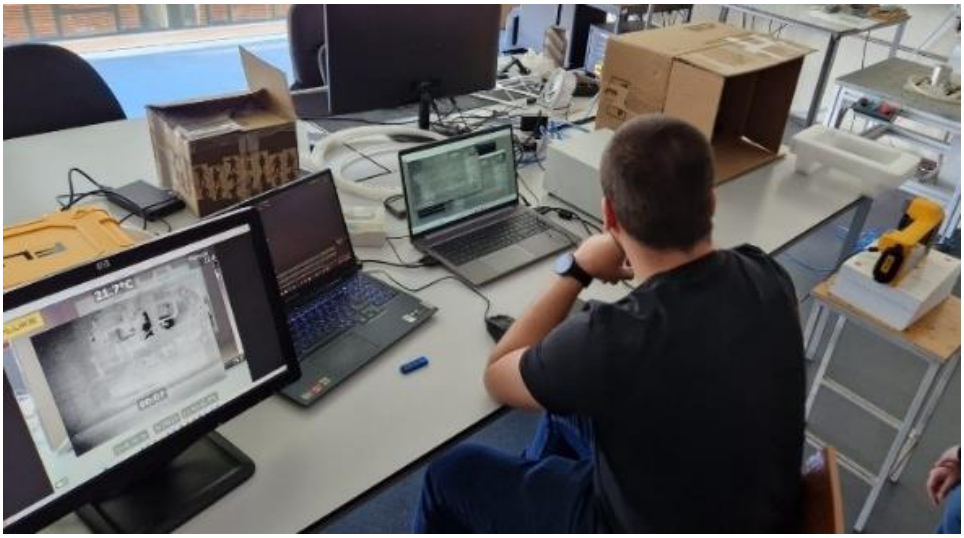


Figura 15- Sistema monitor duplo

Processamento de imagem

Não houve um sistema escolhido como preferencial porque todos apresentam diferentes vantagens e problemas que acabam por depender da equipa que irá implementar o sistema. Como este projeto é apenas para fins pedagógicos e de investigação, não há necessidade de fazer esta escolha.

Foram então criados scripts de aquisição de imagem considerando cada uma das três opções [3][4][5] ,confirmando que são viáveis, e um script de teste para demonstração, utilizando uma gravação previamente guardada. Como a montagem experimental só permite uma fuga e o dispositivo está estacionário, foi criado um vídeo para efeitos de teste, que simula a linha de montagem, com o dispositivo a aparecer, parar, ser analisado e retirado.

Para testar várias fugas no mesmo componente, no final do vídeo aparece uma montagem de três vídeos com *delay*, para que as fugas apareçam em diferentes alturas e localizações no mesmo componente.

O passo seguinte foi desenvolver o *software* de deteção de fugas, utilizando Python pelas razões mencionadas anteriormente. Primeiramente, é necessária uma imagem para *template* onde apenas o componente ou produto a verificar esteja visível. Com esta imagem, será criada uma máscara para o programa identificar automaticamente se o componente está presente e onde está localizado, criando assim uma ROI (*Region Of Interest*) como se pode observar na Figura 16. Depois de definida a ROI, é realizado o processamento da imagem. A partir da intensidade dos pixéis encontrados na ROI, o programa determina se há ou não fuga.

Este método permite uma identificação precisa do componente e a análise eficiente das fugas, garantindo que o processo de verificação é automatizado e eficaz.

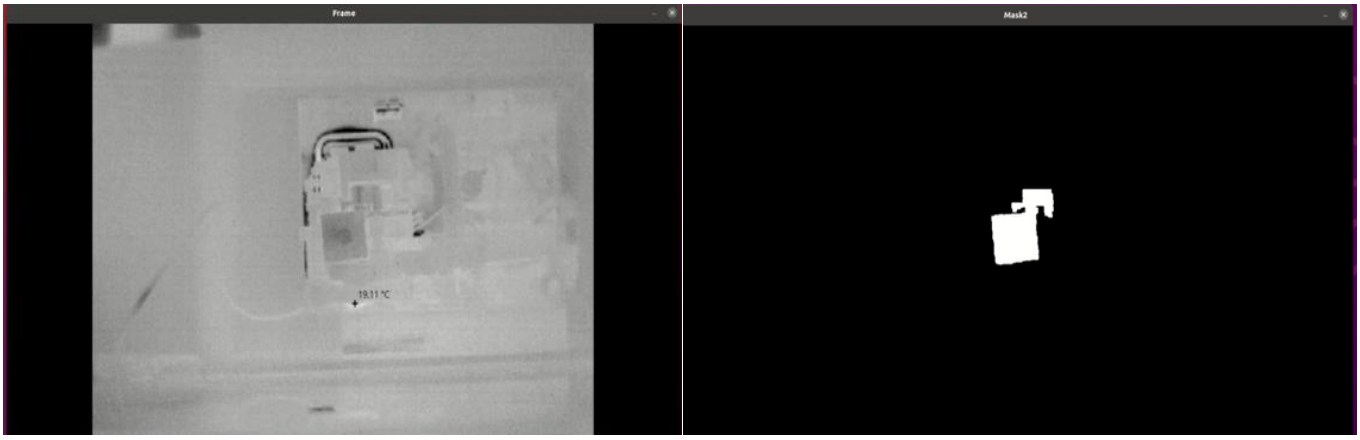


Figura 16- Frame e respetiva ROI identificada

Quando é encontrada uma fuga, como pode ser observado nas Figura 17 e Figura 18, a sua localização e arredores são removidos da análise para evitar a deteção de múltiplas fugas no mesmo local. Após esta etapa, o *frame* atual é guardado e uma legenda é adicionada para indicar a localização da fuga. Esta imagem é armazenada numa pasta identificada com o número do componente e o número da fuga, para que, após o componente ser retirado, o operador saiba exatamente qual é o componente e onde se encontra a fuga como demonstrado nas Figura 19 e Figura 20.

Terminada a verificação do componente, a informação necessária, como o número do componente, a presença ou ausência de fugas e a quantidade de fugas, é enviada via MQTT. As imagens com as fugas identificadas são automaticamente inseridas no repositório do GitHub [6], permitindo que o operador responsável pela reparação dos componentes possa aceder a estas informações posteriormente. Uma demonstração deste processo pode ser vista no vídeo que se encontra no

Youtube [7]. Este processo automatizado garante a rastreabilidade e facilita a intervenção rápida e precisa na reparação dos componentes.

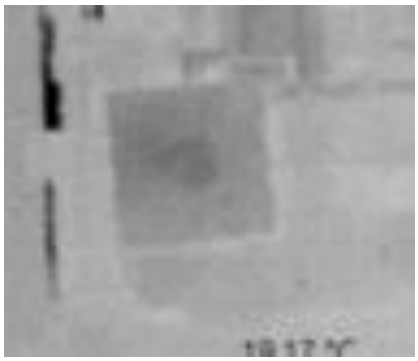


Figura 17-componente sem fuga

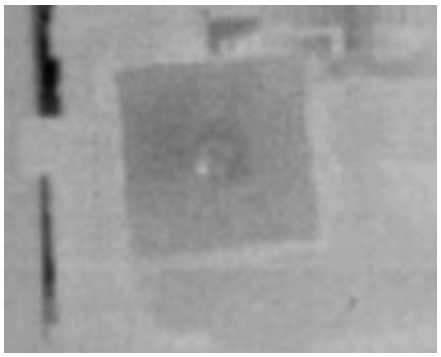


Figura 18-Componente com fuga

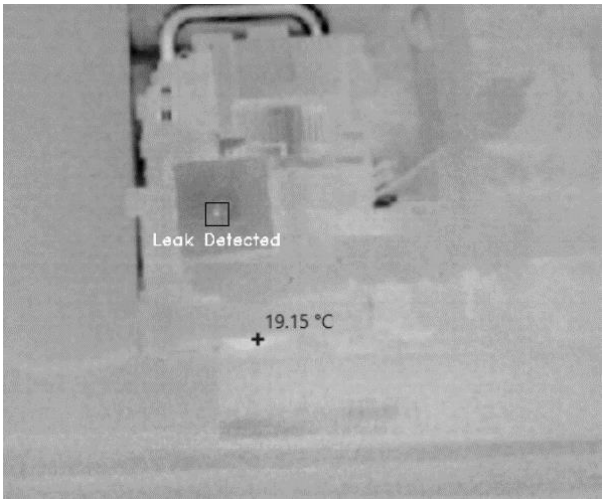


Figura 19-Imagem guardada

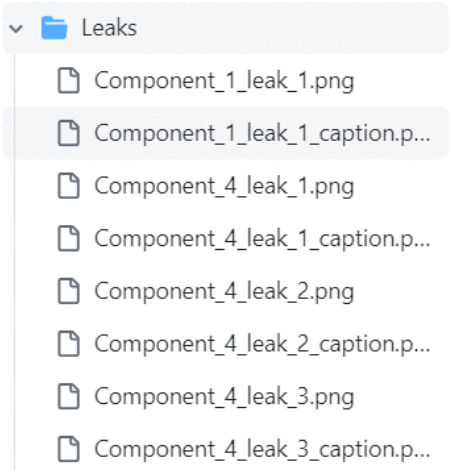


Figura 20- Identificação das imagens

Problemas encontrados

Com a utilização de visão tradicional por Python, é possível encontrar as fugas, como foi demonstrado anteriormente. No entanto, este processo apresenta falhas por diversas razões. Primeiramente, a câmara utilizada faz uma regulação automática do brilho da imagem, que não pode ser desativada. A Figura 21 - Variações de luminosidade observada mostra a junção de quatro *frames* do mesmo vídeo, capturados com alguns segundos de diferença, evidenciando grandes variações de luminosidade mesmo mantendo condições externas constantes.

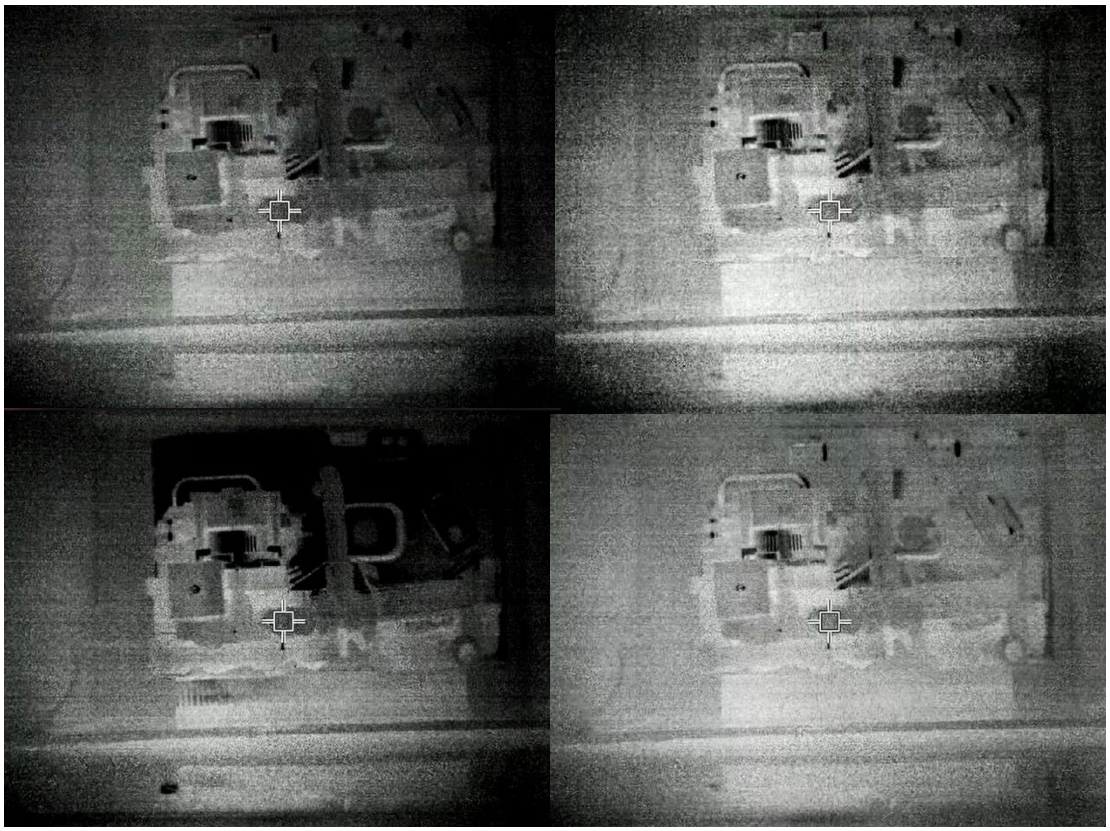


Figura 21 - Variações de luminosidade observada

Portanto, a utilização de visão tradicional apresenta dois principais problemas. Primeiro, a determinação das fugas teria de ser feita a partir de um cálculo que considera a luminosidade geral da imagem, o que pode ser difícil de determinar e varia muito com as condições de operação, tais como a temperatura ambiente, a temperatura do gás em relação ao componente, a quantidade de luz, etc. Segundo, torna-se impossível determinar uma ROI fiável, uma vez que isto depende não apenas da forma da imagem, mas também da luminosidade dos seus pixéis.

Estes problemas serão melhor discutidos mais á frente na secção “Conclusões e trabalhos futuros”.

Acionamento do braço mecânico

Esquema de ligação dos sensores

Após realizada a deteção de fugas das bombas de calor, esta vai dar seguimento na linha de montagem. O próximo passo é retirar a bomba de calor caso esta tenha fugas. Para isso foi usado um sensor indutivo que deteta se existe uma bomba de calor no local da extração e, caso esteja nesse local, o sensor envia um sinal ao braço mecânico de modo que este retire a mesma do local.

Assim, em primeiro lugar foi necessário fazer o esquema de ligação do sensor indutivo. Para isto foi necessário usar um ESP32, um optoacoplador, uma fonte de alimentação de 24V e o próprio sensor indutivo PNP.

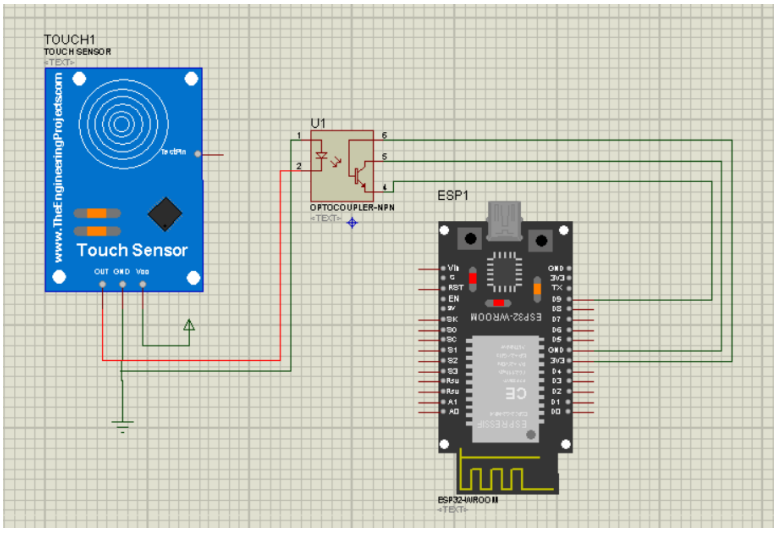


Figura 22 - Esquema de ligação do sensor indutivo

Na Figura 22 está um esquema de ligação que demonstra o que acontece na realidade. Neste caso, o sensor e o ESP32 wroom-32 não podem estar ligados diretamente pois o sensor necessita de 24V para o seu funcionamento enquanto o CPU (*Central Processing Unit*) do ESP32 só aguenta até 3,3V de tensão. Para esta ligação ser fazível de modo que possa haver posteriormente uma ligação entre o sensor e o braço mecânico, é usado um opto acoplador em que se pode fazer ligação de dois componentes com tensões diferentes.

Posto isto, é realizada a ligação dos componentes, demonstrada na Figura 23.

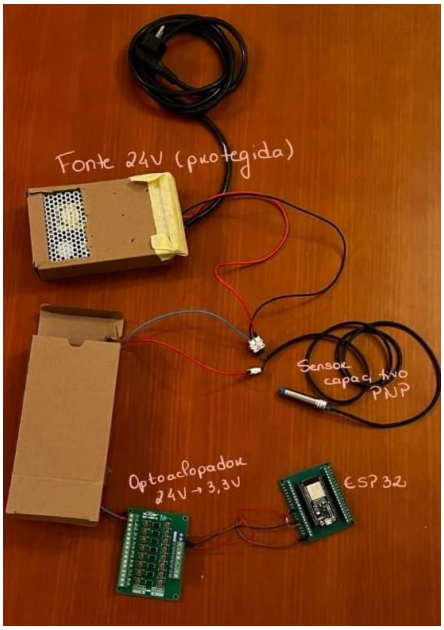


Figura 23 - Ligação sensor indutivo

Comunicação com o braço mecânico

Realizada a montagem do sensor indutivo de maneira que este consiga detetar caso haja uma bomba de calor na linha de montagem que tenha necessidade de ser retirada devido a fugas foi necessário interligar o braço mecânico com o sensor indutivo.

Primeiramente, vai ser usado o braço mecânico UR5e, demonstrado na Figura 24. O braço mecânico UR5e é um robô colaborativo (Cobot) desenvolvido pela Universal Robots, utilizado em atividades industriais em simultâneo com seres humanos como função de auxílio em funções repetitivas (movimentos de componentes), fisicamente demandantes (levantamento de cargas) e/ou perigosas (solda de componentes). Neste caso vai ser usado para retirar uma peça da linha de montagem.

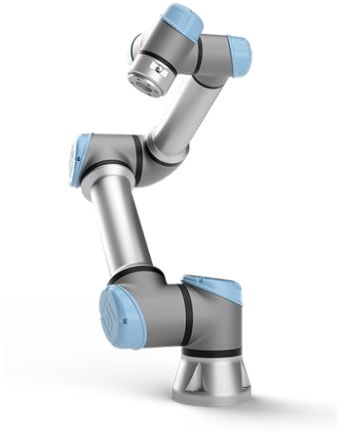


Figura 24 - Braço mecânico UR5e

Para isto, foi usado o programa Arduino IDE com auxílio do ESP32 montado anteriormente. O Arduino IDE é um *software* gratuito e multiplataforma que permite programar placas Arduino e outras placas compatíveis (neste caso, ESP32). O ESP32 destaca-se como sendo um microcontrolador de baixo custo, porém de alta performance que oferece recursos robustos que a tornam ideal para diversos projetos, sendo este um desses casos.

Antes de passar para a programação, foi necessário estabelecer ligação do braço mecânico a um router de modo que o ESP32 possa aceder ao mesmo. Desta maneira, o ESP32 não precisa de estar no mesmo local que o braço mecânico, facilitando a montagem de ambos. Neste caso foi utilizado o router do Laboratório Sistema Flexível de Produção (LSFP). Para isto foi preciso inserir o IP do braço mecânico, sendo neste caso 192.168.1.8, com a porta 30002, no router. A partir deste momento, sempre que for necessário aceder ao braço mecânico, este irá estar no IP 193.137.172.20 (IP do router), com a porta 90 (porta selecionada dentro do router para representar o braço mecânico).

Posto isto, o próximo passo é realizar a ligação do ESP32 ao braço mecânico. Para isto, foi desenvolvido o primeiro código em que se interliga o sensor indutivo com o braço mecânico através do ESP32. Este código tem como fundamento os seguintes passos:

- Realizar a ligação do ESP32 com o braço mecânico através de uma ligação WI-FI, neste caso a rede PA movel de um telemóvel.
- Com esta ligação realizada, o braço mecânico fica em modo espera, estando totalmente dependente das informações enviadas pelo sensor indutivo.
- Assim, se o sensor indutivo detetar uma peça (para este caso, o valor do sensor passa a 0), vai ser ativada uma variável em que fica registado que o objeto foi detetado (com valor 1). Assim que a peça avance na linha de montagem, o sensor indutivo vai passar a 0, ou seja, deixa de detetar a peça, no entanto, a variável objeto detetado continua a estar ativa.
- Com estas duas variáveis definidas desta maneira, o braço mecânico é executado, movendo a peça do sítio. Quando este acaba o seu movimento, desativa a variável que define que o objeto está detetado, passando a 0.
- No final deste processo, as variáveis voltam a estar com as mesmas características que estavam no início, e assim o código é repetido sempre que houver uma nova peça que seja necessário remover do tapete.

Este código pode ser observado no GitHub [8].

Com esta parte do código a funcionar, foi necessário acrescentar linhas de código de modo que permitem mandar a informação ao broker MQTT. O MQTT estabelece-se com um protocolo de comunicação leve e robusto, especialmente direcionado para aplicações da IoT (*Internet of Things*). Este permite a troca assíncrona de mensagens entre dispositivos e sistemas, caracterizando-se pela

alta ampliabilidade e adequando-se perfeitamente a cenários que exigem comunicação em tempo real, sendo este um desses casos. Neste caso foi usado o broker que também estava conectado ao router do laboratório LSFP, o mesmo em que o braço mecânico está conectado. Este foi um dos grandes desafios a combater, pois, em fases de testes e de desenvolvimento do código, estes dois dispositivos estavam conectados a dois *routers* diferentes, havendo um conflito com a ligação aos dois dispositivos ao mesmo tempo. Para contornar este problema foi então conectado o braço mecânico ao mesmo router em que está conectado o MQTT, deixando de existir este problema, ficando com o IP apresentado no início deste capítulo. A Figura 25 representa o esquema de ligação entre todos os dispositivos utilizados, bem como os IP e as portas associados.

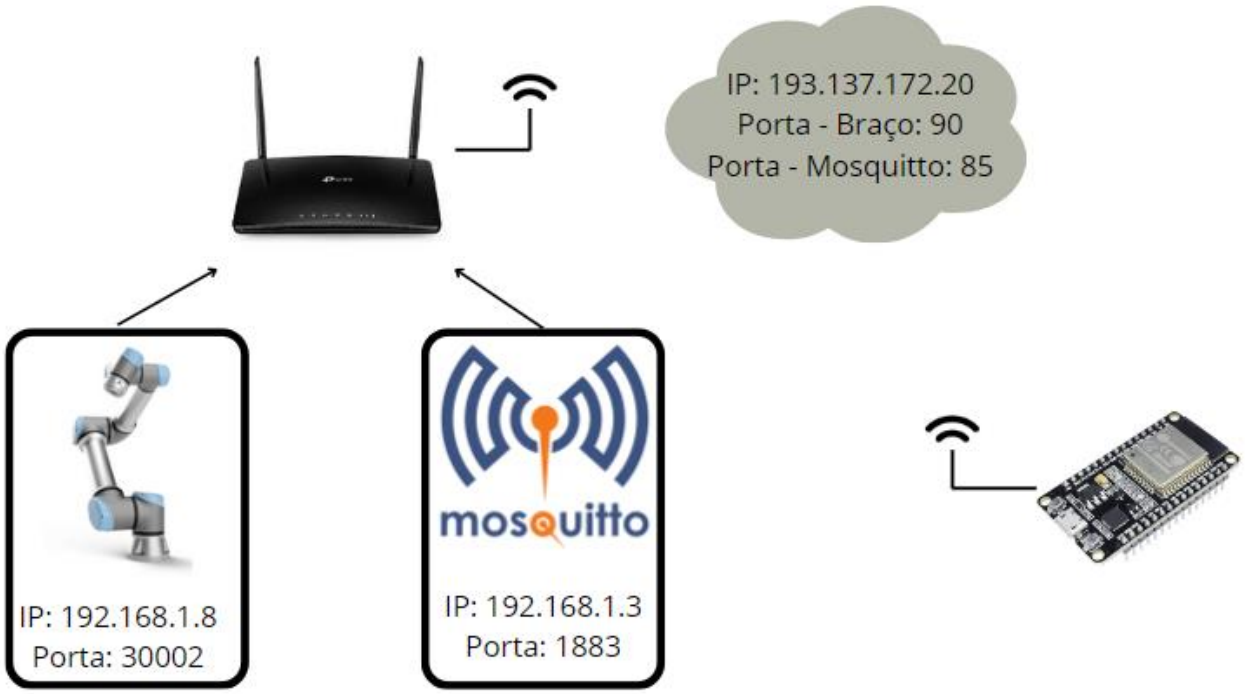


Figura 25 - Esquema de comunicações

Resolvido o problema, foi então enviado como mensagem para o broker se o robô foi ativado ou não. Assim, o código final desenvolvido está apresentado no GitHub [8].

A demonstração do funcionamento do braço mecânico através do sensor indutivo encontra-se no vídeo no Youtube [9].

Conclusões e trabalhos futuros

Aquisição de dados

Atualmente, o sistema monitora e processa dados de fugas, mas não inclui a visualização de imagens térmicas em tempo real. A adição desta funcionalidade envolveria a captura contínua de imagens pela câmara térmica. Cada vez que uma nova imagem fosse capturada, ela seria automaticamente carregada e transmitida para o broker MQTT.

O Node-Red seria uma plataforma ideal para integrar a visualização de imagens devido à sua capacidade de processamento de fluxos de dados em tempo real. As imagens capturadas seriam enviadas para o *dashboard* e, para evitar a sobrecarga de informações, poderiam ser configuradas para desaparecer após alguns segundos, seguindo a cadência de produção.

Para uma gestão eficiente, seria possível implementar duas opções para o tratamento das imagens:

• Visualização Temporária: As imagens poderiam ser visualizadas no *dashboard* e apagadas automaticamente após um curto período, assegurando que apenas a imagem mais recente seja exibida.

• Armazenamento em Base de Dados: Alternativamente, as imagens poderiam ser armazenadas numa base de dados até o fim do turno. Isso permitiria uma análise posterior, facilitando a identificação de padrões e a tomada de decisões importantes.

Visão por computador

Uma forma de contornar os problemas mencionados anteriormente seria a utilização de inteligência artificial. Em vez de uma única imagem como *template*, pode-se utilizar um conjunto de imagens e treinar uma rede neuronal para detetar o componente independentemente da luminosidade geral da imagem, melhorando assim a precisão e a confiabilidade da deteção de fugas. Esta abordagem permitiria um reconhecimento mais robusto dos componentes, independentemente das variações de luminosidade e outras condições operacionais. Além disso, existem *software*s mais dedicados à procura de padrões, que utilizam outras linguagens, sendo um exemplo o Sherlock.

Outro problema é a disponibilidade limitada dos *software*s da Fluke. Embora o sistema de monitor duplo ou o uso do MatLab possam contornar isso, essas soluções não são sustentáveis a nível industrial devido à sua inconveniência e ao custo monetário.

Para uma solução eficiente e prática, é necessário desenvolver um *software* que permita um pós-processamento de imagem adequado, superando desafios como variações de luminosidade. Este

software deve integrar-se facilmente com os sistemas da fábrica, garantindo uma deteção e reparação de fugas mais eficaz e econômica.

Acionamento do braço mecânico

Apesar de o braço mecânico estar funcional e a trabalhar, existe outra possibilidade para o seu acionamento. Esta solução implica que o braço mecânico só é acionado quando a base de dados receber a informação que foi detetada uma fuga, sendo que, por sua vez, o Node-Red envia a informação para o ESP32 e este envia a ordem ao braço mecânico.

Referências

[1] Ze6000. (n.d.). PSA-project. GitHub. Retrieved June 17, 2024, from https://github.com/Ze6000/PSA-project

[2] Nițulescu, I.; Korodi, A.; Supervisory Control and Data Acquisition Approach in Node-RED: Application and Discussions. Faculty of Automation and Computers, University Politehnica Timișoara. **2020**, [**Google Scholar**] [**CrossRef**]

[3] Ze6000. (n.d.). PSA-project: Fluke RSE MATLAB (R2019a)/MATLAB Toolkit [Computer software]. GitHub. Retrieved June 17, 2024, from https://github.com/Ze6000/PSA-project/tree/main/Fluke%20RSE%20MATLAB%20(R2019a)/MATLAB%20Toolkit

[4] Ze6000. (n.d.). PSA-project: Script python_matlab [Computer software]. GitHub. Retrieved June 17, 2024, from https://github.com/Ze6000/PSA-project/tree/main/Script%20python_matlab

[5] Ze6000. (n.d.). PSA-project: Dual Monitor [Computer software]. GitHub. Retrieved June 17, 2024, from https://github.com/Ze6000/PSA-project/tree/main/Dual%20Monitor

[6] Ze6000. (n.d.). PSA-project: Leaks [Computer software]. GitHub. Retrieved June 17, 2024, from https://github.com/Ze6000/PSA-project/tree/main/Leaks

[7] José Duarte. (2024). *Demonstração do software de Visão* [Vídeo]. YouTube.

(https://youtu.be/Cqi7-2zEMSc?si=uo72SQsTsv5ViKt0)

[8] Ze6000. (n.d.). PSA-project: Acionamento do braço mecânico [Computer software]. GitHub. Retrieved June 17, 2024, from https://github.com/Ze6000/PSA-project/tree/main/Acionamento%20do%20bra%C3%A7o%20mec%C3%A2nico

[9] Beatriz Magalhães. (2024). *Demonstração do funcionamento do braço mecânico* [Vídeo]. YouTube. (https://youtu.be/GTRVjb-vFYU)