Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Mecatrónica

Relatório de Estágio

Avaliação e gestão de registos de soldadura por resistência.

Ivo Duarte Fabião

Orientador(es) | João Manuel Figueiredo
Rui Manuel da Silva Nunes

Évora 2023

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Mecatrónica

Relatório de Estágio

Avaliação e gestão de registos de soldadura por resistência.

Ivo Duarte Fabião

Orientador(es) | João Manuel Figueiredo
Rui Manuel da Silva Nunes

Évora 2023

O relatório de estágio foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	|	Fernando Manuel Janeiro (Universidade de Évora)
Vogais	|	Frederico José Grilo (Universidade de Évora) (Arguente) João Manuel Figueiredo (Universidade de Évora) (Orientador)

Évora 2023

Resumo

Avaliação e gestão de registos de soldadura por resistência

Este relatório apresenta um projeto de melhoria contínua na monitorização do processo de soldadura por resistência elétrica realizado na empresa *Kemet Eletronics*, em Évora.

O projeto foi desenvolvido em colaboração com o Departamento de Engenharia Mecatrónica da Universidade de Évora, com o objetivo de desenvolver a monitorização da soldadura, de modo a otimizar a qualidade, eficiência e rastreio do processo.

Para isso, foram realizadas análises detalhadas dos equipamentos utilizados na produção, identificando pontos de melhoria e necessidades de alterações para a monitorização adequada da soldadura. A metodologia utilizada no projeto envolveu etapas de análise, planeamento, implementação e avaliação das melhorias propostas.

Durante a fase de análise, foram coletados dados sobre o processo de soldadura, incluindo parâmetros de operação, tempos de ciclo e resultados de testes de qualidade. Com base nesses dados, foram identificados os principais desafios e oportunidades de melhoria. No planeamento, foram definidas as alterações necessárias nos equipamentos e no sistema de monitorização. Foram realizados estudos de viabilidade técnica e económica para garantir a eficácia das alterações propostas.

Em seguida, as alterações foram implementadas, incluindo a instalação de sensores adicionais, a atualização de *software* e a integração de sistemas de monitorização em tempo real.

Após a implementação, foram realizados testes e análises estatísticas para verificar os resultados obtidos com as alterações no processo de soldadura. Foram avaliados indicadores de qualidade, como a resistência das soldaduras, a uniformidade do processo e a taxa de defeitos.

O relatório apresenta os resultados alcançados, destacando os principais benefícios obtidos com as melhorias implementadas. Além disso, são apresentadas conclusões sobre o projeto, incluindo lições aprendidas e recomendações para trabalhos futuros.

Abstract

Evaluation and management of resistance welding records

This report presents a continuous improvement project in monitoring the electrical resistance welding process carried out at the company *Kemet Eletronics*, in Évora.

The project was developed in collaboration with the Department of Mechatronics Engineering at the University of Évora, with the aim of developing welding monitoring, in order to optimize the quality, efficiency and the level of tracking of the process.

For this, detailed analyzes of the equipment used in production were carried out, identifying points for improvement and need for changes for proper monitoring of welding. The methodology used in the project involved stages of analysis, planning, implementation and evaluation of the proposed improvements.

During the analysis phase, data on the welding process was collected, including operating parameters, cycle times and quality test results. Based on this data, the main challenges and opportunities for improvement were identified. In planning, the necessary changes in equipment and in the monitoring system were defined. Technical and economic feasibility studies were carried out to ensure the effectiveness of the proposed changes.

Then changes were implemented, including installing additional sensors, updating software and integrating real-time monitoring systems.

After implementation, tests and statistical analyzes were carried out to verify the results obtained with changes in the welding process. Quality indicators were evaluated, such as weld resistance, process uniformity and the rate of defects.

The report presents the results achieved, highlighting the main benefits obtained with the implemented improvements. In addition, conclusions about the project are presented, including lessons learned and recommendations for future work.

Agradecimentos

Dedico este capítulo para agradecer profundamente a todas as pessoas que contribuíram de forma significativa para a realização deste estágio e para o desenvolvimento do presente relatório.

O meu sincero agradecimento foca-se aos meus colegas de trabalho e de departamento, professores do Departamento de Mecatrónica da Universidade de Évora e, claro, à minha família.

Em primeiro lugar, agradeço aos meus colegas de trabalho, cuja colaboração foi essencial para o sucesso do meu estágio. O trabalho em equipa foi extremamente enriquecedor, permitindo-me aprender e crescer, tanto pessoal como profissionalmente. Agradeço a cada um pelo apoio constante, pelos debates construtivos e pela motivação mútua.

Quero, igualmente, expressar a minha gratidão aos meus colegas de departamento, cuja hospitalidade e acolhimento foram notáveis. Agradeço, essencialmente, a partilha de conhecimento e experiência, por orientarem-me ao longo do estágio e por criarem um ambiente de trabalho tão estimulante e amigável. As interações diárias foram fundamentais para o meu crescimento profissional e contribuíram, significativamente, para o meu percurso neste estágio.

Não posso deixar de mencionar ainda a minha profunda gratidão aos professores do Departamento de Mecatrónica da Universidade de Évora. A transmissão dos conhecimento com entusiasmo e dedicação, guiaram-me no desenvolvimento das competências necessárias para enfrentar os desafios encontrados neste estágio. Agradeço por incentivarem-me a explorar novas perspetivas e por orientarem-me em direção a uma aprendizagem mais profunda e abrangente.

Por último, mas certamente não menos importante, gostaria de expressar o meu profundo agradecimento à minha família. Aos que estiveram ao meu lado em todos os momentos, agradeço por todo o apoio, encorajamento e empatia que me facultaram. A vossa presença constante foi o alicerce que tornou possível enfrentar os desafios deste estágio e do meu percurso académico. Cada conquista alcançada é também fruto do vosso apoio e dedicação.

A todos os mencionados, o meu mais sincero obrigado. Cada um de vocês contribuiu de forma única para a minha formação e crescimento durante este estágio e ao longo de todo o meu caminho académico. Os vossos ensinamentos, orientações e palavras de incentivo foram indispensáveis e levarei comigo para sempre.

Muito obrigado a todos.

Lista de abreviaturas

CSV *Comma separated values*

CW1 *Crown Welding 1*

CW2 *Crown Welding 2*

CW7 *Crown Welding 7*

CW8 *Crown Welding 8*

CW9 *Crown Welding 9*

OS *Operating System*

Lista de Figuras

Conteúdo

Capítulo 1

Introdução

1.1 Contexto geral

O presente relatório, no âmbito do projeto de estágio realizado na empresa *Kemet Eletronics Corporation* (doravante *Kemet*), visa o desenvolvimento e análise de um programa de gestão de registos centrado na temática de soldadura por resistência, em prol da conclusão do mestrado em Engenharia Mecatrónica. Tendo em consideração um contexto introdutório, é condição basilar um diagnóstico e verificação iniciais sobre os aspetos a serem abordados.

Sabe-se que a soldadura por resistência é um processo estrutural com elevada importância em termos de adaptação e necessidade para diferentes indústrias e materiais.

Tal método pode ser definido pela "união de dois ou mais segmentos de metal, pela aplicação de calor e, por vezes, de pressão."(RWMA,2009:1) Esta união é produzida através de uma descarga de corrente elétrica através dos elétrodos, dispondo de diversas vantagens, tais como: prontidão no processo, alta precisão, bons acabamentos, automatização, entre outros.
[1]

Um dos exemplos da relevância da soldadura de resistência pode ser observado em processos da empresa *Kemet*. A última, fundada em em 1919, produz, essencialmente, condensadores eletrolítico com diferentes características . Apoiados na inovação tecnológica, os seus componentes são desenvolvidos para diferentes indústrias, possuindo os mesmos uma aplicabilidade diversa.

De facto, "contém mais de 1.600 patentes e marcas registadas em todo o mundo, estabelecendo uma posição de liderança para os seus produtos por meio de pesquisa e desenvolvimento avançados (...)."(*Kemet Corporation, 2022*)
[2]

Sendo a *Kemet* uma empresa com presença global, o seu pólo de Portugal configura-se em Évora, dispondo de mais de vinte anos de atuação, aumentando a influência e prestígio da região. Por tal,

[1] Tradução livre pelo autor. No original "(...) joining of two or more pieces of metal by the application of heat and sometimes of pressure. (...) RWMA, 2009:1

[2] Tradução livre pelo autor. No original:"Holding more than 1,600 patents and trademarks worldwide, we have established a leading position for our products via our advanced R&D(...)." Kemet Corporation, 2022

e atendendo que o projeto de estágio se desenvolveu nesse mesmo contexto, é necessário considerar que a parte experimental será inserida na conjuntura anterior, a ter em consideração.

1.2 Contexto específico

Em termos práticos, alguns dos principais clientes da *Kemet* Portugal são empresas como a *Bosch*, a *Sanmina* e o *Valeo*.

Portanto, os condensadores, pelas diferentes aplicações em que podem ser aplicados, como mencionado previamente, dispõe de diferentes especificações. Os condensadores do tipo coroa radial vão ser o principal foco deste relatório, visto envolve o processo que une a coroa ao condensador.

Diante disso, interessa explorar os elementos que antecedem a soldadura de resistência e o seu respetivo procedimento.

Podem ser referidos diversos processos entre os condensadores e a matéria-prima propriamente dita. Sendo assim, e numa primeira instância, encontra-se o ânodo, cátodo e papel dispostos em rolos de grandes dimensões que, posteriormente, são divididos ao longo da sua altura em rolos menores.

Estes são em seguida cortados em vários segmentos ao longo do seu comprimento. Os segmentos são sobrepostos de forma a criar uma camada isoladora, entre o ânodo e o cátodo, de papel. São em seguida enrolados em uma estrutura cilíndrica através do uso de um mandril.

Ao longo do enrolamento dos materiais é cravada a conecção ao ânodo e ao cátodo.

Neste processo, fatores como o tamanho dos segmentos, o alinhamento entre eles, a tensão do enrolamento, o numero de camadas isoladoras utilizadas e entre outros podem afetar o desempenho do condensador.

O processo seguinte seria a pré-construção, neste, a peça fica quase toda montada, ficando apenas em falta selar o condensador.

Neste, a peça é introduzida em uma lata de alumínio para as dimensões apropriadas do enrolamento cilíndrico obtido no processo anterior. De seguida é cravado o terminal do cátodo a lata de alumínio de forma a unir os materiais. É por este motivo que, neste tipo de condensadores, a carcaça do condensador fica associada ao terminal negativo. O terminal do ânodo é cravado á futura cobertura do condensador. Esta possui um anel de borracha para que quando o condensador for selado a carcaça e a tampa fiquem eletricamente isolados.

Os condensadores devem ser impregnados, isto é, a estrutura é mergulhada num banho de eletrólito sendo esta uma substancia liquida que contem iões. Esta substancia condutora serve para garantir o contacto de toda a superfície de oxido criada posteriormente através do processo de envelhecimento. Após a peça ser impregnada com o eletrólito, esta é fechada e fica selada para o exterior.

Em seguida, a peça segue para o envelhecimento, cujo objetivo se prende pela criação da de uma camada de óxido e, consequentemente, fazer surgir as propriedades elétricas do condensador, como capacidade e polaridade.

Nesta, são aplicados diferentes níveis de tensão e corrente controlada progressivamente de forma a permitir o desenvolvimento uniforme de uma camada de oxido.

Com o término da etapa anterior, as peças deverão possuir, idealmente, as características segundo as quais foram projetadas.

Para garantir essa vertente, ou seja, que as propriedades elétricas estão em conformidade, todas as peças são obrigatoriamente testadas para garantir o valor de capacidade, resistência em série, entre outros, relevando os diversos parâmetros que devem ser cumpridos. Caso se verifique o sucesso nos testes elétricos, isto é, o cumprimento de todas as especificações mecânicas e elétricas, os condensadores são marcados, estando prontos para serem condicionados e posterior processo de envio e adjudicação.

No entanto, alguns clientes têm especificações próprias que é necessário abranger antes do envio, como referido.

Assim, existem peças que depois do método de testagem têm de ser sujeitas a um outro processo complementar. No anterior, é soldada uma coroa ao condensado, relembrado que o terminal negativo fica a carcaça do condensador assim como a coroa, e o terminal positivo mantém-se vindo da cobertura. Agora, o condensador deve ser montado suportando-se na coroa.

Interessa referir que no procedimento anterior tem de existir uma garantia de força específica, num determinado intervalo, em prol da correta separação da coroa do condensador. Isto porque, a presença de força em demasia ou em menor quantidade provocará consequências. Tais como, a fácil separação da coroa do condensador ou (o sublinhado por outras palavras), respetivamente. Para além do constrangimento da força efetiva da soldadura, existem outros fatores quem tem de ser tidos em conta, como o centramento da coroa em relação ao resto do condensador.

Por conseguinte, é necessário existir rigor em todas as as fases descritas tendo em vista o mínimo desperdício. Dado que cada vez que o mesmo ocorre todo o processo elaborado deixa de servir o seu propósito.

Visto que estas, após este processo estão prontas para serem enviadas para o cliente, perdas neste processo significam um desperdiço dos processos anteriores. Cada peça perdida representa uma perda de lucro, sendo que caso não fosse a especificação pedida da coroa, a mesma estaria em condições ideais para venda.

O processo da soldadura da coroa ao condensador é realizado através de uma descarga elétrica processada entre a carcaça do condensador e a coroa, enquanto é realizada uma pressão específica da última sobre o condensador. O objetivo é que o ponto entre o condensador e a coroa, que como apresentam uma descontinuidade, sejam o conjunto de pontos onde a resistência elétrica do sistema é superior.

Assim, quando a descarga elétrica é feita, devido à relação P = I x I x R, é observado que nos pontos onde a coroa entra em contacto com o condensador dissipa-se potencia na forma de calor, através da formula anterior, principalmente na descontinuidade.

A potencia dissipada origina uma subida acentuada de temperatura nos pontos de contacto entre a coroa e o condensador. Assim, neste ponto, material tende a fundir-se, fixando a coroa graças à pressão exercida superiormente, ocorrendo a cedência da coroa e a união dos materiais.

Estas soldaduras são realizadas por um equipamento, fabricado pela *Amada*, o *ISQ-20*, que é um controlador de soldadura. Possui o controle e a potência necessária para realizar estas descargas de corrente, originando uma diferença de potencial entre os seus dois terminais de 3-5V. De destacar

ainda que possui uma capacidade energética enorme, realizando descargas na ordem das dezenas de milhares de amperes durante curtos intervalos de tempo.

Para controlar a corrente que debita, o controlador possui um anel, que tem que estar montado corretamente nos cabos de soldadura, de forma a medir a corrente para que se consiga auto-controlar.

Isto é, se for realizada uma soldadura no qual o anel está desconectado, o controlador irá debitar o máximo de corrente que consegue, assim o anel serve para limitar a corrente quando esta já está no nível pretendido.

Também é possível alterar o tempo de soldadura, ou seja, aumentar ou diminuir o intervalo de tempo durante o qual o controlador está a debitar corrente. Assim, através da fórmula do calor mais utilizada na soldadura por resistência elétrica - Heat = I x I x R x tempo de soldadura - percebemos que os fatores mais importantes para o calor, eletricamente, são a corrente, a resistência e a duração da soldadura. Mecanicamente, a pressão e o alinhamento são os principais contribuidores para o estado da soldadura.

Logo, sempre que é realizada um soldadura é importante caracterizá-la.

Para esse fim existe um equipamento, o *MG3* (igualmente fabricado pela Amada) que tem como função única a monitorização deste tipo de soldaduras. Recorrendo a um anel de medição de corrente similar ao controlador, de um terminal positivo e um negativo, de modo a medir a diferença de potencial entre o elétrodo negativo e positivo onde estes são montados. Em cada soldadura, o equipamento de monitorização irá registar qual foi a corrente de pico, corrente efetiva, tensão efetiva, tensão de pico, a resistência global do sistema (calculado pelo equipamento através da queda de tensão e da corrente medida, segundo a lei de ohm, V = R* I) a duração da soldadura, a potência, entre outros.

Deste modo, o equipamento descrito tem também a capacidade de rejeitar ou homologar soldaduras com base na avaliação de uma ou duas variáveis descritas anteriormente.

No entanto, o que se verificou é que não existia nenhum método eficiente para a correta monitorização do processo em questão, da parte específica de soldadura por resistência.

Logo, existindo essa lacuna foi desenvolvido um programa para gestão de registo de soldadura. Este servirá para criar um ponto de acesso aos registos produzidos pelo instrumento de monitorização de soldadura durante a sua atividade. A motivação é o aumento da rastreabilidade do processo subjacente através da correspondência destes registos com fatores relacionados com a produção e facilitando assim o posterior acesso aos dados.

Anteriormente, os registos produzidos pelo instrumento de monitorização da soldadura ficam guardados mas sem conceção real e tem que ser verificados manualmente quando são realizados estudos sobre estes. Isto acontece pois os registos de soldadura guardados pelo instrumento de monitorização apenas contem parâmetros relacionados com a soldadura realizada em especifico e nenhum outra informação que contextualize a soldadura no processo relativamente ao produto.

Assim, com o programa anterior será possível obter diferentes tipos de informações, a serem exploradas adiante, que permitirão não só o desenvolvimento de uma ação analítica e preventiva, como também uma melhoria geral da eficiência de todo o processo. Isto porque, por meio de uma monitorização eficaz e acessível será possível explorar dados e corrigir problemas, levando idealmente ao aumento do desempenho geral durante o processo de soldadura da coroa ao condensador.

1.3 Estrutura do relatório de estágio

O relatório de estágio está dividido em cinco capítulos:

1) Introdução: Nesta, é realizada uma contextualização do problema assim como uma introdução a alguns conceitos úteis para compreender objetivo do trabalho;

2) Estudo do problema: Abordagem em mais detalhe de cada um dos equipamentos em produção, das alterações necessárias a cada um de forma a fazer a monitorização da soldadura, entre outros.

3)Alterações de equipamentos: Neste capitulo são desenvolvidas as modificações que são realizadas a cada um dos equipamentos, sejam estas de natureza elétrica ou de automação.

4)Programa de gestão registos: Desenvolvimento do programa que realiza a gestão dos registos produzidos pelo instrumento de monitorização de soldadura. A totalidade do código do programa desenvolvido, esta disponível entre os anexos E e X e esta comentado de forma a facilitar a compreensão do mesmo. Na apresentação deste capitulo, são apresentados excertos de código quando é pertinente para explicar a lógica utilizada para a conceção do programa.

5) Controlo de processo: Capitulo relacionado com a integração de todos os sistemas projetados anteriormente e com os testes iniciais realizados.

Capítulo 2

Estudo do problema

2.1 Estado atual dos equipamentos

Na unidade de Évora da *Kemet Eletronics*, existem cinco equipamentos de soldadura de coroa o condensador, estes foram construídos em diferentes épocas e para soldar diversos tipos de coroa. Os equipamentos foram projetados com a finalidade de soldar a coroa superiormente, ou seja, o ponto de soldadura fica no topo do condensador, na respetiva aba. Existe apenas uma exceção ao anterior, sendo que a máquina de soldadura de coroa nº7 (ou 'CW7') foi concebida com vista a soldar coroas de apoio lateral.

Os equipamentos mencionados dispõe de diferentes níveis de implementação da monitorização da soldadura, sendo relevantes as seguintes considerações:

- Equipamentos como a máquina nº1, semi-automática, apenas possuem um controlador de soldadura, portanto não detém um equipamento de monitorização de soldadura nem tem capacidade de "rejeitar" peças;
- A máquina nº2 também não tem monitorização da soldadura, no entanto a automação proveniente do construtor da máquina abrange uma entrada digital para a rejeição de soldaduras, assim como um evento de erro;
- A máquina nº7, 8, 9, 10 e 11 são equipamentos mais recentes, que já incluem o instrumento de monitorização, mas não é possível fazer associação dos dados, por isso, os registos não se extraviam ficam perdidos, todavia ficam desconectados;

Assim, o projeto recai sobre, alterar os equipamentos nº1 e 2 de modo a tornar possível a rejeição de peças, recorrendo à monitorização da soldadura e implementar em todos os equipamentos uma forma de associar os registos à produção da máquina, bem como tentar estimar limites e variáveis de controlo do processo.

2.1.1 Máquina nº 1

A máquina nº1 apresenta-se como o equipamento que necessita de mais alterações tendo em consideração a possibilidade de monitorização e controlo da soldadura. Isto porque, como a anterior se

apresenta como semi-automática, sendo que praticamente não detém automação tratando-se apenas de uma sequência controlada por relés. Desse modo, com o botão iniciar desce uma proteção, através do relé temporizado, e é comutada a eletrovávulva que coloca pressão no cilindro superior com o elétrodo positivo. O último desde e dispõe um pressoestato para que quando a pressão definida seja atingida é dado um impulso para o controlador de soldadura com vista a realizar a descarga eletrica. Então, por fim, o relé temporizado volta ao seu estado normal e a respetiva proteção recua.

Neste equipamento é necessário implantar um pequeno projeto de automação que habilite ou desabilite o botão *start* e que controle as saídas digitais do equipamento de monitorização da soldadura.

2.1.2 Máquina nº 2

A máquina de soldadura de coroa nº2 está equipada com uma automação mais avançada e já detém automação. Sendo também semi-automática, já possui entradas digitais programadas para a possível falha por soldadura. Assim, utilizando as saídas digitais do equipamento de monitorização da soldadura, deve ser implementado o equipamento na máquina, tal como estruturar as ligações necessárias para que o sinal possa ser utilizado pela máquina.

2.1.3 Restantes equipamentos

Nas restantes máquinas nº7, nº9, nº10 e nº11 o equipamento de monitorização da soldadura já está implementado. Deste modo, os valores ficam guardados localmente no formato de registos proveniente do dispositivo de monitorização de soldadura em conjuntos de dados agrupados em 19999 registos contidos em um ficheiro com extensão *.log*.

2.2 Instrumento de monitorização de soldaduras.

O equipamento responsável por realizar a monitorização de cada soldadura é o *MG3D*, construído pela empresa *Amada Weld Tech*. Este utiliza um anel para realizar a medição da corrente no processo e um par de terminais para fazer a leitura da diferença de potencial entre os elétrodos negativo e positivo. O instrumento anterior é igualmente capaz de fazer o registo do deslocamento que ocorreu aquando da soldadura, ou seja, verificar em quantos micrometros o eletrodo se deslocou durante a soldadura. Este possui dois canais de medição independentes.

Neste instrumento é possível a definição de limites com base em uma ou duas variáveis de monitorização. Caso estes sejam utilizados quando é realizada uma soldadura, com base nos valores medidos, o equipamento faz um julgamento se a soldadura está dentro dos limites de controlo definidos. Caso esteja dentro dos limites, existe uma saída digital ("*Ok*") que é ativada, caso contrário, existe outra saída digital que fica conectada ("*Not good*").

Posteriormente, os valores medidos são guardados localmente no equipamento. Quando o número de registos é igual ao limite definido no *buffer*, o equipamento guarda os dados de todas as soldaduras presentes nesse *buffer* numa *pen drive* ou numa pasta na rede local. O ficheiro guardado tem então extensão *.log*, é um ficheiro do tipo *comma separated value* (ou traduzido para português "valores separados por virgula") no qual cada linha é referente a uma soldadura realizada.

Além do mais, o equipamento faculta a possibilidade de alterar de programa, permitindo assim guardar e alterar rapidamente os valores dos limites de controlo estabelecidos, o que pode ser útil para diferentes produtos produzidos

2.3 Conceito Global

Inicialmente é necessário introduzir nos equipamentos em falta o instrumento de monitorização. Para tal, será necessário executar diferentes alterações eletrónico nos equipamentos. Assim, todos os equipamentos possuem um instrumento de monitorização *MG3D*, ou existem disponíveis para integrar no mesmos. Deste modo é necessário padronizar a utilização do instrumento para todos os equipamentos, na monitorização assim como no registo de soldaduras para que sejam o mesmo tipo de registos para diferentes equipamentos.

Visto que é possível guardar os registos automaticamente é necessário introduzir os dados em uma base de dados para fazer a gestão destes registos e oferecer uma maior facilidade no acesso a estes registos de modo a melhorar a rastreabilidade do processo que decorre nos equipamentos.

Capítulo 3

Alterações de equipamentos

3.1 Crown welding 1

O Crown welding 1 é um equipamento que realiza soldadura de coroas ao condensador. Sendo um equipamento o semi-automatico, este implica que a coroa e o condensador são colocados manualmente nos elétrodos. As soldaduras são realizadas quando um nível pressão, definido no pressostato do cilindro utilizado no elétrodo positivo, é atingido entre a coroa e o elétrodo.

Este é o equipamento que necessita o maior número de alterações para integrar o instrumento de monitorização na sua automação. O equipamento, possui um *PLC* da marca *Omron* ("*CP1E*") e um HMI igualmente da *Omron* ("*NB5Q-TW01B*").

O PLC utilizado apenas possui 8 entradas e 8 saídas digitais. Na sua aplicação atual, este apenas é utilizado como contador com uma entrada e um saída e o modo de operação implica o operador inserir o número de peças a produzir no *HMI*, quando começar a operação, cada vez que for feita uma soldadura é dado um impulso na entrada "I0.0"do *PLC*, contabilizando assim que foi soldada uma peça.

Posteriormente quando a contagem das peças atinge o valor inserido inicialmente o *PLC*, a saída digital "Q0.0"é ativada e esta comuta um relé pelo qual tem que passar o impulso do botão de *start* do equipamento. Assim o inicio do ciclo é bloqueado, não sendo possível realizar soldaduras. Para realizar estas soldaduras o equipamento usa o controlador *ISQ-20*, tambem fabricado pela *Amada weld tech*

A segurança do equipamento, recorre a um rele de segurança para garantir que todas as portas estão fechadas e barreiras de segurança desimpedidas. Quando o botão de start é pressionada este rele tem que estar armado para que o ciclo de soldadura do equipamento seja iniciado.

Este equipamento, originalmente não possui o instrumento de monitorização de soldadura *MG3* e assim não é possível fazer a uma verificação as soldaduras realizadas. Visto que existem instrumentos do tipo para introduzir no equipamento, este pode assim ser integrado no equipamento.

No entanto, para tornar o seu uso idêntico ao de outros equipamentos é necessário proceder a uma serie de modificações ao equipamento pretendido.

A monitorização da soldadura deve ter a capacidade de intervir no processo de forma a alertar o operador do equipamento caso ocorram soldaduras consideradas irregulares.

Assim o conjunto de modificações a realizar devem abranger a implementação do instrumento de monitorização de soldadura no equipamento e a sua integração no ciclo de soldadura. Este deve ser capaz de parar o equipamento quando detetar uma soldadura que exceda um limite definido para um determinado parâmetro.

3.1.1 Modificações necessárias

As modificações a realizar ao equipamento de modo abranger o instrumento de monitorização de soldadura podem se dividir em duas partes.

Existem as modificações elétricas, aqui estão contempladas as alterações necessárias ao circuito elétrico do equipamento de forma a introduzir o instrumento de monitorização no equipamento e todas as novas ligações que estão associadas. Por outro lado, é fundamental alterar o programa contido no *PLC* de forma abranger estas ligações e a permitir a interface com o instrumento de monitorização.

O programa a alterar deve monitorizar as saídas do instrumento e atuar conforme seja a avaliação da soldadura. Caso esta seja considerada dentro do controlo do processo, (*"OK"*) pelo instrumento de soldadura, esta deve ser contabilizada e apresentada no *HMI*.

Se soldadura for considerada irregular, isto é , ultrapassa algum dos limites estabelecidos o *PLC* deve contabilizar a peça, neste caso como peça não boa. O *PLC* nesta situação só deve permitir a soldadura seguinte quando o condensador for introduzido em uma caixa das peças que não devem seguir no processo.

Para garantir que o condensador foi introduzido na caixa, é colocado um sensor optico á entrada para que este de um sinal ao *PLC*. O operador tem também de confirmar que introduziu o condensador na caixa, para isso, deve ser integrado um botão que deve ser pressionado quando o condensador foi introduzido na caixa.

Este botão, a quando da soldadura irregular, fica iluminado com uma luz encarnada para que seja visível que o equipamento parou devido á soldadura.

No início da produção o *PLC* deve limpar o contador do instrumento de monitorização da soldadura e quando é terminada a produção, o ficheiro que contém os registos de soldadura devem ser guardados. Estas ações são feitas através da interface digital do instrumento de monitorização, recorrendo ao *PLC*.

A interface digital do equipamento, recai sobre o conjunto de modificações elétricas necessárias. Para realizar a comunicação entre o *PLC* e o instrumento de monitorização é utilizado um conjunto de saídas e entrada digitais relacionadas com o controlo do mesmo.

Isto implica a construção dos cabos e fichas que possibilita a utilização a estas entradas e saídas do equipamento. Em seguida, é necessário realizar estas ligações ao *PLC*.

As alterações efetuadas no programa têm como objetivo monitorizar as saídas do instrumento e atuar conforme seja a avaliação da soldadura, existindo dois cenários:

a) A soldadura é considerada correta e deve ser contabilizada;

b) A soldadura ultrapassa algum dos limites estabelecidos e não pode ser aceite. Nesse caso o PLC deve contabilizar a peça chumbada, mas somente permitir a soldadura seguinte quando a recusada for introduzida na respetiva caixa de peças chumbadas, acionando o sensor adequado, e depois da confirmação manual da introdução da peça através de um botão.

No início da produção o *PLC* deve limpar o contador do instrumento de medição e quando é terminada a produção o ficheiro que contém os registos de soldadura deve ser guardado.

Estas ações são feitas através da interface digital entre o *PLC* e o equipamento de medição, atraves das ligações presentes nas figuras 3.3 e 3.2

3.1.2 Modificações eletrónicas

De modo à adaptação do equipamento as modificações realizadas na sua automação, nomeadamente o uso de mais entradas e saídas digitais e para isso é necessário realizar um serie de alterações eletrónicas ao equipamento.

A integração do instrumento de monitorização vai recorrer as interfaces *X54*, *X55* e *X57* deste. Assim para o implementar no equipamento é preciso fazer as ligações de interface através das fichas *X54*, *X55* e *X57* assim como as ligações de medição da soldadura.

Assim os componentes utilizados para realizar as medições são o anel de medição de corrente que deve ser atravessado pelo cabo de soldadura proveniente do terminal positivo do controlador de soldadura. Os terminais de medição de diferença de potencial são instalados entre o elétrodo positivo e negativo do equipamento.

Tendo então em consideração o exposto anteriormente, nos próximos pontos serão consideradas as modificações eletrónicas necessárias para integrar o instrumento de monitorização na automação do equipamento.

Ficha X54 - Saídas digitais *MG3D*

A construção das fichas *X54*,*X55* e *X57* deve seguir as ligações disponibilizadas pelo fornecedor, presente no anexo A. Para a construção da ficha *X54* é necessário uma ficha D-Sub de 25 contactos (também conhecida por *DB-25*).As conexões feitas devem obedecer assim ao esquema de ligações presente na figura 3.1.

Na última afigura-se a necessidade de fornecer 24 *Volts* externos ao instrumento de monitorização para que o mesmo possa comutar esse sinal. Assim, no pino 9 e 22 devem ser introduzidos os 24V geral do equipamento.

O pino 25, correspondente ao "comum", deve ser ligado no 0V do equipamento para garantir que as referências de tensão se encontram ao mesmo potencial.

Por outro lado, as saídas a utilizar que se mostram relevantes para utilização e que é necessário obrigatoriamente de ligações estabelecidas são: o sinal que o equipamento de monitorização está pronto para monitorizar soldaduras no pino 1, o *Ok* para os canais 1 e 2 (pino 14 e 20) e o *not good* para os canais 1 e 2 (presentes no pino 15 e 7).

Feitas as ligações anteriores existe a possibilidade de acesso as saídas anteriores provenientes do equipamento de monitorização da soldadura.

Ficha X55 - Entradas digitais *MG3D*

A ficha X55 está relacionada com o controlo do equipamento de monitorização, mas neste caso é a ficha das entradas digitais. A mesma, assim como a X54 é uma ficha D-Sub de 25 contactos, apenas difere da última devido a corresponder à versão fêmea do tipo de conector.

Similarmente ao trabalho feito para a a anterior e recorrendo à figura 3.2 que corresponde a parte do diagrama das ligações disponibilizado pelo fornecedor.

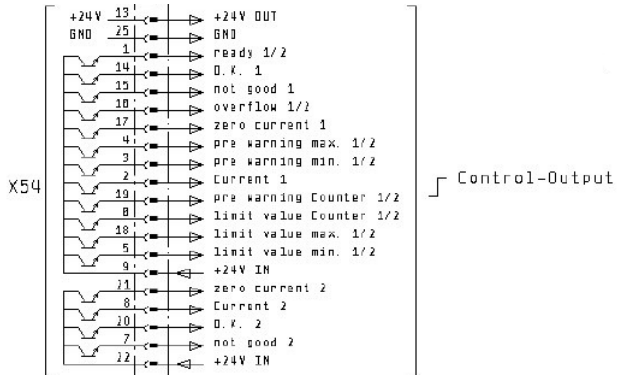


Figura 3.1: Saídas digitais instrumento de monitorização de soldadura

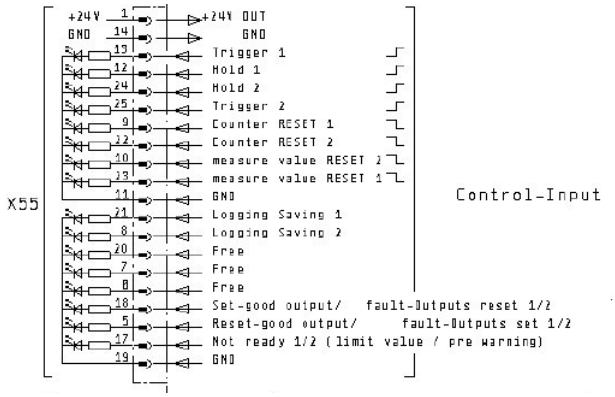


Figura 3.2: Entradas digitais instrumento de monitorização de soldadura

É assim necessário fornecer o comum (0 Volts) da tensão que vai ser comutada pelo *PLC*. Para tal é imperativo ligar ao comum os pinos 11 e 19. Os sinais que carecem de ser empregues, como referido anteriormente, são os sinais necessário para limpar os contadores internos do equipamento, ou seja o pino 9 para limpar o contador do canal 1 e o pino 22 para limpar o contador do canal 2.

Além disso, é fundamental a utilização do pino 21 e 8 para que o equipamento de monitorização seja capaz de gravar os dados na localização pretendida através de um impulso externo

Ficha X57 - Alteração do programa do instrumento de monitorização

A alteração do programa procede-se de modo similar ao controlo externo do equipamento, a ficha neste caso é do tipo D-Sub de 15 contactos macho e assim a ficha a construir deve ser fêmea.

É, novamente, necessário fornecer o comum do sinal que vai ser usado, sendo que este deve ser ligado no pino 5 como é descrito na figura 3.3.

Visto que, nesta aplicação, no máximo apenas são usados quatros *bits* para enumerar o programa, são feitas as ligações referentes aos seguintes: bit 1 no pino 1, bit 2 no pino 9, bit 4 no pino 2 e o

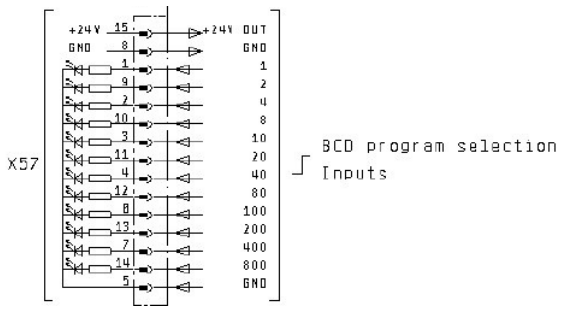


Figura 3.3: Entradas digitais alteração programa instrumento de monitorização de soldadura

bit 8 no pino 10 da ficha correspondente.

Diferenciação de canais de medição

Como foi anteriormente referido, o equipamento referente à monitorização de soldadura requer a realização de medições aquando da soldadura.

A medição da corrente é realizada por meio de um anel que deve ser colocado em um local arbritario do cabo de que liga o controlador de soldadura ao elétrodo positivo.

Este deve estar orientado conforme o sentido indicado pelo fabricante, sendo relevante que no presente será considerado a situação de conter um símbolo "+"(a indicar o respetivo lado que deve estar a apontar para o lado do controlador de soldadura).

O cabo liga a ao anel deve ser ligado na entrada *IK1*, caso a medição seja para ser feita no canal 1 ou *IK2* se a medição for efetuada no canal 2, do instrumento de monitorização de soldadura.

Por sua vez, a tensão é medida através de um cabo com dois terminais, sendo que o positivo deve ser ligado no elétrodo positivo e o negativo no elétrodo negativo.

De forma similar à medição da corrente, este cabo deve ser ligado na ficha *UK1* caso a medição seja para ser feita no canal 1 ou *UK2* caso a medição seja para ser feita no canal 2 do instrumento de monitorização de soldadura

A aplicação em causa não implica o uso de dois canais de medição e no caso em específico o autômato não possui o número de entradas necessárias para suportar toda essa interface.

No entanto, já se mostrou útil ser capaz de utilizar ambos os canais de medição para perceber se existe algum problema nas medições de um determinado canal do instrumento de monitorização.

Como na situação analisada ao longo do presente relatório os elementos tratam-se apenas de sinais digitais, pode ser utilizado um sistema em que ou estão selecionadas a saídas relativamente ao canal 1 ou relativamente ao canal 2.

Visto o sinal "*ready*"(X54.1), segundo a figura 3.1, indica se o equipamento está preparado para realizar uma medição. Este é comum a ambos os canais e por isso não precisa de estar abrangido.

Apenas é necessário os sinais de "*OK*"e "*Not Good*"para cada canal provenientes do instrumento de monitorização assim como os sinais dos sinais de limpeza do contador interno do instrumento de monitorização da soldadura e de guardar os registos.

Assim, pode ser utilizado um relé com 4 contactos em que os sinais do instrumento de monitorização de soldadura são ligados, um canal aos contactos normalmente abertos e outros aos normalmente fechados.

Deste modo, se o relé for armado, a interface com o instrumento de monitorização passa a ser feita para um dos canais e quando for desarmado para o outro.

Atendendo ao anterior, compreende-se a utilização de um pequeno interruptor para fazer armar/desarmar este rele. O relé utilizado para o efeito foi construído pela *Omron* sendo o modelo *MY4N*.

Assim as modificações elétricas a realizar, descritas anteriormente, são inseridas na forma de um esquema elétrico apresentado no anexo B.

Neste, são descritas as ligações elétricas entre o *PLC* e instrumento de monitorização de soldadura. É também exemplificado o uso do rele descrito para fazer a alteração do canal de medição cujas saídas são diferentes. Estas atendem a considerações feitas posteriormente neste relatório, relativamente as entradas e saídas utilizadas do *PLC*.

3.1.3 Automação

Sendo o objetivo a introdução do instrumento de monitorização no equipamento anteriormente mencionado, será imperativo monitorizar as suas saídas.

Assim de modo a fazer a interface com o instrumento de monitorização são usadas as fichas *X55*, *X54* e *X57*. A descrição das ligações á interface do instrumento de monitorização de soldadura são descritas no anexo A. Através deste, são consideradas relevantes para aplicação projetada, o conjunto de entradas:

- "*Ready*"1/2, X54.1, ficha X54 no pino 1 é referente ao sinal a indicar que o instrumento de monitorização esta pronto para efectuar uma medição, seja no primeiro ou segundo canal.
- "Ok 1", X54.14, ficha X54 no pino 14 indica que a ultima medição realizada pelo canal 1 está dentro dos limites definidos;
- "*Not Good 1*- X54.15" , na ficha X54 no pino 15 o sinal indica que a ultima medição realizada no canal 1 esta fora de os limites estipulados;

O "Ok 2 - X54.20" e "*Not Good* 2 - X54.7"são idênticos aos sinais para o canal 1, mas neste caso para o canal 2. Diferenciando assim, o pino atribuído ao correspondente sinal.

É também fundamental tirar partido das entradas digitais do instrumento de monitorização de soldadura como "Counter reset 1 - X55.9" ," Counter reset 2 - X55.22" ,"Logging Saving 1 - X55.21" e "Logging Saving 2 - X55.8".

A alteração do número do programa é feita através da ficha X57 sendo do tipo D-sub de 15 pinos e utilizando codificação binária.

Como apenas existem quatro saídas disponíveis no PLC, somente é possível fazer combinações com 4 bits, ou seja, 16 combinações diferentes.

Visto que o equipamento CW1 não é capaz de produzir muitas tipologias de produtos diferentes, 16 programas é mais do que suficiente para a aplicação.

Estas quatro saídas ,estão desfasadas dois bits nas saídas do *PLC*, visível no anexo B. Isto deve-se ao facto existem variantes deste PLC que não possuem as saídas digitais, Q0.6 e Q0.7.

Tendo esse fator em consideração, em caso de avaria é possível utilizar um desses modelos que existe amplamente no centro de operações de Évora, necessitando de uma alteração mínima no programa de automação.

Com o conhecimento da interface do equipamento e seguindo o plano anteriormente descrito do funcionamento pretendido do equipamento, procede-se à construção de uma tabela de entradas e saídas a serem utilizadas pelo programa, descrita assim na tabela 3.1.

Definidas as variáveis do programa, prosseguimos com o desenvolvimento do *grafcet* funcional do equipamento cumprindo os seguintes requisitos:

- Possibilidade de alteração do nº do programa;
- Limpar o contador do instrumento de monitorização ou seja do MG3;
- Guardar registos manualmente através da interface HMI;
- Paragem do equipamento quando peça excede os limites definidos no instrumento de monitorização;
- Sistema de contagem de peças.

Entrada	Descrição	Saidas	Descrição
I 0.0	EV Pulse	Q 0.0	EV Cut
I 0.1	Accept MG3	Q 0.1	Bit 0
I 0.2	Reject MG3	Q 0.2	Bit 1
I 0.3		Q 0.3	Save log
I 0.4	Botao confirm.	Q 0.4	Luz Nok
I 0.5	Sensor Nok	Q 0.5	Clear Counter
I 0.6	Ready MG3	Q 0.6	Bit 3
I 0.7		Q 0.7	Bit 4

Tabela 3.1: Entradas e saidas digitais Crown Welding 1

- Caixa de rejeição com sensor e botão de confirmação de introdução na caixa;
- Criação de avisos de alerta ao utilizador do equipamento caso o anterior não execute determinada ação num período de tempo estipulado;

O *grafcet* funcional desenvolvido, é apresentado na figura 3.5. Exposto o desenvolvimento do *grafcet* com os seus respetivos requisitos pode-se proceder a programação do autómato.

Através do o programa *Cx-Programmer*, utilizado para a programação dos autómatos da *Omron* da serie utilizada. O programa assim desenvolvido, esta incluído no anexo C, abrange as modificações propostas.

Neste é aplicada a lógica proposta pelo *grafcet* da figura 3.5 utilizando ás entradas e saídas presentes na tabela 3.1.

A interface gráfica, que utiliza um *HMI* também da marca *Omron*, cujo modelo utilizado é o *NB5Q-TW01B*.

De forma a configurar e programar a anterior é utilizado a ferramenta *NB-Designer*.

Neste foram introduzidos mais tres janelas ao programa, construída uma nova janela com a finalidade de mostrar ao operador qual foi o veredicto da soldadura, decidido pelo instrumento de monitorização. Outra, com o objetivo de auxiliar a resolução de problemas por parte dos serviços técnicos indicando quais são as entradas e saídas atuais do *PLC*, assim como o estado, do *grafcet* da figura 3.5, em que o programa se encontra.

Deste modo, os principais objetivos da interface humana são:

- Representação gráfica do resultado da soldadura, isto é, a sinalização virtual encarnada caso a soldadura seja considerada fora dos limites e verde caso a soldadura esteja dentro dos limites;
- Apresentação da contagem de peças boas / peças más;
- Introdução de comandos manuais de limpeza do contador e de gravação de registos do instrumento de monitorização;
- Apresentação e registo de erros;
- Possibilidade de um modo que apresente o número atual de cada peça aquando da sua rejeição em prol da sua separação e registo do seu número de série para tornar possível o rastrear de parâmetros monitorizados durante a soldadura.
- Verificação das entradas e saídas lógicas do *PLC*;
- Opção de desativar funcionalidades.

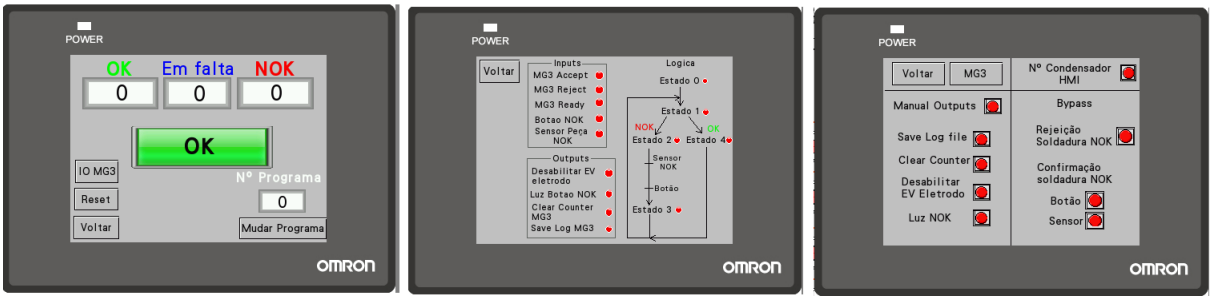


Figura 3.4: Paginas de interface humana - Crown welding 1

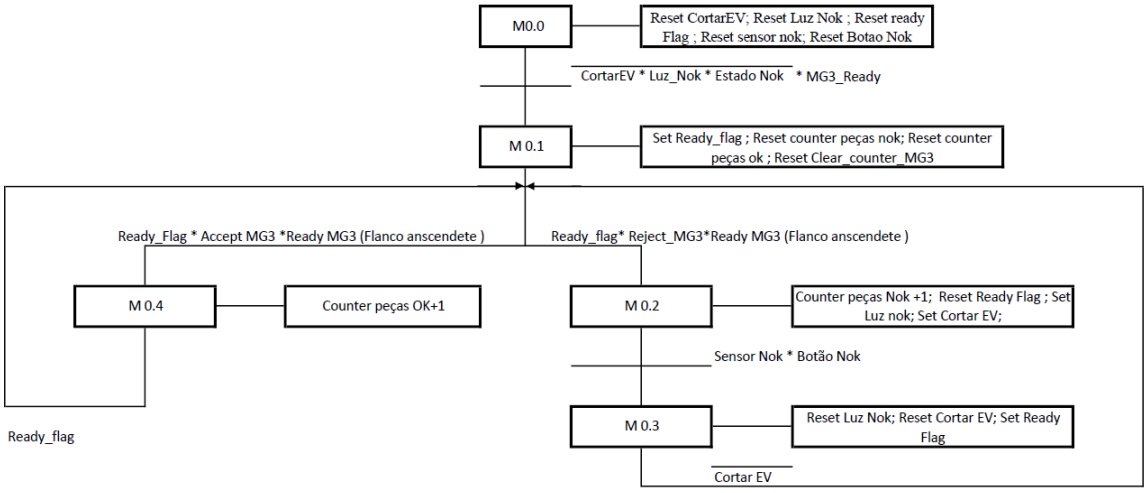


Figura 3.5: *Grafcet* Funcional proposto para alterações equipamento CW1

3.2 Crown welding 2

O equipamento Crown welding 2 é um equipamento também semi-automatico como a nº1, mas que possui um automato para implementar o ciclo da mesma . Este equipamento, serve para realizar soldadura da coroa ao condensador sendo estas através de um controlador de soldadura do tipo *ISQ20*.

O equipamento, já possui entradas lógicas prontas para receber um sinal a indicar um soldadura fora dos limites. Isto acontece pois o próprio controlador de soldadura possui a possibilidade de definir limites internos para ele proprio, podendo deste modo validar uma soldadura. No entanto, o objetivo é integrar um instrumento independente para fazer a monitorização da soldadura e consequentemente do proprio controlador.

Outras funcionalidades como a mudança de programa do controlador de soldadura também ja estão incluídos na automação do equipamento. Como tanto o controlador como o instrumento de monitorização são fabricados pela mesma empresa, estes possuem o mesmo tipo de comunicação, sendo possível assim copiar o sinal que seria para alterar o numero do programa do controlador de soldadura pode tambem servir para mudar o numero do programa do instrumento de monitorização.

Existem outras saídas digitais implementadas no autómato que podem ser utilizadas para outras operações, como a interface do instrumento de monitorização. O equipamento tem a capacidade de limpar o contador interno do controlador de soldadura, atraves de um sinal digital. Utilizando este sinal é possível limpar também o contador interno do intrumento de monitorização assim como ordenar que sejam gravados os ficheiros de registos de soldadura.

Assim, essencialmente o equipamento crown welding 2 já possui todas as entradas e saídas necessárias para implementar o instrumento de monitorização de soldadura e a possibilidade de avaliação de soldaduras com base no mesmo.

3.2.1 Modificações eletrónicas

De forma a integrar o instrumento de monitorização no equipamento de soldadura de coroa numero 2 é necessário proceder em primeiro lugar á construção das ligações que fazem a interface entre o autómato e este.

As ligações de entradas e saídas do instrumento de monitorização são desenvolvidas de forma similar ao realizado para a "crown welding 1". É assim necessário, construir as fichas que fazem interface, novamente seguindo as orientações do fornecedor como é descrito no anexo A.

Assim, as fichas que é necessário construir são novamente a *X54* que corresponde as saídas digitais do instrumento de monitorização, *X55* as entradas digitais de controlo e *X57* as entradas de alteração do numero do programa.

No caso da ficha *X54*, com base na figura 3.1,são feitas as ligações para os sinais:

- "Ready 1/2", sinal que indica que instrumento de monitorização de soldadura esta pronto para realizar medições, no pino 1.
- "OK 1", homologação da ultima soldadura realizada no canal um, através do pino 14.
- "Not good 1", indicando que a ultima soldadura realizada no canal um ultrapassou um dos limites estipulados, através do pino 15.

Para alem das ligações anteriores é ainda necessário fornecer +24V do equipamento para o instrumento de monitorização, para as saídas utilizadas deve ser fornecido no pino 9.

As saídas digitais necessárias do instrumento de monitorização ficam assim acessíveis através da ficha *X54*.

As ligações necessárias para a ficha *X55*, que correspondem as entradas de controlo do instrumento de monitorização, presente na figura 3.2.

- "Counter RESET 1", sinal que força o instrumento de monitorização de soldaduras a limpar o contador interno do canal 1, no pino 9.
- "Logging Saving 1", sinal que obriga instrumento de monitorização de soldaduras a guardar, no destino selecionado, o ficheiro de registos referentes ao canal um, no pino 21

É também necessário fornecer 0V (comum) do equipamento para o instrumento de monitorização, esta referencia é necessária para seja possível comutar as entradas utilizadas, nos pinos 19 e 11.

Por fim é construida a ficha responsável pela alteração do numero do programa do instrumento de monitorização de soldadura. O construtor do equipamento atribuiu oito saídas do *PLC* para fazer a alteração do numero do programa do controlador de soldadura. Visto que tanto o controlador como o instrumento de monitorização são fabricados pela mesma empresa, utilizam o mesmo metodo para fazer a alteração do numero do programa. Assim sendo, é possível ir buscar estes sinais e utilizar para fazer também a alteração do numero do programa do instrumento de monitorização.

Com base na figura 3.3, é necessário ligar os oitos bits utilizados pelo equipamento.

Assim, é construida a ficha *X57*, utilizando as entradas:

- Referente ao numero 1 no pino 1;
- Referente ao numero 2 no pino 9,
- Referente ao numero 4 no pino 2;
- Referente ao numero 8 no pino 10;

- Referente ao numero 10 no pino 3;
- Referente ao numero 20 no pino 11;
- Referente ao numero 40 no pino 4
- Referente ao numero 80 no pino 12.

Para alem destas ligações é novamente necessário fornecer o comum do equipamento (0 Volts) no pino 5 para que seja possível comutar estes sinais.

Assim as alterações elétricas realizadas visam essencialmente a construção dos conectores assim como a integração no equipamento.

Para introduzir estas ligações ao equipamento, foi criado um esquema sendo este uma modificação do esquema original do fornecedor do equipamento.

Assim, o esquema das alterações elétricas realizadas no equipamento crown welding 2 esta disponível no anexo D.

Neste estão presentes as ligações que devem ser feitas, após serem construidas as fichas *X55*; *X54* e *X57*, no equipamento de modo a permitir a interface com o instrumento de monitorização de soldadura.

Capítulo 4

Programa de gestão de registos

4.1 Objetivo do programa

De modo a gerir todos os registos provenientes dos instrumentos de monitorização de soldadura presentes nas diferentes linhas de produção, surge a necessidade de criar um programa que efetue a gestão dos dados, bem como correspondência dos mesmos com detalhes específicos da produção em cada equipamento. O programa a ser criado deve assim oferecer uma solução de rastreabilidade e gestão dos registos provenientes da monitorização do processo de soldadura por resistência elétrica.

À vista do anterior, a interface do programa deve permitir, de forma acessível, a introdução de dados relativos à produção para que os mesmos sejam complementados aos registos da monitorização do processo para aumentar as capacidades do programa. Os registos, por sua vez, necessitam de ser tratados e armazenados em uma base de dados para que estejam prontamente disponíveis no futuro, de forma a serem desenvolvidos pelo programa ou extraídos para utilização em plataformas externas.

Em relação à extração de dados da base de dados, a mesma deve oferecer a capacidade de filtrar dados segundo múltiplos parâmetros introduzidos pelo utilizador. Ademais, o programa deve igualmente produzir relatórios desenvolvidos sobre os dados provenientes do instrumento de monitorização de soldadura armazenados na respetiva base, com vista à sua consequente análise.

Por outro lado, é imperativo o programa disponibilizar funcionalidades de gestão do mesmo como, por exemplo, proceder à alteração da localização dos ficheiros dos registos relativos aos diferentes instrumentos de monitorização em utilização, assim como verificação do acesso a essa localização e adicionar novos equipamentos ao programa de controlo de registos de monitorização de soldadura para abranger diferentes necessidades futuras da produção.

O programa tem assim de cumprir os seguintes requisitos:

- Associar dados do instrumento de medição a dados da produção como equipamento, número de lote, descrição, entre outros;
- Disponibilizar os dados armazenados conforme seja solicitado;
- Gerir base de dados dos registos;

- Interface de introdução e tratamento de dados.

4.2 Linguagem de programação e bibliotecas

O programa desenvolvido para fazer o tratamento dos registos de soldadura foi programado na linguagem de programação *Python 3*. A linguagem escolhida teve em conta a facilidade de implementação da mesma, com uma sintaxe simples e um amplo repositório de bibliotecas de livre acesso que amplificam as capacidades da linguagem realizar diferentes tipos de operações com implementações acessíveis, diminuindo substancialmente o tempo necessário para desenvolver um programa com as funcionalidades pretendidas.

Tais funcionalidades vão, na sua maioria, utilizar variadas bibliotecas para implementar os algoritmos do programa. De destacar que as bibliotecas escolhidas para o desenvolvimento do programa de gestão de registos de soldadura devem ser de livre acesso. Para além disso, o programa requer uma interface gráfica em prol da sua acessibilidade e fácil compreensão a qualquer utilizador. Portanto, a biblioteca utilizada para esse propósito é a biblioteca *[8]*, cuja atuação possibilita a interface entre o compilador do *Python 3* e o conjunto de ferramentas Tk que tem como propósito o desenvolvimento de interfaces gráficas, sendo o mesmo de livre acesso. Assim, através desta biblioteca, estão ao dispor um conjunto de ferramentas disponíveis para a criação e modificação dinâmica de objetos em uma interface gráfica no programa em desenvolvimento.

Desse modo, durante os vários processos realizados pelo programa de gestão de registos, guardados pelo instrumento de monitorização, os algoritmos vão necessitar de aceder a diferentes localizações no dispositivo em que o programa esta ser executado, assim como na rede local. Tal é indispensável para realizar diferentes tipos de operações, nomeadamente abrir os ficheiros provenientes do instrumento de monitorização da soldadura ou criação de ficheiros. Para realizar essas operações recorre-se ao módulo *OS* do *Python 3* que oferece ferramentas de operações no sistema operativo. Os ficheiros que o programa vai necessitar são os documentos guardados pelo instrumento de monitorização de soldadura. Contudo, devido ao modo como este instrumento foi programado, este guarda os ficheiros em diferentes localizações e necessitamos aceder e iterar sobre as mesmas.

Assim, estes ficheiros são guardados no formato *.log*, ou seja, um ficheiro de texto com a estrutura de um ficheiro do tipo *.Comma separated values* (CSV). O último é um ficheiro em que a informação é guardada na forma de texto, existindo um delimitador que indica quando a informação terminou. Interessa referir que este tipo de ficheiros tem capacidade de armazenar diferentes valores. Por isso, os ficheiros *.log* são, essencialmente, ficheiros do tipo *CSV* estando assim formatados de modo similar. Para a sua leitura ou escrita de ficheiros a ferramenta a utilizar é o módulo *CSV* do *Python 3*.

Porém, os registos provenientes do instrumento de monitorização necessitam de algumas considerações com vista a uniformizar os dados em prol do seu formato ser coerente e legível. Para tal, é necessário, por exemplo, proceder à alteração do formato da data de cada registo, pois o programa depende da data e hora de cada registo para realizar a correspondência dos dados da produção com os dados do instrumento de monitorização da soldadura. Nesse sentido, para proceder a operações com dados no formato de data, recorremos à biblioteca *datetime* que é uma biblioteca nativa do *Python 3* que serve o propósito.

Posteriormente, após os ficheiros serem abertos e interpretados corretamente, é necessário armazenar os dados de forma acessível. Com esse propósito, é essencial implementar uma base de

dados para que a informação seja guardada de forma estruturada, consistente e que esteja sempre disponível para respetiva consulta. À vista disso, é utilizado o módulo *sqlite3*, sendo o seu objetivo realizar a interface com a biblioteca escrita para a linguagem de programação C denominada *SQ-lite*. A última oferece muitas das funcionalidades de uma base de dados SQL ("*Structured Query Language*" ou Linguagem de consulta estruturada), mas que não necessita de um processo separado para um servidor. Com esta biblioteca podemos efetivar ações sobre a base de dados de modo a criar, alterar, eliminar ou consultar elementos da mesma através de consultas estruturadas que vão ser construídas dinamicamente pelo programa e executadas sobre a base dados.

Os dados obtidos a partir destas consultas serão desenvolvidos de modo a estarem disponíveis para o utilizador final, seja através de gráficos e considerações tomadas pelo programa utilizando ferramentas matemáticas como a biblioteca Numpy que possui muitas funcionalidades que podem ser utilizadas para fazer operações em conjuntos de dados. Além disso, é igualmente significativo produzir uma representação gráfica dos dados disponíveis de modo a obter uma outra perspetiva sobre os resultados obtidos com vista à sua análise. Para construir diferentes tipos de gráficos de forma dinâmica vai ser utilizada a biblioteca *Matplotlib* que dispõe de diversas ferramentas relacionadas com o propósito anterior e que deve ser utilizada na criação de todos os gráficos que o programa produzir.

Estes podem ser exportados ou utilizados para a criação de relatórios referentes aos registos do instrumento de monitorização da soldadura, por resistência, em um lote da produção. A construção dos relatórios será essencial para uma melhor compreensão de todos os elementos e é elaborada recorrendo à biblioteca *PyFPDF*. A última, sendo uma biblioteca de criação de ficheiros PDF, FPDF é uma biblioteca originalmente escrita na linguagem de programação PHP, porém é portado para o *Python 3* mediante da biblioteca central utilizada no programa, já mencionada anteriormente. Desta forma será possível utilizar os gráficos e cálculos feitos no programa para criar um ficheiro no formato padrão PDF com os resultados numéricos e gráficos obtidos.

4.3 Desenvolvimento do programa

4.3.1 Interface gráfica

Todo e qualquer programa, para se apresentar como acessível ao maior número de utilizadores, deve apresentar uma interface gráfica. A última, permite então interagir com o primeiro de forma intuitiva. Logo, no ponto 4.3.9 será explicada a interface gráfica do programa desenvolvido e expresso no presente relatório.

Assim, o objetivo é criar um conjunto de páginas para o programa de forma a facilitar o acesso as funcionalidades que o programa possui. Através desta interface será então possível utilizar as funções do programa.

Em termos de implementação, a linguagem de programação *Python 3* possui intrinsecamente uma biblioteca de programação de interfaces gráficas. A biblioteca em questão é denominada *Tkinter* e com esta é possível criar janelas e objetos com os quais é possível interagir. Esta tem como vantagem a sua simplicidade de implementação assim como a quantidade de recursos que disponibiliza.

Os objetos e janelas que forem construídos ao longo do programa vão utilizar esta biblioteca para alcançar, dentro do possível, os resultados pretendidos. As janelas necessárias são a janela principal, que apenas pode existir uma, e as janelas secundarias que podem-ser criadas e destruídas conforme necessário. Assim a interface global do programa vai estar situada na janela principal, neste caso, se a janela for fechada a execução do programa é interrompida.

As páginas relativas às principais funções do programa como introdução de dados, atualização de dados, extração de dados e criação de registos são concebidas por meio da utilização dos recursos associados às janelas secundárias.

Assim, os objetos que vão ser mais utilizados são etiquetas, caixas de introdução de texto e botões, para tal, a biblioteca dispõe de métodos para os anteriores. Os objetos introduzidos na pagina, necessitam ser criados e posicionados. A criação do objecto dependo do mesmo, mas para o posicionamento o método que é utilizado, em todo o programa, para posicionar os objetos é o *grid*. Este utiliza coordenadas relativas para posicionar os objetos na pagina.

Dentro desta perspetiva, para criar a janela principal e todos os seus constituintes procedeu-se ao desenvolvimento de uma função denominada *main_f*, estando a mesma disponível para consulta no anexo E. A função mencionada é então responsável por definir a página principal do programa, os objetos contidos nesta e o seu posicionamento.

Em primeiro lugar, a janela principal é criada e este objeto é guardado em uma variável, denominada *root*. São definidos os parâmetros para esta janela, como o titulo e o tamanho da mesma recorrendo a métodos da biblioteca utilizada. De seguida, são definidos e posicionados, na página principal, os cinco objetos utilizados sendo estes quatro botões e uma etiqueta com uma imagem.

Os botões são criados utilizando o método *Button*. Ademais, para a janela principal se tornar funcional necessita de pelo menos três parâmetros, o objeto da janela, o texto e um comando para função que deve ser executada quando o botão for pressionado.

Pode-se, assim, observar a implementação do botão que inicia a janela, de atualização de base de dados, através da figura 4.1, que é um excerto de código contido, na função *main_f*. Desse modo,

```
2854    #Criar e posicional botao de entrar no pagina para introduzir dados de lote
2855    b_data_enter = Button(root, text="Introduzir Dados \n Lote ", command=data_enter_click)
2856    b_data_enter.grid(row=2,column=4,padx=15,pady=10)
```

Figura 4.1: Implementação de botão que abre janela de introdução de dados da produção

o objeto é criado recorrendo ao método referido, utilizando como parâmetros a página principal, o texto que deve estar no botão ("Introduzir dados de lote", no caso) e a ação que deve tomada quando este é pressionado.

Por conseguinte, a função a executar é definida quando o botão for pressionado, utilizando o parâmetro *command* do método, sendo que neste caso esta definido para a função *data_enter_click*.

Desse modo, a função anterior procede à criação uma página secundária do programa com o objetivo de possibilitar ao utilizador a atualização dos registos da base de dados. Esta é abordada no ponto 4.3.4.

Em seguida é posicionado na janela utilizando o método *grid* sobre o objeto. Como parâmetros são utilizados a linha e coluna onde deve ficar posicionado e o espaçamento que deve ter para o seus objetos vizinhos.

Os restantes botões presentes na página são configurados relativamente a: pagina de atualização de registos de soldadura na base de dados, à exportação de ficheiros da base de dados e da constituição de relatórios. As opções disponíveis na pagina principal do programa são visíveis na figura 4.3. Ou seja, a implementação de todas as páginas mencionadas processa-se de forma idêntica à descrita no parágrafo anterior, variando a função que cada uma executa quando são pressionados.

Outro tipo de objeto que vai ser amplamente utilizado pelo programa são etiquetas. Estas são responsáveis por apresentar texto ou até imagens na janela. À vista disso, existem diversos modos de apresentar as imagens, por meio da utilização da biblioteca *Tkinter*, mas para funcionalidade desejada este método é aplicável.

Com tais características, a apresentação de texto e imagens na janela recorre ao método *Label* da anterior. Então, a definição de etiquetas procede-se de forma similar à utilizada para a criação do botão. Contudo, o método apresenta-se como distinto, não se atribuindo uma função a executar, visto que não é possível, por parte do utilizador, interagir com o objeto.

Por outro lado, e devido a questões estéticas, é posicionada uma imagem com o logótipo da empresa na janela principal do programa. A implementação desta imagem é demonstrada na figura 4.2, mas serve como exemplo para a criação de qualquer objeto do tipo *Label*, utilizado extensivamente durante o programa.

Diferenciando assim o uso , na figura, do argumento *imagem* ao invés de *text* Em prol da apresentação de imagens, através do método anterior, é imperativo a prévia importação da imagem para o programa. Com esse intuito, recorre-se à biblioteca *Pillow* para criar um objeto *ImageTk* com a informação contida no ficheiro com a imagem. Tal objeto é agora compatível com o método do tipo *Label*. Utilizando o argumento *image*, referenciando a variável que contem a informação da imagem existente no ficheiro, é possível criar o objeto com a imagem. Posteriormente, segue-se o posicionamento do objeto na janela, como acontece para todos os objetos criados.

Em seguida, existe a necessidade de implementar funcionalidades na barra de ferramentas. Tendo esse aspeto em consideração, o próprio programa dispõe de algumas funções de visualização

```
2845    #abrir ficheiro com imagem da empresa e interpretar imagem com biblioteca PIl (pillow)
2846    img = ImageTk.PhotoImage(Image.open(open("resources\Kemet.png", 'rb')))
2847
2848    #criar objecto de etiqueta mas com a imagem da empresa e posicionamento no ecra
2849    label_main_program = Label(root, image = img)
2850    label_main_program.grid(row = 0, column=2,pady=20,columnspan=3)
```

Figura 4.2: Implementação de imagem na janela principal

de dados e de gestão. Mesmo não constituindo as principais funcionalidades do programa, estas devem estar acessíveis. Assim a barra de ferramentas e é local apropriado para este tipo de funções do programa.

Para atingir esse fim é utilizado o método *Menu* sobre a variável *root*, que contem a janela principal, sendo assim criado o objeto que contem a barra de ferramentas. Sobre este são definidas opções de ações possíveis, cuja declaração sucede-se de forma similar à utilizada na construção do botão. Neste caso, estas ações são para abrir janelas, como a janela de introdução de novos equipamentos no programa, ou a verificação do acesso aos destinos onde estão guardados os ficheiros de registos de soldadura.

Por fim, é necessário executar o ciclo para página principal para a mesma entrar em funcionamento. O ciclo é iniciado através do método *mainloop*, no objecto da janela principal *root* no caso, que é responsável por manter a janela do programa aberta. Com tal ação, o ciclo será capaz de aguardar pelas diversas interações na interface, de forma a ter potencial de resposta a essas mesmas interações. Assim, quando o ciclo é iniciado, a janela é executada, passando a estar operacional, contendo os elementos anteriormente descritos.

Através da execução da função *main_f*, obtém-se a interface visível na figura 4.3. A anterior interface caracteriza-se então por ser a página principal do programa.

Por último, interessa ainda destacar que no decorrer do desenvolvimento programa será imprescindível a criação de diferentes tipos de páginas secundárias seus respetivos objetos utilizados. Para tal, são sempre utilizados os métodos explicados ao longo do presente ponto, existindo a possibilidade de proceder a pequenas alterações conforme a necessidade da página. Sendo assim, mesmo na criação das posteriores janelas secundárias a implementação dos objetos é similar.

Figura 4.3: Janela principal do programa

4.3.2 Base de dados

A aplicação é na sua essência um gestor de uma base de dados, sendo que as principais operações que este desempenha estão relacionadas com interações com a base de dados e a gestão dos dados a adicionar.

A base de dados utilizada recorre, como foi referido previamente, à biblioteca do *python 3* [7] denominada *Sqlite3* [2]. Esta apresenta, praticamente, todas as funcionalidades de uma base de dados SQL com maior capacidade. Sendo assim, e na aplicação descrita no presente relatório, a base de dados apenas vai servir como um ponto comum onde os registos da monitorização da soldadura são armazenados para que possam ser consultados e alterados mantendo a sua acessibilidade.

Assim, a base de dados em *sqlite3* é apropriada para o programa que desenvolvido, dado que tem uma implementação bastante acessível e oferece consultas estruturadas. Caso no futuro exista necessidade de aumentar as capacidades da base de dados existem outras alternativas, como *MySql* ou *Oracle* que não necessitam de alterações substanciais ao código, mas aumentam as capacidades da base de dados em aplicações que tenham essa necessidade. No entanto, todas as funcionalidades necessárias estão disponíveis na biblioteca mais leve a ser utilizada.

Em termos práticos, a base de dados é constituída por múltiplas tabelas, uma para cada equipamento em produção, uma apenas para os dados relativos à produção real em cada equipamento e uma tabela para armazenar os valores de força de tração de testes destrutivos realizados periodicamente durante a produção. As tabelas que armazenam os dados reais relativos da produção e valores relativos aos testes destrutivos, são tabelas auxiliares ás referentes aos registos do instrumento de monitorização de soldadura para cada equipamento em produção. Isto porque, é a partir destas que é atualizada a informação da produção nas tabelas dos registos de cada equipamento.

Deste modo, para cada equipamento em produção, a base de dados possui uma tabela que contém entradas para cada registo de soldadura gravado pelo instrumento de monitorização. Bem como colunas destinadas aos dados relativos à produção que são introduzidos pelo operador do equipamento, antes do início da produção. Por conseguinte, para cada equipamento em produção que possui um instrumento de monitorização de soldadura é criada uma tabela na base de dados com as seguintes colunas:

1. Número de lote - entrada do tipo texto que serve para armazenar o número de lote;
2. Descrição - entrada do tipo texto para armazenar notas ou informações da produção;
3. *Part Number* - entrada do tipo texto que guarda o tipo de produto a ser produzido no equipamento;
4. *Item* coroa - coluna que guarda o item interno da coroa para uma certa entrada (texto);
5. Lote Coroa - coluna que guarda o número de lote das coroas a serem utilizadas na produção (texto);
6. *Setpoint I* - coluna que armazena o valor de corrente definido no controlador de soldadura (texto);
7. Referência elétrodo - coluna para guardar registo da referência do elétrodo utilizado na produção(texto);
8. Data - coluna que guarda a data o registo (texto);

9. Hora- coluna que guarda a hora do registo (texto);
10. Contador - coluna que guarda o valor do contador interno do instrumento de monitorização para o registo - (inteiro);
11. Programa - coluna que guarda o número do programa selecionado para cada registo (texto);
12. Ipk - coluna que guarda o valor da corrente de pico (em [kA]) registado pelo instrumento de monitorização (texto);
13. Upk - coluna referente ao valor da tensão de pico (em [V]) registado pelo instrumento de monitorização (texto);
14. Potência - coluna que armazena o valor da potência do processo estimada pelo instrumento de monitorização de soldadura, em [kW] (texto);
15. s1 - Deslocamento, em [um], do elétrodo durante a soldadura para o canal 1.
16. s3 - Deslocamento, em [um], do elétrodo durante a soldadura para o canal 2.
17. Força, estimativa da força exercida pelos elétrodos, em [N].
18. Pressão, pressão medida atraves de sensor no eletrodo, em [Pa]
19. Corrente RMS - coluna que guarda o valor da efetivo da corrente (em [kA]) registado pelo instrumento de monitorização, calculado segundo a raiz quadrada média (texto);
20. Q - coluna que armazena o valor do calor resultante do processo estimado pelo instrumento de monitorização de soldadura, em [As] (texto);
21. Tensão RMS - coluna que guarda o valor médio efetivo da tensão (em [V]) registado pelo instrumento de monitorização, calculado segundo a raiz quadrada média (texto);
22. Ut - coluna que guarda o valor da tensão por segundo do processo. Estimado pelo instrumento de monitorização é a unidade é [Vs];
23. Ws - Potencia por segundo da soldadura, medido em [Ws]
24. tw - coluna que guarda a duração do processo segundo registado pelo instrumento de monitorização, em [milissegundos]
25. STATUS - coluna que guarda a informação do instrumento de monitorização, por exemplo caso estejam limites definidos nesta coluna vai registar se a uma soldadura está dentro dos limites de controlo do processo.
26. LogId Esta coluna é responsável por identificar cada entrada na base de dados. É preenchida automaticamente quando adicionamos um novo registo a uma tabela que tenha esta coluna de modo a que posteriormente seja mais acessível e célere a consulta de entradas na base de dados.

Assim, cada medição realizada pelo instrumento de monitorização contem toda a informação relativa aos parâmetros de soldadura. Nas tabelas auxiliares, vão ser guardados registos relativos ao numero de lote, item da coroa, data de inicio, data de fim de um lote, entre outros. Ou seja parâmetros da produção.

Estes vãop ser introduzidos pelo operador, de modo a que seja possível associar registos de soldadura a parâmetros da produção. A tabela referente aos dados da produção tem a identificação de *updatedata*, cada entrada nesta tabela contém informação referente à produção num equipamento como: número de lote, a descrição, o tipo de peça, o valor de corrente definido para o processo, o item e lote da coroa utilizado e por fim a referência do elétrodo, como foi referido anteriormente.

Para além destas informações, dispõe a hora de início e término de determinada produção, assim como a identificação do equipamento da produção em si. Outros algoritmos terão como funcionalidade verificar estes registos de produção e averiguar se possuem registos provenientes do instrumento de monitorização que se adequem ao intervalo de tempo em análise.

Já a última tabela de apoio armazena os valores obtidos nos testes de tração realizados destrutiva mente sobre as peças em produção.

Estes testes existem para que seja feita uma amostragem das peças de forma a garantir que a especificação de força necessária para separar a coroa do condensador e avaliando assim o comportamento mecânico esperado da soldadura. As peças sujeitas ao teste, que verifica o esforço axial necessário para separar a coroa do condensador, são destruídas visto que não são aproveitadas peças que já passaram por este processo.

No entando, caso exista conhecimento do valor do contador interno do instrumento de soldadura, para uma determinada peça, podemos, mais tarde, associar à tabela que contém os registos de soldadura para cada equipamento, o valor de força de tração obtido no teste correspondendo assim os valores registados pelo instrumento de monitorização com a força com que a coroa foi unida ao condensador.

A tabela em que os dados dos testes destrutivos são guardado é denominada *pullforcedata*. Esta, possui apenas quatro colunas sendo elas referentes ao equipamento que soldou a peça que vai ser testada, o número de lote dessa mesma peça, o valor do contador interno do instrumento de monitorização e o valor de força medido. Posteriormente, o programa irá confirmar se possui algum registo que coincida com alguma entrada nesta tabela. Caso se verifique tal situação irá consagrar-se uma atualização da coluna da descrição para o valor de força medido e introduzido.

A interface com a base de dados acontece recorrendo à biblioteca *sqlite3*, como foi anteriormente referido. As operações de criação de uma base de dados, consultar entradas, introduzir dados, entre outros, acontecem sempre do mesmo modo.

Em primeiro lugar, é necessário criar um objeto que seja a conexão à base de dados. Para tal é utilizada função *connect* da biblioteca *sqlite* em que o seu retorno é o objeto da ligação à base de dados.

Após ser estabelecida a ligação à base de dados é necessário criar um cursor, este é uma estrutura muito utilizada no controle de base de dados, pois serve como uma memória temporária de onde é possível serem elaboradas operações na base de dados como, por exemplo, adicionar ou consultar entradas.

Esta estrutura é criada utilizando o método *cursor* no objeto da ligação à base de dados. Posteriormente a esta estrutura estar criada já é possível interagir com a base de dados, assim para interagir com uma base de dados SQL tem de ser utilizadas consultas estruturadas.

Apesar de estar a ser utilizada uma biblioteca leve de SQL existe a necessidade de construir estas consultas estruturadas de modo a interagir com a base de dados. As consultas seguem os princípios de construção idênticos ao de linguagens *SQL*. Após a consulta estar construída pode ser executada sobre a base de dados, a biblioteca em utilização simplifica o processo apenas sendo necessário passar a consulta ou uma variável que contenha a consulta através do método execute no objeto do cursor passando a consulta como argumento.

Caso tenha sido feita uma alteração à base de dados devemos comprometer-nos utilizando o método *commit* no objeto da conexão à base de dados se pretendermos confirmar as alterações. Se tiver sido feita uma consulta podemos aceder os dados obtidos utilizando o método *fetchall* no cursor da base de dados. Assim, todas as novas entradas , alterações e consultas de dados na base de dados são implementadas de modo similar ao exemplo apresentado anteriormente sofrendo pequenas alterações conforme a aplicação.

4.3.3 Introdução de registos de soldadura na base de dados

O instrumento de monitorização de soldadura guarda os registos dos parametros monitorizados durante a mesma na sua memória interna. Quando estes registos atingem um número definido, ou quando controlado externamente, os mesmos são guardados na forma de um ficheiro *.log*. Deste modo, o instrumento cria um conjunto de caminhos onde pode armazenar diferentes parâmetros, configurações e registos. Contudo, o objetivo principal será aceder à diretoria onde estes ficheiros de registos estão guardados.

Por isso, o programa para encontrar todos os ficheiros relacionados com registos de soldadura deve ser indicado para o caminho */logging*, no conjunto de pastas criadas pelo instrumento de monitorização.

No caminho pretendido, a organização dos ficheiros utilizada pelo instrumento de monitorização tem ser levada em consideração. Assim sendo, este procede à criação de uma pasta referente a cada dia em que é guardado um conjunto de registos. O formato do nome da pasta é *yy-mm-dd*, cujo *yy* representa os últimos dois algarismos do ano, *mm* o mês do ano, sendo antecedido de zero se for inferior a dez, e *dd* o dia da semana.

Cada uma destas pastas criadas pode conter um ou mais ficheiros *.log* que comportam os registos de soldadura produzidos pelo instrumento de monitorização. Os ficheiros no formato *.log*, cujo nome do ficheiro se trata de um número incremental e a diferenciação do canal de medição a que os registos são referentes. Assim, a denominação que os ficheiros de registo possuem encontra-se no formato *1_xxxx.log* e *2_xxxx.log*, consoante o canal de medição do registo. Por sua vez, *xxxx* é o número incremental que vai aumentando com o número de ficheiros de registos na diretoria */logging* , independentemente da quantidade de pastas.

De modo a abordar a estrutura utilizada, o programa precisa de iterar sobre as pastas referentes aos dias, de modo a alcançar todos os ficheiros que contém registos de soldadura. Para tal, podemos considerar o seguinte algoritmo com o propósito de encontrar as localizações no sistema dos registos de soldadura guardados pelo instrumento de monitorização:

1. Alterar localização de trabalho para a diretoria que queremos verificar;
2. Obter os caminhos para cada pasta com registos;
3. Guardar o caminho para cada ficheiro de registos na diretoria atual;
4. Repetir 2 e 3 até a lista obtida em 1 estar terminada;
5. Retornar lista com os caminhos para todos os ficheiros de registos possíveis de avaliar.

Consequentemente, o ciclo utilizado pelo programa para executar esta tarefa é demonstrado na figura 4.4. À vista disso, recorrendo à biblioteca *os* é possível alcançar todas estas pastas, originando assim um ciclo que recorre ao método *listdir* para que quando seja inserido um parâmetro com um caminho, retorne uma lista com todos os ficheiros e respetivos acessos presentes na diretoria. Visto que, como foi referido anteriormente, em primeiro lugar é necessário avaliar um conjunto de pastas relativas a cada dia. É assim necessário em seguida, alcançar cada ficheiro em cada uma dessas

```
873 #Criar lista para guardar caminhos para ficheiros de registos
874 list_logs_path=[]
875
876 #ciclo que itera as pesquisas dos conteudos da pasta e os guarda na lista list_logs_path
877 for folder in os.listdir(path):
878
879     for file_name in os.listdir(path+ "\\" + str(folder)):
880         list_logs_path.append(path+ "\\" + str(folder)+ "\\" + str(file_name))
881
882 #inverter lista obtida para que ultima entrada passe para a primeira
883 list_logs_path.reverse()
```

Figura 4.4: Ciclo de obtenção de caminho para ficheiros de registo

pastas. Para isso podemos iterar o método sobre a lista de pastas obtida resultando, assim, uma lista com o caminho para todos os ficheiros localizados.

Após o procedimento anterior, é possível aceder à lista que contém os caminhos para todos os ficheiros de registo de um certo equipamento. Logo, o conteúdo de cada um desses ficheiros deve ser verificado por meio da analise de cada um destes. Os registos encontrados, podem necessitar de ser adicionados á base de dados ou podem já estar contida na mesma.

Diante disso, é necessário proceder a essa distinção entre registos. O programa deve abrir o ficheiro e verificar as datas presentes no mesmo e, em seguida, ser construída uma consulta para ser elaborada á base de dados sobre todos os dados relativos ao intervalo de tempo a que correspondem as entradas presentes no ficheiro de registos.

Isto é, em primeiro lugar, é necessário abrir o ficheiro de registos e importar o seu conteúdo. Para tal, foi então concebida uma função cuja finalidade prende-se com determinado caminho para um ficheiro de registos - o programa deve abrir o ficheiro, verificar e formatar o ficheiro de forma coerente. É para esse fim, implementada a função denominada *get_log_file*, que pode ser consultada no anexo F. Esta recorre à biblioteca *CSV* para abrir um de determinado ficheiro no formato *.CSV*.

A função aceita como parâmetro um caminho para um ficheiro de registo de soldadura. Com o caminho utiliza o método *reader* da biblioteca CSV para extrair os conteúdos do ficheiro. Em seguida, avalia todos os registos quanto à sua validade e procede à formatação das entradas da data e do contador interno dos registos do instrumento de monitorização, de modo a facilitar a posterior consulta na base de dados. Os registos válidos e no formato correto são inseridos em uma lista que, quando o programa termina de avaliar o ficheiro de registos, corresponde ao retorno da função.

Obtida a lista de registos, é necessário proceder a uma comparação com os registos na base de dados de modo a determinar que referências adicionar. Para isso, é criado um ciclo, tendo a sua aplicação como é demonstrada na figura 4.5, com a finalidade de obter uma lista com todas as datas presentes na enumeração de registos presentes no ficheiro.

Assim, é necessário captar todos os registos na base de dados relativos a um determinado equipamento no período de datas obtido. Para desempenhar essa tarefa foi construída uma função, *download_db_data*, disponível no anexo G, que tem como propósito construir e executar consultas à base de dados para um determinado equipamento e um conjunto de datas.

Esta função vai criar, para cada data presente na lista, uma consulta à base de dados de modo a obter os registos ja contidos na anterior para cada uma dessas datas. À posteriori, todos os registos

```
#Criar variavel para guardar lista de datas presentes no ficheiro log
dates =[]

#Obter registos no ficheiro de registos
logs = get_log_file(log_file)
#prints de debug que informa quantos registos existem num determinado ficheiro de registos
print("tamanho do ficheiro de logs: ", len(logs) , " | logs no ficheiro: ",log_file)

#ciclo que verifica datas presentes num ficheiro de registos
for log in logs:
    #caso não tenha a data na lista deve adicionar
    if log[0] not in dates:
        dates.append(log[0])
```

Figura 4.5: Ciclo de obtenção de datas presentes no ficheiros de registo

obtidos pela função são compilados numa lista pelo programa que a retorna quando terminar todas as consultas que tem de realizar. Esta é executada em seguida na nossa função de atualização da base de dados, como demonstrado na figura, 4.6.

Caso o retorno da função seja uma lista do mesmo tamanho, não existe necessidade de inserir os dados na base de dados, pois essas entradas já estão presentes na mesma. Já se a consulta retornar uma lista vazia, podem ser então adicionadas todas as entradas do ficheiros de registos à base de dados, dado que a última não contém entradas relativas a esse intervalo de datas.

Caso se verifique a necessidade de adicionar registos à base de dados é executada uma função, *insert_db*, demonstrada no anexo H, que tem como intuito a inserção de um conjunto de dados na tabela da máquina que lhe for referida. Esta aceita como argumento o equipamento cuja tabela é para alterar na base de dados e as entradas a introduzir na mesma.

A função origina a ligação à base de dados e cria um cursor sobre essa mesma ligação. Em seguida, verifica se os registos provenientes do ficheiro de registos estão todos com a mesma dimensão (número de colunas).

A lista com os registos é convertida para uma variável do tipo tuplo (tipo de variável similar à variável lista mas imutável), em prol da utilização do método *insertmany* da biblioteca *sqlite*.

Este método itera automaticamente sobre o tuplo criado, inserindo na base de dados cada elemento presente no mesmo, sem necessidade de iterar no programa cada uma das novas entradas

Não obstante, existe igualmente a possibilidade do número de resultados da consulta ser superior a zero, mas diferente do número de entradas na base de dados referentes a essas datas. Nesse caso é necessário avaliar ambos os conjuntos de dados para diferenciar as entradas que já estão na base de dados das que necessitam de ser adicionadas.

Para fazer essa comparação utilizamos um pequeno ciclo, como pode ser observado na figura 4.7, que apenas é executado na condição descrita. O programa cria, desse modo, uma lista com a data e hora de cada registo, já contido na base de dados. Com esta lista o programa verifica se existem registos com data ou hora diferentes dos registos já contidos na base de dados.

Se existir algum registo de soldadura, no ficheiro em análise, que cuja data e hora não esteja na lista dos registos da base de dados, então este é adicionado a um conjunto para ser acrescentado, visto que não foi encontrado na mesma. Caso seja feita toda a verificação e, todavia, não seja encontrado um único registo em falta na base de dados, a variável de limite é incrementada.

Adicionalmente, existe necessidade de limitar o número de ficheiros analisados para que o programa não percorra sempre todos os ficheiros de registos quando fizer uma verificação. Porém,

```
#Ciclo que verifica as datas obtidas
if len(dates)>0  :

    #executar função que cria consulta á tabela de um determinada maquina para o intervalo de datas
    db_data =download_db_data(machine,dates)

    #Prints de debug inidicando ficheiro em analise registos encontrados
    print("Datas a verificar na base de dados: ",dates, " equipamento", machine)
    print("Tamanho da BD : ",len(db_data)," |Tamanho do ficheiro com LOGS : ",len(logs),
          " | Referente ás datas ",dates, "para a maquina" ,machine)

    #Caso o tamanho da lista de registos na base de dados seja igual do ficheiro de registos
    if len(db_data) == len(logs):

        #incrementar variavel limite
        strike=strike +1

        #Print de debug indicando que os registos ja estão na base de dados
        print("Option - Todos os ",len(logs)," logs ja estavem presentes na base de dados ")

    #Caso não exista nenhum registo na base de dados para as datas analise
    elif len(db_data)==0:
        #reset á variavel de limit
        strike=0
        #registos encontrados
        n_items_adicionados=n_items_adicionados+len(logs)
        #executar função que adiciona registos á base de dados
        insert_db(machine,logs)

```

Figura 4.6: Ciclo caso tamanho de registos na base de dado igual a zero ou a número de entradas no ficheiro de registos

configura-se importante a possibilidade de ser realizada uma verificação completa a todos os ficheiros quando for pretendido.

Para esse propósito é utilizada uma variável de limite para controlar o numero de ficheiros de registos de soldadura analisados, durante a atualização, em que não foram encontrados registos em falta na base de dados.

É assim definido o "*strike limit*" representando o número limite de ficheiros seguidos que o programa pode encontrar sem adicionar nenhum registo à base de dados. Para cada ficheiro avaliado, em que os registos já constem na base de dados, esta variável é incrementada em uma unidade. No caso desta chegar ao valor limite, o ciclo presente na figura 4.6 e 4.7 é interrompido e a verificação de ficheiros termina.

A introdução de entradas na base de dados consiste assim num algoritmo capaz de conceber a procura dos ficheiros de registos existentes, a respetiva interpretação dos ficheiros, a análise dos registos, a comparação dos registos e ,por fim, a introdução dos registos em falta na base de dados.

Para facilitar a operação do programa, foi criada uma janela na interface gráfica com o objetivo de disponibilizar as ferramentas de atualização da base de dados ao utilizador. Assim, é permitida a atualização de um equipamento em específico ou a alteração da variável limite da atualização da base de dados.

Utilizando então a biblioteca *Tkinter*[8] foi introduzido um botão no programa principal, cuja denominação se prende por "Atualizar Base de dados". O mesmo, quando pressionado, procede à abertura de uma janela secundária, idêntica à figura 4.8. A janela é aberta, pois o botão quando é pressionado executa a função *b_update_db_click*, demonstrada do anexo I. Esta tem como objetivo criar e gerir a página de atualização da base de dados.

Quando esta função é executada, em primeiro lugar é criada uma janela e, em seguida, o programa executa uma verificação dos equipamentos que dispõe como configurados no programa, de forma a definir caixas de seleção para todos.

```
964 #Caso numero de registos na base de dados > 0 mas diferente do numero de registos no ficheiro
965 elif len(db_data) != len(logs) and len(db_data)>0:
966
967     data_update_db=[]
968     dates_db=[]
969
970     #Criar lista com data e hora de cada registo
971     for data in db_data:
972         dates_db.append((data[7],data[8]))
973
974     #Print de debug
975     print("Tamanho da lista de logs comparar ", len(dates_db)," para a maquina ",machine)
976
977     for item in logs:
978
979         if (item[0],item[1]) in dates_db:
980             #Remover datas que forem encontradas para diminuir lista dinamicamente
981             dates_db.remove((item[0],item[1]))
982
983         else:
984             #adicionar á lista a adicionar á base de dados
985             data_update_db.append(item)
986
987     if len(data_update_db)==0:
988         #Incrementar variavel de limite caso não existam registos para adicionar á base de daos
989         strike=strike +1
990
991     else:
992         #limpar variavel limite
993         strike = 0
994         #executar a função para adicionar registos á base de dados com os registos em falta encontrados
995         insert_db(machine,data_update_db)
```

Figura 4.7: Sequência de comparação de entradas da base de dados com registos no ficheiro

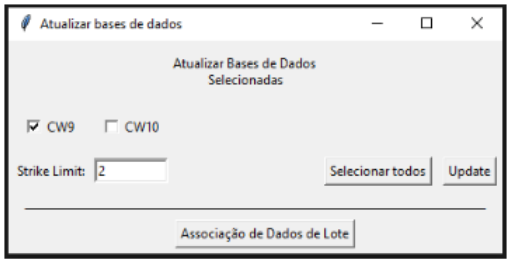


Figura 4.8: Janela de atualização da base de dados e associação de dados

Posteriormente, cria os outros objetos relacionados com a atualização da base de dados, como um botão para selecionar todos os equipamentos, denominado por "Selecionar todos". O botão de *update* inicia uma função que verifica quais equipamentos que se encontram como selecionados e produz uma *thread*, responsável por executar a função *verify_logs*, descrita anteriormente e disponível no anexo J.

Para atualizar os registos na base de dados relativos a cada equipamento é necessário pressionar o botão "*Update*", este executa a rotina descrita para cada equipamento selecionado.

A interface gráfica permite também visualizar o desenvolvimento da atualização dos registos na base de dados, através de uma mensagem e barra de progresso (presente na página principal do programa). A última vai evoluindo à medida que o programa atua conforme a avaliação dos ficheiros da lista de ficheiros de registos, a verificação da quantidade de registos até determinado momento adicionados à base de dados, assim como a que equipamento são os registos referentes.

O botão, "Atualizar base de Dados", que abre a janela com atualização da base de dados fica

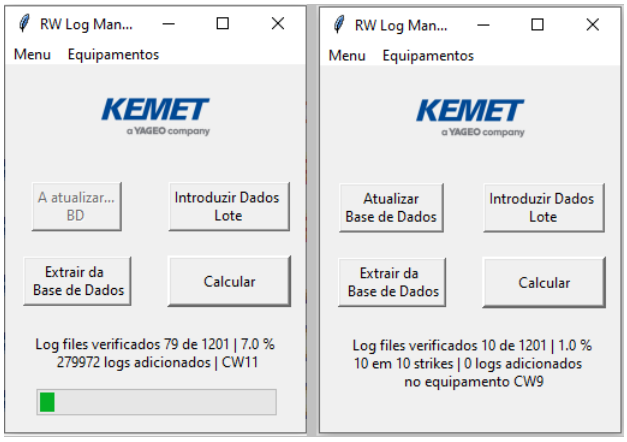


Figura 4.9: Exemplo de comportamento interface gráfica durante atualização de registos e interrupção por limite definido.

desabilitado enquanto atualização decorre, como está exemplificado na figura 4.9. De modo a bloquear o acesso a esta pagina enquanto programa executa todas as tarefas de procura e analise de ficheiros de registos de soldadura.

4.3.4 Introdução dados relativos à produção

A base de dados, após serem introduzidos registos de soldadura, possui varias entradas correspondentes a soldaduras de coroas realizadas. No entanto, estas apenas possuem parâmetros inerentes á soldadura mas ainda nada que as identifique. É necessário introduzir nestas entradas parâmetros da produção como o numero de lote e outros detalhes para que seja possível relacionar ambos estes tipos de dados.

De modo a corresponder os registos do equipamento de soldadura com dados relativos à produção em cada equipamento, é necessário introduzir no programa os dados relativos à produção. A maioria dos parâmetros que se pretende adicionar aos registos de soldadura não são conhecidos pelo equipamento no qual o processo é realizados e têm assim que, obrigatoriamente, ser introduzidos de forma manual no programa de gestão de registos de soldadura.

A data, hora e equipamento de um registo são únicos, visto que não é possível, nos equipamentos utilizados, acontecerem duas soldaduras em simultâneo. Por esse motivo, tais parâmetros são sempre distintos e podem ser utilizados para averiguar se um registo diz respeito a diferentes lotes e tipologia de produtos, se for conhecido o intervalo de tempo que estes são processados nos equipamentos.

O objetivo prende-se, então, pela criação de um conjunto de campos, recorrendo à interface gráfica, onde devem ser introduzidos os parâmetros relativos à produção. O programa deve registar não só os anteriores, bem como a data e hora de início e do fim da produção, para que estes possam ser guardados na base de dados e consultados posteriormente de modo a fazer a correspondência com os registos de soldadura.

Durante o processo normal, são testadas peças, quanto à resistência ao esforço axial. Se for conhecido, o número interno do instrumento de monitorização de soldadura, para uma determinada soldadura é possível associar o esforço axial medido a uma peça em especifico. Esta funcionalidade pode revelar-se útil caso estejam a ser feitos testes no equipamento que envolvam associar essa variável ao estudo.

Assim, para realizar a introdução de parâmetros de soldadura no programa e consequentemente na base de dados são implementadas um conjunto de funções de forma a introduzir estas funcionalidades no programa.

A função denominada *data_enter_click*, presente no anexo K.1, é executada quando é pressionado o botão "Inserir dados de lote" na página principal do programa.

Inicialmente, a função vai desenvolver uma janela secundária e executar a verificação de todos os equipamentos definidos no programa, de forma a criar uma caixa de seleção com os equipamentos disponíveis. Em seguida, os restantes objetos de introdução de dados, botões e respetivas etiquetas são criados e posicionados na janela. As ações que devem ser tomadas quando os botões são pressionados estão definidas dentro da função, sendo que podem ser destacadas as seguintes:

- Função *start_lot* - responsável por iniciar o lote e pelo respetivo armazenamento interno dos parâmetros da produção num ficheiro, definida no anexo K.2;
- *F*unção continue_lote - responsável pela continuação do lote, dispondo da possibilidade de, caso o programa ou janela foram fechados, permitir a recuperação de todos os dados guardados sem que os mesmos se percam, definida no anexo K.4;

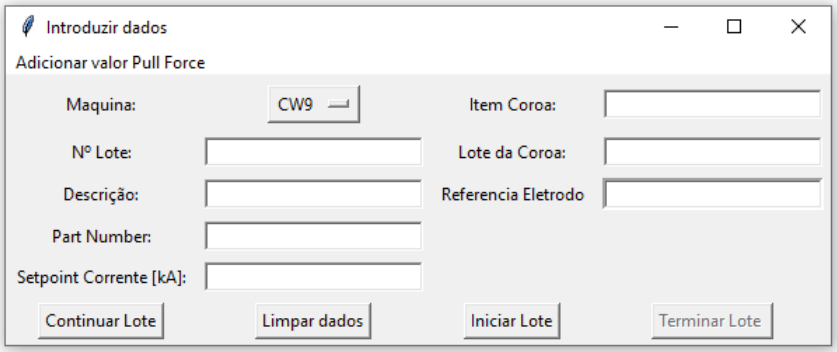


Figura 4.10: Janela de introdução de dados relativos á produção

- Função *end_lote* - responsável pelo armazenamento permanente, na base de dados,das especificações relativas à produção, assim como a data e hora inicial e final da mesma, definida no anexo K.3.

O tempo é registado através da biblioteca *datetime*, utilizando o método *now*. Desse modo, a data e hora são guardados juntamente com os restantes na base de dados recorrendo á função *insert_time_db*, definida no anexo L. Esta função é responsável por, dado um conjunto de dados relativos a parâmetros da produção incluindo a data e hora inicial, introduzir na tabela *updatedata* da base de dados.

Para garantir a posteiror correta correspondência de parâmetros de produção e registos de soldadura, os parâmetros da produção devem ser inseridos manualmente e idealmente antes da produção do equipamento iniciar. Isto porque, após a inserção dos dados é necessário iniciar a produção do lote no programa.

Quando é pressionado o botão "Iniciar Lote", visível na imagem 4.10, o programa executa a função *start_lot*, guardando a hora a que o lote foi iniciado, assim como a respetiva criação de um ficheiro de texto, cuja denominação do ficheiro é o número de lote (que contém os valores introduzidos na página). O ficheiro é guardado por meio da biblioteca *CSV* com o objetivo de recuperar os dados caso a janela/ programa seja fechado ou ocorra uma falha de energia.

Através do botão "Continuar Lote" que, quando pressionado, executa a função *continue_lot* que abre um explorador de ficheiros do sistema operativo. Com o ficheiro selecionado, recorrendo á biblioteca *CSV*, é extraído o conteúdo do ficheiro e a informação é novamente colocada nas entradas do programa. A hora inicial fica a recuperada no ficheiro. O programa, em seguida, comporta-se de forma similar à que seria se o lote tivesse sido iniciado no momento. Assim, é possível recuperar a hora de início da produção.

Assim que a produção no equipamento acabar totalmente devemos pressionar "Terminar Lote" (botão apenas fica disponível depois de ser iniciado ou continuado um lote). Esta ação vai executar a função *end_lot* que vai introduzir todas as entradas disponíveis na interface, juntamente com a

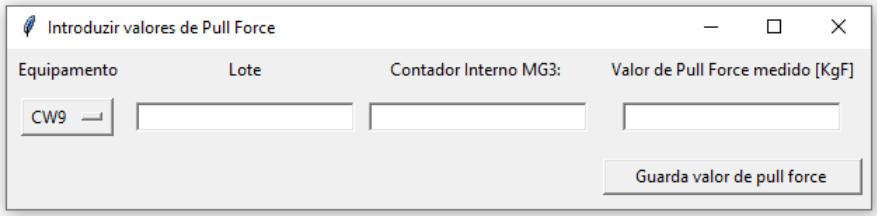


Figura 4.11: Janela de introdução de dados relativos ao teste destrutivo realizado

hora inicial e final em uma lista. Essa lista é em seguida inserida na base de dados, na tabela de dados relativos á produção, através da função *insert_time_db*,disponível no anexo L.

Existe também a opção de introduzir o esforço axial para uma determinada peça. Na janela de introdução de dados, através da opção "Adicionar valor Pull Force"disponível no canto superior esquerdo, na barra de ferramentas, da pagina de introdução de dados da figura 4.11.

Quando esta opção é selecionada uma nova janela com quatro campos de inserção de dados é criada. Estes são criados recorrendo á função *add_pull_force_value* definida na integra no anexo M.1, utilizando métodos similares aos utilizados pela função *data_enter_click*.

As entradas disponíveis são novamente o equipamento em que a peça foi soldada, o numero de lote, o contador interno do instrumento de monitorização e o valor de esforço axial medido através do teste destrutivo. Visto que esta janela é aberta recorrendo á janela de inserir dados relativos á produção, que já inclui o equipamento e o numero de lote, estes dados são preenchidos por defeito na janela aberta.

Possui um botão, "Guardar valor de pull force", que quando é pressionado é executada uma função, *save_pullforce_click*, definida no anexo M.2, esta avalia se os dos dados inseridos nos campos são validos. Caso os dados sejam validos, é criada e executada uma consulta á base de dados no âmbito de inserir os dados relativos ao esforço axial. Os dados são inseridos na tabela *pullforcedata*, que guarda todos os valores introduzidos de esforço axial para todos os equipamentos.

4.3.5 Correspondência de dados de produção e registos de soldadura

Os dados guardados até ao momento encontram-se tal modo que os registos de soldadura, os dados da produção e os valores de força medidos no teste destrutivo não possuem uma relação entre os mesmos. Assim de modo a possibilitar a consulta de registos de soldadura com base em diferentes parâmetros de produção é necessário associar os dados que tiverem relação entre si.

Como foi referido anteriormente, o método utilizado para identificar as relações entre dados da produção e os registos de soldadura é a hora e data em que ambos sucedem. O objetivo principal é então determinar se um registo de soldadura ocorreu ou não em um determinado intervalo de tempo a que estão associados os dados inerentes à produção.

Posto isto, para fazer a correspondência entre os dados, em primeiro lugar, o programa deve consultar todas as entradas que existem na tabela auxiliar, da base de dados, relativa aos dados da produção. Através desta, é criada uma lista que detém o resultado dessa consulta, onde estão incluídas a data e hora a que a produção no equipamento começou e terminou, o equipamento e os restantes dados relativos em especifico à produção. À posteriori, é basilar proceder-se à avaliação de cada entrada na lista anterior. Assim, o programa, por meio da data e hora inicial e final e o equipamento de cada entrada da lista, constrói uma consulta à base de dados para verificar se existem registos de soldadura que ocorram no equipamento entre a data inicial e data final.

Se a consulta à base de dados retornar um conjunto de registos, significa as soldaduras correspondentes ocorreram no período indicado, mas que no caso não foram ainda associados dados da produção a esses mesmos registos. Logo, tais entradas devem ser atualizadas com os dados associados à produção. Caso o resultado faculte uma lista vazia, significa que ou os registos que existem na base de dados já possuem dados relativos à produção associados ou ainda nem sequer existem esses registos na base de dados. Assim, para um conjunto de dados relativos à produção é possível associar registos de soldadura provenientes do instrumento de monitorização.

A função responsável por elaborar a associação dos dados, das diferentes tabelas, é a função *date_log_match*, disponível em detalhe no anexo N. A última utiliza como único argumento o caminho para a base de dados, visto que todas as operações ocorrem com dados já contidos na mesma. Inicialmente, deve ser estabelecida uma ligação à base de dados, sendo feita uma consulta relativamente a todas entradas presentes na tabela auxiliar com dados relativos à produção.

Para cada um desses, é avaliado qual o equipamento a que os dados da produção são referentes e o intervalo de tempo em que ocorreu a mesma, ocorrendo a transformação dos dados da produção em objetos *datetime*. A figura 4.12 apresenta excertos do código relacionados com a consulta a todos os elementos da base de dados referentes à tabela que guarda os detalhes da produção. Além do mais, é possível, igualmente, observar na figura referida a forma como o ciclo de avaliação dos registos se inicia assim como a criação de umas das variáveis, a data e hora de inicio do lote, do tipo *datetime* que adapta-se à utilização para comparar os registos de soldadura.

Em consequência é assim construída uma nova consulta à base de dados, utilizando os dados anteriores, filtrando por resultados que não tenham numero de lote associado, mas que tenham ocorrido no equipamento e nas respetivas datas em questão. Através dessa consulta é possível determinar se existem registos que necessitam de ser atualizados com detalhes referentes à produção.

```
295     #criar e executar consulta a todos os elementos da tabela update data
296     cursor_db.execute("SELECT * FROM updatedata")
297
298     #guardar dados obtidos pela consulta em variavel
299     data_para_update= cursor_db.fetchall()
300
    //
307     #iterar sobre dados de produção
308     for datas in data_para_update:
    //
310
345         #criar objetos para a hora inicial e final no formato datetime
346         data_inicio_lote = datetime(int(datas[1][:4]), # Ano
347                                     int(datas[1][5:7]),# mes
348                                     int(datas[1][8:10]), #  dia
349                                     int(datas[1][11:13]), #  hora
350                                     int(datas[1][14:16]), #  minuto
351                                     int(datas[1][17:19])) #  segundo
```

Figura 4.12: Exemplo pratico da obtenção de dados relativos á produção e inicio do ciclo de comparação de registos

Se o resultado da consulta for uma lista vazia, compreende-se que não existem registos que necessitem de correspondência de dados e o programa avança para a entrada seguinte de detalhes de produção.

Logo, sucede-se uma avaliação do outro conjunto de dados de um determinado intervalo de tempo para um equipamento.

No entanto, interessa ainda referir que caso o resultado da consulta de dados fosse uma lista de registos, tal significaria a existência de necessidade de verificar se estes são ou não válidos para serem considerados a atualizar. O programa deve, assim, iterar sobre estes resultados obtidos e comparar se a hora e data do registos ocorreu no intervalo de tempo em que se ocorreu a produção. Esta comparação ocorre mediante a transformação da data e hora de cada registo em um objeto do tipo *datetime*, comparando, simultaneamente, com os objetos, já neste formato de tempo, relativos à data inicial e final da produção. O exemplo da aplicação pratica encontra-se presente na figura 4.13. Ademais, se o registo tiver ocorrido dentro do intervalo de tempo em que os detalhes relativos à produção indicam, então o *LogId* deste registo é guardado. Após se verificarem todos os registos, os que se revelarem que ocorreram no intervalo de tempo que estamos a verificar são atualizados na base de dados.

A atualização é feita recorrendo aos detalhes da produção e aos *LogId's* registados durante a análise anterior, esta é uma variavel de gestão da base de dados. Consiste em um valor único para cada registo e são geridos pela biblioteca *SQLite3*. É utilizada pois a tarefa de encontrar elementos específicos em uma base de dados de dimensão média mostrou-se um processo demorado. Isto acontecia devido ao facto da base de dados ter de avaliar todos os registos até encontrar o registo procurado, processo que se pode revelar longo.

É de especial preocupação devido ao facto do tamanho da base de dados crescer à medida que novos registos são adicionados. Desta resultaria que o processo de correspondência de registos de soldadura e parâmetros de produção demoraria exponencialmente mais tempo, devido ao crescimento dos registos na base de dados.

Portanto, a variável *LogId* é concebida e utilizada pela base de dados para realizar a localização de um registo específico. Visto que a base de dados utiliza páginas para guardar as entradas, esta variável permite à base de dados localizar de imediato em que página se encontra o registo procurado, diminuindo substancialmente a procura na base de dados pelo registo a atualizar. Por

```
388         #Criar uma lista para guardar LogId de registos que necessitem ser atualizados
389         lista_ID=[]
390 
391         #Iterar registos obtidos pela consulta á base de dados
392         for item in data_db:
393 
394             #formatação se necessario - introduzir millisegundos porque existem registos que incluem
395             if item[8][6:8]=='':
396 
397                 hora_log = datetime(int(item[7][6:10]), int(item[7][3:5]), int(item[7][:2]),
398                                     int(item[8][:2]), int(item[8][3:5]), int(0))
399             else:
400                 hora_log = datetime(int(item[7][6:10]), int(item[7][3:5]), int(item[7][:2]),
401                                     int(item[8][:2]), int(item[8][3:5]), int(item[8][6:8]))
402 
403             #se a data de um registo estiver contida entre a hora inicial e final da produção
404             if hora_log > data_inicio_lote and hora_log < data_fim_lote:
405 
406                 #adicionar á lista o identificador do registo
407                 lista_ID.append([item[-1]])
```

Figura 4.13: Ciclo de verificação se registo pertence ao intervalo de tempo

isso, para atualizar os registos encontrados são utilizados os *LogId* de cada registo.

Por outro lado, é igualmente essencial corresponder os valores de força de tração com os registos de soldadura a que estes dizem respeito. Para desempenhar essa tarefa é construida uma função, denominada *pull_force_update*, definida no anexo O. Esta tem maior simplicidade, pois os dados são atualizados diretamente, não existindo necessidade de verificar tantos registos. A função recolhe todos os dados relativos aos testes de tração feitos que foram registados na base de dados. É necessário, então, introduzir estes dados relativos a testes de tração.

Através do equipamento, número de lote e do número do contador interno é possível localizar o registo de soldadura correspondente. Para cada conjunto de dados relativos a testes destrutivos o programa deve utilizar os valores anteriores de forma a construir uma consulta à base de dados para que seja introduzido o valor obtido de força no registo da soldadura específica. O valor de força é assim introduzido no campo "*Description*"do registo a qual estiver associado. Assim, o programa deve percorrer todos os dados existentes de valores de testes feitos e tentar atualizar cada um na base de dados. A função *pull_force_update* é executada a seguir à função*date_log_match* devido à necessidade do número de lote para localizar o registo para substituir a informação que estiver no campo "*Description*".

De forma modo a permitir ao utilizador realizar a correspondência de registos da base e dados foi criado um botão que inicia o processo. O anterior apresenta-se localizado na página da atualização da base de dados segundo o nome "Associação de dados do lote". Assim, e presente na figura 4.8, quando pressionado, o programa cria e executa um processo paralelo que exerce a função denominada *match_routine*, que pode ser consultada no anexo P.

Esta é responsável por gerir as funções anteriormente descritas para efetuar a associação de registos. As funções necessitam de ser executas num processo paralelo, visto que possuem um numero elevado de operações, no ambiente normal o programa ficaria à espera de que cada uma acabasse de ser executada não permitindo ações por parte do utilizador.

Consequentemente, com vista à criação do processo paralelo que execute as funções de atualização de base de dados, recorre-se à biblioteca *threading*. Enquanto se procede à execução das funções, o processo paralelo fica em espera e não o programa, permitindo a utilização normal do mesmo. Assim, a função *match_routine* é executada em um processo paralelo.

```
2191    #botão de inicio de associação de dados, quando pressionado é inciada thread da funcao match_routine recorrendo a uma função lambda
2192    button_data_association=Button(data_update_wi,text="Associação de Dados de Lote",
2193                                   command= lambda : threading.Thread(target=match_routine, args=()).start())
2194
2195    #posicionamento do botão na janela de atualização de dados
2196    button_data_association.grid(row = row_i+3, column=1,columnspan=3,padx=5,pady=5)
```

Figura 4.14: Implementação do botão de atualização de dados

Na figura 4.14 é demonstrado como é criado o botão de "Associação de dados de lote"e como é construido o processo paralelo onde as funções de associação de dados vão operar.

Então, com vista à criação do processo paralelo, através da ação de pressionar o botão, é preciso atribuir ao parâmetro *command*, do objeto do botão, a criação e execução do processo. O parâmetro espera um função como entrada, mas como apenas é necessário executar uma linha de código para iniciar o processo paralelo recorre-se a uma função *lambda*. Esta, uma função anónima, pode ter um número arbitrário de argumentos, contudo apenas uma expressão. Assim, pode ser utilizada para criar e iniciar o processo paralelo diretamente no objeto do botão definindo-a no parâmetro *command* do objeto do botão.

Durante a associação de registos de soldadura a parâmetros da produção, ou seja enquanto o processo paralelo estiver a ser executado, o programa deve bloquear temporariamente o acesso á pagina de atualização da base de dados. A função, *match_routine*, anexo P, começa por desabilitar as funcionalidades do botão que abre a janela de atualização e associação dos dados. Em seguida, fecha a janela, ficando apenas a página principal do programa aberta. É executada a função *date_log_-match* que introduz dados relativos à produção nos registos de soldadura e, posteriormente, a função *pull_force_update* que introduz valores específicos de esforço axial no registo da correspondente peça. O progresso feito por estas funções é demonstrado recorrendo a uma barra de progresso acompanhada de uma mensagem informativa apresentados na página principal do programa.

Como foi mencionado no início do presente subponto, era necessário uma associação dos registos de soldadura e dos dados de produção, após a sua introdução na base de dados. Para isso, o conjunto de funções descritas anteriormente introduziram, nos registos de soldadura, os dados referentes à produção e aos testes de destrutivos realizados. Deste modo, com as ferramentas corretas, existe a possibilidade de obter um conjunto de registos de soldadura utilizando como base os dados da produção. Com o acesso aos dados facilitado é possível entregar ou demonstrar os dados ao utilizador para que ofereçam uma maior compreensão do processo. Segue-se, no próximo subponto, o desenvolvimento das ferramentas de acesso à base de dados assim como de tratamento e apresentação dos mesmos.

4.3.6 Extrair registos da base de dados

Com a introdução e posterior associação de registos de soldadura e dados da produção na base de dados, surge a necessidade de aceder aos mesmos. As funções implementadas, até aqui, apenas realizam consultas especificas à base de dados relacionadas com a sua funcionalidade. Por esse motivo, não se apresentam como aptas para realizar consultas, segundos as novas possibilidades provenientes da associação de dados.

Deste modo, o objetivo é criar as ferramentas necessárias para desenvolver uma consulta à base de dados. A consulta deve ser personalizada consoante a informação que é pretendido obter. Os dados resultantes dessa consulta têm de ser disponibilizados para o utilizador final em formato padrão e consistente. Além disso, a extração dos dados deve igualmente ser simplificada, mas ainda assim que ser capaz de filtrar por diferentes parâmetros e combinações dos mesmos. Em termos de implementação, as consultas à base de dados procedem-se de forma similar à utilizada até agora por outras funções como a *download_data_db*, disponível no anexo G.

Neste caso específico é construída uma consulta, mas com base nas entradas do utilizador. As últimas vão ser introduzidas através da interface gráfica do programa me desenvolvimento. Os resultados obtidos são, de seguida, escritos em ficheiro e disponibilizados ao utilizador do programa.

Assim, a função é iniciada quando o botão "Extrair da base de dados", disponível na página principal da aplicação, é pressionado. Desse modo, surge a abertura de uma janela, utilizando o mesmo método que para as outras janelas secundárias criadas até aqui. O resultado é então a construção da estrutura da página, que inclui os objetos com as entradas relativos a filtros a utilizar durante a consulta á base de dados. Os filtros disponíveis vão ser, essencialmente, os campos relativos a dados da produção, assim como a data e o contador interno do instrumento de monitorização, permitindo diferentes combinações destes campos.

Assim, são geradas as entradas e as etiquetas para inserir os seguintes dados: data, número inicial e final do contador do instrumento de monitorização de soldadura, o equipamento e um campo que diz respeito a dados da produção este é denominado *"Keyword"*, querendo fazer referencia a palavra chave.

Neste campo são apresentadas as opções: Número de lote, número de peça, descrição, referencia do elétrodo e o item e lote da coroa utilizada. É também criado um botão, com etiqueta "Gerar ficheiro", que quando pressionado deve avaliar o conjunto de entradas disponíveis na janela. O botão, assim como as entradas criadas, é visível na figura 4.15. O código relacionado com a construção destas estruturas está disponível na função *extract_click* e pode ser consultada no anexo Q.1.

Quando o botão "Gerar registos"for pressionado é executada a função *generate_file*. Esta é definida dentro da função *extract_click* e pode ser conferida no anexo Q.2 . É necessário obter os as entradas inseridas pelo utilizador de modo a criar uma consulta com base nos mesmos. O programa executa, assim, a função *query_buider*, disponível no anexo R, que tem como funcionalidade, dado um conjunto de parâmetros, gerar e realizar uma consulta à base de dados recorrendo às mesmas, retornando o resultado dessa consulta. O conjunto de parâmetros introduzido revelam-se como as entradas disponíveis na página.

Após a função *query_buider* retornar os dados obtidos, através da consulta, é questionado ao utilizador se pretende conceber um ficheiro com base nos resultados obtidos, caso existam registos

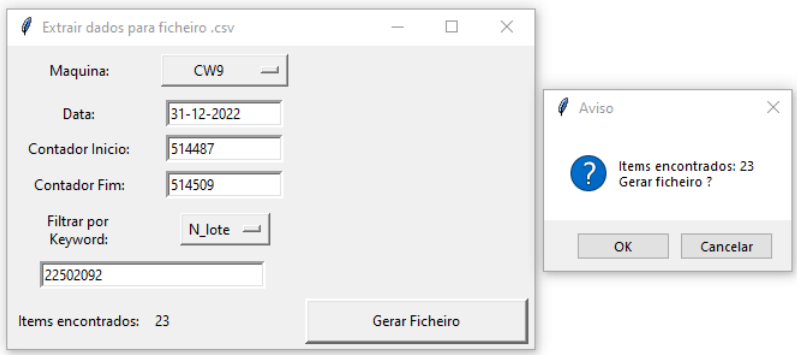


Figura 4.15: Janela de extração de dados em funcionamento

para a consulta efetuada. O número de registos encontrados é mostrado no canto inferior esquerdo da janela de "exportar dados", assim como na janela de aviso. O exemplo da figura 4.15 exibe o comportamento do programa quando são encontrados registos para os parâmetros utilizados, sendo que no respetivo exemplo são aplicados todos os filtros disponíveis na página. Se o utilizador decidir que deseja gerar um ficheiro com base nos registos encontrados, deve pressionar "Sim"na janela de aviso aberta. Se for pressionado sim, o programa utiliza os métodos já aplicados de criação de ficheiros *.CSV*, recorrendo á biblioteca CSV e aos métodos *writerow* e *writerows*.

Tendo em conta o anterior, a denominação do ficheiro é originada com base nos filtros utilizados. Por outro lado, a organização interna do ficheiro é a idêntica à da tabela da base de dados correspondente. Assim, a primeira linha é populada com uma linha a indicar o equipamento a que os registos dizem respeito. A segunda linha é o cabeçalho da "tabela"de dados, sendo idêntica aos campos existentes na base de dados. Estas duas linhas são escritas sequencialmente utilizando o método *writerow*. As restantes linhas dizem respeito aos registos de soldadura em si e são escritas no ficheiro utilizando uma lista com todos os registos obtidos como argumento do método *writerows*.

Em síntese, até ao presente subponto o programa revelava a capacidade para introduzir novos registos de soldadura na base de dados, registos de produção e associar estes dados. Com a explicação anterior existirá a possibilidade adicional de retirar esses dados da base de dados conforme as entradas do utilizador. Isto permite a consulta de registos contidos na base de dados utilizando os parâmetros introduzidos, assim como a exportação desses registos para guardar através de um ficheiro *CSV*. Com este formato o ficheiro com os registos pode ser utilizado, por um numero muito elevado de ferramentas, visto que, este formato e organização de ficheiros é amplamente utilizada.

4.3.7 Extrair relatório

O programa de gestão de registo de soldadura é capaz de extrair os registos da base de dados, como demonstrado no ponto 4.3.6. Existe, no entanto, necessidade de implementar a capacidade do programa fazer uma apresentação gráfica e numérica dos dados contidos na base de dados. Tal habilidade facilitará o desenvolvimento e progresso de estudos relacionados tanto com o processo como com a produção, imprescindíveis para a analise e bom funcionamento dos equipamentos.

Para tal, é necessário uma função que seja capaz de fazer uma primeira análise a um dado conjunto de dados. Sendo possível a extração dos registos da base de dados, esta analise apenas será uma perspetiva global do processo para uma determinada condição. Essa condição pode ser qualquer um dos dados introduzidos relativos à produção, como o número de lote, o número da peça, a descrição, o item da coroa, o lote da coroa e a referência do elétrodo.

Através de umas destas condicionantes devem ser feitas as consultas necessárias à base de dados, de modo a obter todos os registos para esta. Os registos devem ser, em seguida, analisados quanto à sua média e respetivo afastamento da mesma. Estes devem ser apresentados na forma gráfica construindo um gráfico de linhas e um histograma.

Em termos práticos, a função *data_report_window* foi desenvolvida com o intuito de implementar a janela secundária e os seus objetos. Esta é caracterizada no anexo S.1 e utiliza métodos idênticos aos utilizados no desenvolvimento de outras janelas secundarias e objetos presentes nas diferentes funcionalidades do programa.

Desse modo, em prol da avaliação de quais dados devem ser usados para a construção de gráficos, existe um botão definido na página, que quando pressionado executa a função *calc_click*. Essa função diz respeito ao conjunto de ações que devem ser tomadas de modo a extrair os dados da base de dados e realizar os cálculos, com vista à avaliação referida. A função é disponibilizada para consulta no anexo S.2.

Em primeiro lugar, a função deve obter a lista de todos os equipamentos, depois, recorrendo a essa lista, utiliza a função *query_builder*, abordada no ponto 4.3.6, para realizar consultas à base de dados. Nesta caso, as consultas construídas apenas utilizam a palavra chave introduzida na interface gráfica. A consulta é repetida para todos os equipamentos e os registos obtidos são guardados, temporariamente, em uma lista. Segue-se, a necessidade de calcular para cada uma das variáveis possíveis, a média e o desvio padrão do conjunto de dados encontrados.

Por conseguinte, são construídos oito grupos onde um registo pode estar incluído, estes são:

1. Registos com o valor contido entre a média e a média mais o desvio padrão;

2. Registos com o valor contido entre a média mais o desvio padrão a média mais duas vezes o desvio padrão;

3. Registos com o valor contido entre a média mais duas vezes desvio padrão e a média mais três vezes o desvio padrão;

4. Registos com o valor superior a média mais três vezes o desvio;

5. Registos com o valor contido entre a média e a média menos o desvio padrão;

6. Registos com o valor contido entre a média menos o desvio padrão e a média menos duas vezes o desvio padrão;

7. Registos com o valor contido entre a média menos duas vezes desvio padrão e a média menos três vezes o desvio padrão;

8. Registos com o valor inferior a média menos três vezes o desvio

Com o resultado da média e do desvio padrão de um conjunto de dados, são avaliadas todas as entradas desse conjunto de modo a distribuir os registos pelos conjuntos de grupos referidos. Estes grupos servem de contentores na construção do histograma. Este processo é repetido para todas as variáveis e as que possuírem a média diferente de zero são, em seguida, enviadas para a função responsável por criar os gráficos que vão ser utilizados no relatório. Essa função é denominada por *plots_generator*, definida no anexo T.

Assim, a anterior é implementada para construir os gráficos necessários com o objetivo da sua introdução no relatório. Para isso, a função recorre à biblioteca *Numpy* e *Matplotlib* para atingir os fins pretendidos. Os cálculos necessários, como a média e o desvio padrão são realizados utilizado a biblioteca *Numpy*. Já os gráficos são construidos utilizando os recursos disponibilizados pela biblioteca *Matplotlib*.

Posteriormente, a construção possível de gráficos terá duas tipologias diferentes, de linhas e de histograma. No gráfico de linhas são utilizados diretamente os valores disponibilizados pelo base de dados e os valores de média e desvio padrão calculados. Estes valores são agrupados em conjuntos da mesma dimensão para que sejam todos introduzidos no mesmo gráfico. Por sua vez, no caso do histograma, o último utiliza contentores cujos dados podem estar inseridos, sendo a sua definição como os grupos anteriormente enumerados.

Os dois tipos de gráficos conseguidos podem ficar no formato similar ao apresentado nas figuras 4.16 e 4.17. Mais especificamente, os anteriores são relativos à resistência do processo para um determinado lote e são utilizados no relatório exemplo exportado, presente no anexo V. Estes são guardados numa pasta, na diretoria de trabalho da aplicação, denominada *graphs*.

Após estar concluída a criação de gráficos para as variáreis, cuja média é diferente de zero, é necessário introduzir a informação num ficheiro PDF. Este processo é desenvolvido pela função *reports_generator*, disponível para consulta no anexo U. A anterior utiliza como argumento os dados resultantes dos cálculos realizados pela função *calc_click* e os gráficos obtidos através da *reports_generator*. Por meio dos dados anteriores, constrói-se cada página do PDF do relatório, com a seguinte estrutura: cabeçalho comum, cálculos e gráficos obtidos para cada variável.

Assim, para extrair um relatório, é necessário pressionar o botão com o nome "Relatórios" na página principal do programa. Na página secundária, para extração de relatórios, é apresentada uma caixa de seleção múltipla com as diferentes categorias de entradas a utilizar, um campo para introdução de texto e um botão denominado "calcular".

Introduzida a categoria e o valor especifico para a mesma, quando é pressionado o botão, o programa vai fazer as operações descritas anteriormente. De seguida, apresenta o número de registos encontrados, para as entradas anteriores, e questiona o utilizador se pretende gerar o ficheiro. Se o utilizador pressionar *ok*, o ficheiro é gerado e guardado na pasta *reports* da diretoria do programa.

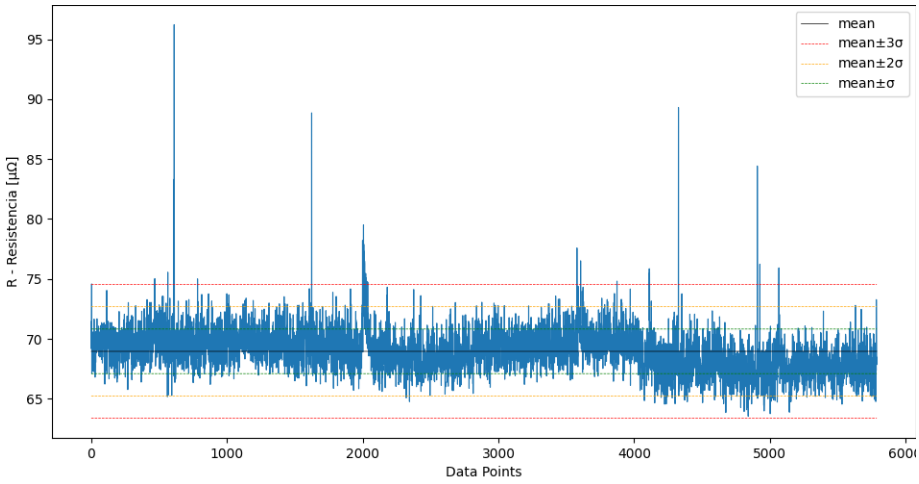


Figura 4.16: Exemplo de gráfico de linhas gerado pela função *plots_generator*

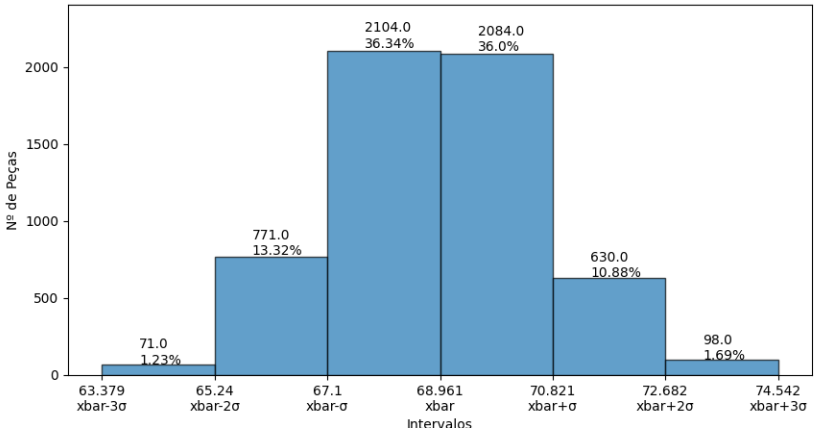


Figura 4.17: Exemplo de histograma gerado pela função *plots_generator*

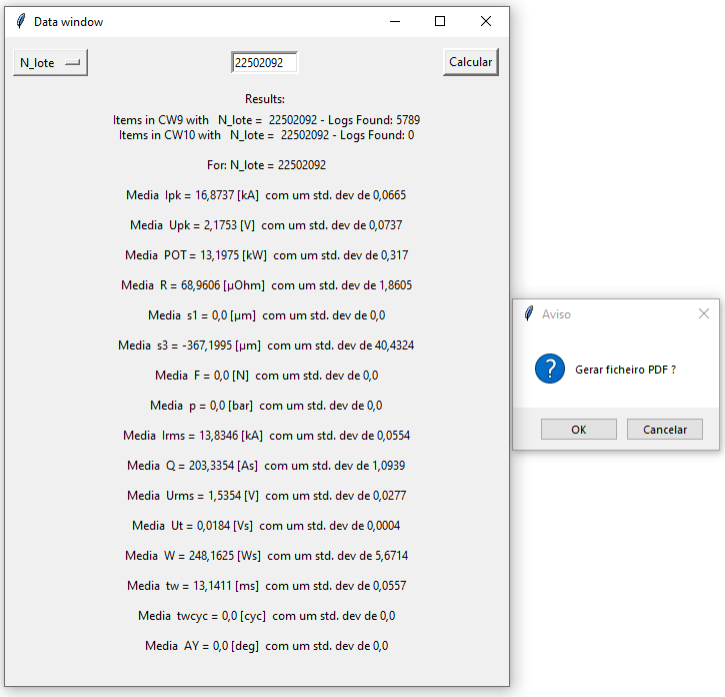


Figura 4.18: Implementação do botão de atualização de dados

Então, o funcionamento esperado da página é exibido na figura 4.18. A mesma exemplifica, para um dado número de lote com registos associados na base de dados, o comportamento do programa e os resultados apresentados na interface. Ainda sobre esta questão, no anexo V encontra-se disponível o relatório que seria produzido com base nas informações introduzidas na figura 4.18.

4.3.8 Desempenho dos equipamentos

Uma métrica muito importante em ambiente industrial é a relação entre o tempo ativo e inativo de um equipamento.

Isto porque, os mesmos devem estar, idealmente, operacionais o maior tempo possível. Cada vez que um determinado equipamento se encontra parado prejudica a produção da respetiva linha e, consequentemente, poderá criar desvantagens em termos de organização e *timings* para com o cliente.

No entanto, durante a produção existem contra-tempos como avarias, afinações, ajustes necessários, entre outros. Estas situações geram períodos de inatividade nos equipamentos, sejam estes curtos ou mais longos.

Para controlar este nível da produção, é utilizado um documento que tem como métrica, essencialmente, o número de peças produzidas por hora. Estes dados são introduzidos pelo operador, com base no número de peças produzidas pelo equipamento, na última hora, sendo preenchido a cada hora do dia. Tal controlo é imprescindível para considerar o desempenho de cada equipamento para um determinado espaço de tempo.

Como analisado ao longo do presente relatório, o programa de gestão de registos de soldadura tem acesso aos registos de soldadura de cada equipamento. Visto que esse é o principal processo dos equipamentos, é valido assumir que se existirem registos de soldadura o equipamento esteve em operação. Nos períodos em que não exista registos de soldadura, então o equipamento teve parado.

O programa dispõe assim da capacidade de realizar a produção de um equipamento para uma determinada data, oferecendo vários formatos possíveis para a visualização do desempenho de um equipamento.

Portanto, e com base no documento original, o desempenho do equipamento é registado uma vez por hora, ou seja, vinte e quatro vezes por dia. O programa tem acesso detalhado à data e hora de cada registo, podendo dividir eficientemente o dia em intervalos arbritarios de tempo em que esses registos podem ser introduzidos. Isto oferece a potencialidade de dividir o dia em blocos não só de uma hora, mas também de trinta, vinte, quinze, dez e cinco minutos.

Assim, a função *machine_performance_w*, foi desenvolvida para permitir ao utilizador introduzir um conjunto de datas e o tamanho do período do dia a utilizar, de forma a gerar gráficos com base nestes dados. Esta pode ser consultada no anexo W.1 e é responsável, por um lado, pela implementação da janela onde os dados vão ser inseridos e, por outro, pela execução de um conjunto de ações com os anteriores.

De tal forma, a janela e as entradas criadas são demonstradas na figura 4.19. Para executar esta função é necessário acessar na barra de ferramentas, da página principal, à opção *Desempenho equipamentos* no menu.

No que diz respeito à introdução das datas por parte do utilizador no programa, tal ação deverá ser executada recorrendo à expansão *Tkcalendar*, que introduz uma ferramenta com um calendário à biblioteca do *Tkinter [8].*

As datas podem ser inseridas escolhendo no calendário e, em seguida, pressionado o botão *Inserir* presente ao lado da entrada de texto em que se pretende introduzir a data. A tarefa desempenhada quando estes botões são pressionados é definida no anexo W.2 e W.3.

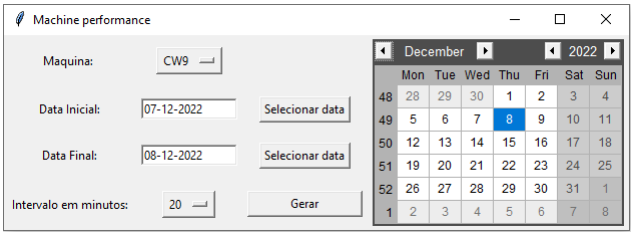


Figura 4.19: Janela de extração de gráficos de desempenho dos equipamentos

Quando o botão "gerar" é pressionado, a função *b_gerar_click* é executada. Esta vai registar as entradas da data introduzida pelo utilizador. Em seguida, procede-se uma verificação do intervalo de datas para determinar quantas dias existem entre as datas.

É assim construida uma lista com todas as datas em que o programa deve produzir um gráfico. Obtido tal intervalo de datas, são realizadas várias consultas à base de dados de modo a determinar os registos na base de dados para este conjunto.

Para cada dia os registos de soldadura são guardados numa lista. A lista é ordenada e os registos são divididos em grupos, sendo o número de grupos definido pela quantidade de intervalos de tempo que um dia dispõe. Por exemplo, se forem utilizados intervalos de uma hora existem vinte e quatro intervalos para descrever um dia completo, já se forem considerados intervalos de trinta minutos existem quarenta e oito intervalos para descrever o dia. Assim, consoante a entrada do utilizador os registos são divididos nos grupos correspondentes.

Através da formação desses grupos, é calculado a média de número de soldaduras por hora, dividindo o número de peças presentes em cada intervalo pelo período de tempo a que o intervalo em si corresponde. Estes valores são guardados e serão utilizados para fazer o gráfico para determinado dia.

Na figura, 4.19 é demonstrada a interface do programa na página do desempenho dos equipamentos. Os gráficos gerados pelo programa são exemplificados na figura 4.20 que representa um dos gráficos resultantes das entradas da figura 4.19. Estes foram gerados para o equipamento *Crown Welding 9* entre os dias 7 de dezembro de 2022 e 8 de dezembro de 2022.

Assim, é possível extrair gráficos, como o demonstrado, para um intervalo de tempo arbitrário. Além do mais, a prinicipal vantagem de ser possível alterar o intervalo de tempo utilizado e assim oferecer mais detalhes sobre esse intervalo de tempo.

Tome-se como exemplo a figura 4.20, para uma melhor compreensão da teoria. Foram utilizados intervalos de tempo de vinte minutos, assim sempre que se verifique um ponto neste gráfico que seja zero significa que ocorreu um período de 20 minutos de inatividade. Desse modo, é evidente que o equipamento esteve inativo entre pouco depois das oito da manhã e as onze da manhã. Não obstante, ocorreu, igualmente, uma paragem entre as 19:00h e as 20:00h, possivelmente derivado da mudança de turno que ocorre neste intervalo de tempo. Outros pontos mínimos locais, como entre as 04:00 e 05:00 da manhã ou entre as 16:00h e as 17:00h, nos quais a produção no equipamento caiu pontualmente. Isto pode significar que correu uma paragem de tempo inferior a 20 minutos. Seria possível perceber melhor quanto tempo o equipamento esteve parado ao requerer um gráfico para mesma data, mas utilizando um intervalo de tempo menor.

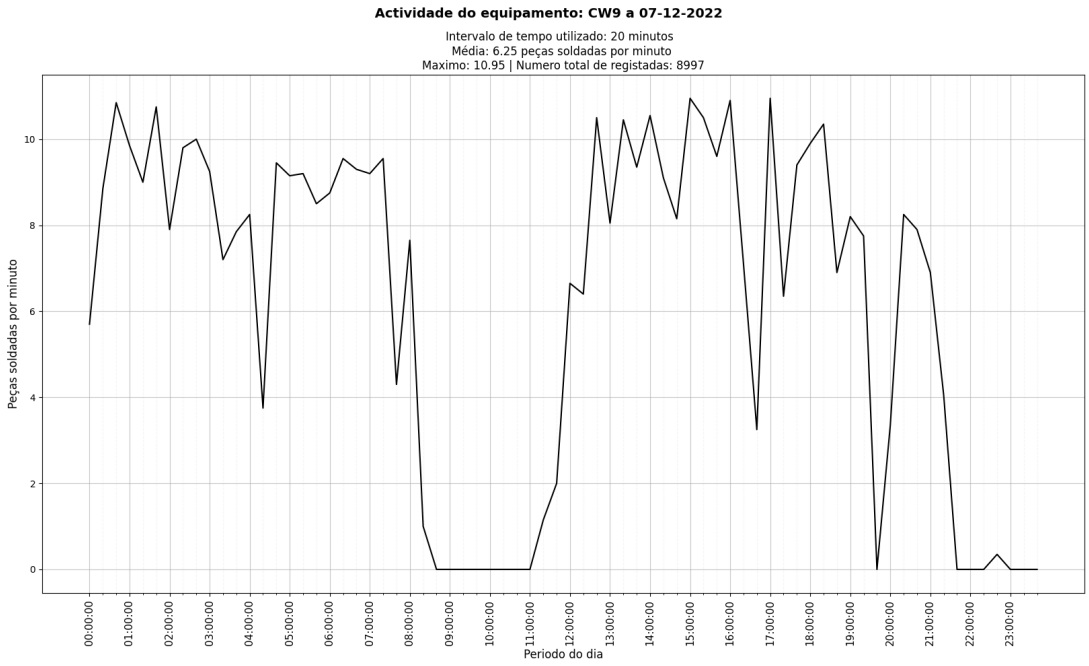


Figura 4.20: Gráfico do desempenho de um equipamento utilizando intervalos de 20 minutos

Assim, estes gráficos permitem que a produção seja capaz de visualizar e analisar o desempenho de um dado equipamento, de modo a perceber a sua variação para uma determinada data. Esta é mais detalhada quanto menor for o intervalo de tempo escolhido para a construção do gráfico.

4.3.9 Gestão do programa

Outra questão importante do programa é a necessidade de funções que permitam verificar e alterar os acessos aos caminhos dos ficheiros de registos dos equipamentos, assim como a possibilidade de adicionar outros equipamentos.

Estas funcionalidades são úteis para evitar que seja alterado o código do programa quando ocorrem tais necessidades. Revelam-se assim de extrema importância, visto que o programa é suposto ser compilado em um executável com todas as dependências inerentes.

À vista disso, para atingir esses fins, foi adicionado um novo menu à aplicação. Este, situado na barra de tarefas, sob o nome "Equipamentos", quando selecionado exibe as três funcionalidades anteriores: "Adicionar", "Alterar"e "Listar".

Antes de se proceder à interpretação destas funções é vital explicar como o programa armazena a lista de equipamentos. A lista de equipamentos que o programa utiliza encontra-se num ficheiro da diretoria, sendo o mesmo denominado de "*configs.CSV*". Este é um ficheiro do tipo *CSV* que ,essencialmente, contem duas entradas por linha, ou seja, uma referente à identificação do equipamento e outra com o caminho para o local onde se encontram guardadas as pastas com ficheiros de registo para esse equipamento. O caminho para a base de dados fica, igualmente, registado neste ficheiro utilizando o nome "*db_RW*"em vez do nome do equipamento.

Aquando da iniciação do programa, a lista anterior é então importada, através de um pequeno ciclo executado que utiliza a biblioteca *CSV* para interpretar o ficheiro.

Logo, os conjuntos de equipamento e caminho são importados e configurados na forma de tuplo para serem introduzidos em uma lista global que contem a informação de todos os equipamentos. Por esse motivo, muitas funções quando são executadas apresentam a necessidade de verificar esta lista para disponibilizarem as opções de equipamentos ao utilizador.

O caminho para a base de dados, por sua vez, é guardado como uma variável global. Assim, sempre o que o programa necessita de alguma destas informações obtém-na através das variáveis globais descritas.

No entanto, pode existir necessidade, como foi mencionado de efetivar alterações a estas configurações. Para tal, existe necessidade de implementar mais um conjunto de funções que apresentem essa possibilidade ao utilizador.

Recorrendo a recursos similares aos utilizados até agora no âmbito da criação de janelas secundárias e objetos, são implementadas as funções *config_change*, *machine_list_add* e *machine_list_stats*.

A função *config_change* implementa a janela que servirá para alterar o caminho para o conjunto de pastas de ficheiros de registo de um determinado equipamento. Esta possui uma caixa de seleção com os equipamentos registados e uma entrada de texto em branco. Se for pressionado o botão "Ver caminho atual"essa entrada de texto é preenchida com o caminho gravado para o equipamento selecionado na caixa de seleção.

Para se proceder à alteração do caminho, pode ser inserido, manualmente, na caixa de texto o novo caminho desejado ou se selecionar "Procura Pasta"é aberto um explorador de pastas do *windows* para que seja indicado o caminho. Em seguida, para guardar as alterações, deve ser pressionado o botão "Guardar". Este executa a função *save_config_change* presente no anexo X.5,

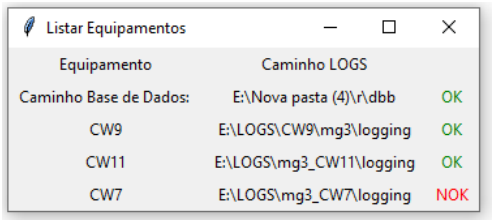


Figura 4.21: Janela de verificação de acessos do programa

que altera a lista global dos equipamentos, eliminando a entrada anterior para esse equipamento e inserindo o caminho presente na entrada de texto. Logo após, é alterado o ficheiro "*configs.csv*"de modo a efetivar a alteração realizada.

Ademais, para adicionar um novo equipamento é utilizada a função *machine_list_add*, sendo a mesma responsável por construir a janela de introdução de um novo equipamento. Desse modo, são construidas duas entradas, uma para o nome do equipamento a registar e outra para o caminho para os ficheiros com registos de soldadura desse equipamento.

Quando as entradas são preenchidas e é pressionado "guardar o novo equipamento"este é adicionado à lista global e é alterado o ficheiro de configurações como na função *save_config_change* de modo abranger este equipamento.
É executada a função, *create_table*, disponível no anexo X.4. Esta utiliza como argumento o nome do equipamento a adicionar, essencialmente, realiza uma consulta à base de dados em que cria uma nova tabela para o novo equipamento.

Para verificar o correto acesso a todos os caminhos que o programa necessita é implementada a função *machine_list_stats* disponível no anexo X.2 . Quando é pressionado o botão "Lista", no menu da barra de ferramentas para os equipamentos é aberta uma janela em que o programa exibe o resultado de verificação do acesso aos caminhos dos ficheiros dos equipamentos e da base de dados. Na figura 4.21 é demonstrado o comportamento do programa, quando este botão é pressionado. Neste, o caminho para o equipamento "CW7"não existe e, por isso, encontra-se marcado a vermelho. Os restantes estão a verde, significando que o programa consegue aceder aos caminhos associados a esses equipamentos.

Assim, encontraram-se implementadas as funcionalidades de gestão do programa de modo a facilitar futuras alterações necessárias. Estas são explicadas em maior detalhe no anexo X.

4.4 Implementação do programa na linha de produção

A fase final do desenvolvimento do programa de gestão de registo de soldadura será a sua implementação e, para tal, o mesmo deve ser testado. É necessário testar o comportamento dos algoritmos, anteriormente descritos, em vista à escala dos dados e à velocidade com que estes são atualizados e associados. Para realizar os testes é utilizado para apenas um equipamento, a *Crown Welding 9*.

Este é o equipamento apropriado, pois tem uso elevado, o que origina muitos registos de soldadura e parâmetros da produção a associar. Para além disso, este é um dos equipamentos mais críticos deste processo e, assim, é de todo o interesse que este programa seja implementado de forma a facilitar a recolha dos dados de soldadura.

Com tais características, o programa é compilado utilizando a biblioteca *Pyinstaller*, sendo através desta criada uma pasta que contem todas as dependências necessárias para o programa funcionar sem necessidade de instalação das bibliotecas ou do *python*. Nesta pasta fica igualmente contido o executável do programa.

De ressaltar que, em prol da acessibilidade posterior em relação à monitorização do programa, são criados dois executáveis. Por um lado, um cuja a única janela que fica aberta é a janela do programa, devendo ser utilizado pela produção. Por outro lado, outro executável que abre também a janela de comandos para que seja possível visualizar as mensagens de "*debug*"espalhadas pelo programa. .

Aplicado este programa em um computador adjacente ao equipamento, foi procedida a alteração da ordem de trabalhos de modo a abranger a introdução dos parâmetros da produção no equipamento antes do inicio dessa produção.

4.4.1 Testes finais ao programa

O programa de gestão de registos de soldadura esta agora pronto para ser testado em operação na produção. Visto que existem nuances que apenas se consegue perceber quando o programa é utilizado para a sua finalidade, no caso do programa desenvolvido um dos testes que era imperativo realizar era o do comportamento com o aumento de registos na base de dados.

Inicialmente surgem diversos problemas relacionados principalmente com a velocidade de corresponderia e atualização de registos na base de dados e com a formatação de alguns dados.

Problemas como a formatação dos dados foram prontamente resolvidos visto que se tratava de pormenores técnicos relacionados com o modo da escrita do código que facilmente podiam ser alterados.

No entanto, os problemas relacionados com a velocidade de atualização dos registos mostraram maior dificuldade na otimização. A barra de progresso que o programa utiliza durante estes processos surgiu exatamente pela inicial necessidade de visualizar se o programa não tinha bloqueado.

Foi necessário varias iterações destes algoritmos de forma a simplificar e reduzir ao máximo o numero de processos necessários para que estes decorram de forma mas célere. Estas passaram por alterações das consultas a base de dados assim como da comparação dos registos em si. Um exemplo disto é que inicialmente o programa, a quando de introduzir novos registos de soldadura na base de dados, comparava todos dos parâmetros de cada entrada. Na realidade, era apenas necessário

comparar a data e a hora para garantir que um registo esta ou não contido na base de dados. Este tipo alterações deram uma velocidade satisfatória a estes processos

Assim após alguns ajustes ao programa, este mostrou-se efetivo a realizar a tarefa para a qual foi desenvolvido. À data de escrita deste relatório, o programa encontra-se aplicado ainda apenas no equipamento *crown welding 9* e possui cerca de 377938 registos de soldadura com parâmetros de produção associados.

Existem ainda assim registos que não possuem estes valores associados. Inicialmente foi considerado um problema no programa, contudo à posteriori foi verificado que se tratava de falha humana, não ocorrendo sequer introdução de parâmetros de produção por parte do operador do equipamento.

Os maiores problemas que surgiram com a implementação do programa estão relacionados com a associação de dados na base de dados. O algoritmo sempre fez o trabalho suposto, mas foi necessária muita otimização para tornar o processo mais estável com o aumento da escala dos dados.

A solução passou pela utilização dos referidos *log_id*, assim como otimização da função de forma a comparar apenas o estritamente necessário.

Capítulo 5

Controlo do processo

Já com os instrumentos de monitorização de soldadura e o programa de gestão de registos de soldadura em funcionamento na linha de produção pode-se iniciar a realização de testes de forma definir os limites de controlo do processo adjacente. De forma a determinar os limites de controlo para o processo de soldadura por resistência é necessário ter em conta as seguintes fases:

1. Identificar as variáveis criticas do processo: o primeiro passo para determinar os limites de controlo do processo é identificar as variáveis inerentes ao processo que tem efeito na qualidade final da soldadura. Nesta podem estar incluídos parâmetros elétricos como a corrente utilizada e o tempo de soldadura assim como parâmetros mecânicos como o formato do elétrodo, pressão do elétrodo a quando da soldadura, entre outros.

2. Registar dados do processo: É necessário fazer o registo de diferentes parâmetros durante o processo de forma a perceber se existem tendências ou padrões anormais que possam ser indicadores de problemas com o processo.

3. Calcular o índice de capacidade do processo Cpk: O calculo do Cpk para cada variável do processo permite que seja medida a capacidade do processo de produzir soldaduras dentro de um certo nível de qualidade estabelecido.

4. Determinar os limites de controlo: com base no índice de capacidade do processo é possível determinar limites de controlo para cada variável critica do processo. Estes limites definem um intervalo, em que um determinada variável critica deve estar contida num intervalo de valores de forma a que o processo produza peças dentro do padrão de qualidade.

5. Implementar um plano de controlo: Implementar um plano de monitorização que tome ações corretivas quando as variáveis do processo excedem os limites de controlo. Estas ações podem ser um ajuste nos parâmetros de soldadura, a troca do elétrodo, entre outros.

6. Monitorização continua: o processo deve ser continuamente monitorizado de forma a garantir que as variaveis de controlo estão dentro dos limites e ajudar a identificar alterações no processo que necessitem de uma ação corretiva. Durante a monitorização constante devem também melhorados os planos de controlo.

No âmbito do presente relatório, o principal objetivo é disponibilizar as ferramentas necessárias para que a monitorização do processo seja possível. Estas ferramentas, devem assim facilitar esse mesmo estudo através da disponibilização e associação de registos de soldadura e produção.

A determinação de varáveis criticas, calculo da capacidade do processo, o plano de controlo recaem sobre ações que devem ser tomadas por equipas dedicadas ao processo.

5.1 Estudos realizados

Uma das primeiras verificações da utilidade do instrumento de monitorização de soldaduras e do programa de gestão dos registos, produzidos pelo mesmo, produziu-se aquando do primeiro estudo com base nos registos que existiam na base de dados.

Tendo em conta o mencionado anteriormente, o estudo referido apresenta como principal objetivo o conhecimento e compreensão sobre quais são as variáveis do processo, que para cada tipo de peça fabricada que traduzem uma maior variação no valor de força registado no teste destrutivo conduzido à respetiva peça. Deste modo, foram registados os valores de força medidos para um conjunto de peças. Portanto, as entradas que pertencerem ao mesmo tipo de peças são consideras em um grupo.

Posteriormente, e para cada um destes grupos, verifica-se quais as três variáveis que possuem maior correlação com a variação da força da soldadura. Assim, estas variáveis são contabilizadas para cada grupo de forma a compreender, para a totalidade dos grupos, quais foram as variáveis que se manifestaram em maior quantidade com correlação à variação da força medida.

Assim para o conjunto de dados avaliado, as variáveis que surgem mais vezes com correlação à força da soldaduras são: A resistência, a corrente média e a potencia.

Visto que a potencia elétrica pode ser calculada por meio da multiplicação da resistência pelo quadrado da corrente, é claro o motivo pelo qual a potencia aparece nesta lista. Como o calor gerado durante uma soldadura por resistência elétrica, depende da potencia assim como do tempo de soldadura e do próprio material e sabendo que o tempo de soldadura usualmente varia muito pouco durante o processo as variareis com maior influencia na soldadura vão ser a resistência e a corrente, que consequentemente representam a potencia.

No entanto, existem algumas duvidas relativas ao teste anterior que recaem sobre o conjunto de variações, não elétricas, do processo que podem afetar o valor de força medido.

Como foi referido anteriormente, neste processo existem varias condições que podem comprometer a qualidade da soldadura. Visto que estes fatores não foram tomados em conta no estudo inicial, é necessário testar para perceber de que forma diferentes conjuntaras de operação do equipamento podem ocorrer durante o processo normal.

Para tal foram criados 7 grupos, cada um destes é composto por duas hipóteses em que dispomos de 10 peças para cada uma das hipóteses.

Os modos considerados são os seguintes:

1. Peças manuseadas com/sem luvas
2. Coroas manuseadas com/sem luvas
3. Condensador acima/abaixo da altura nominal, mas dentro da especificação.
4. Coroas de lotes diferentes;

5. Condensador inclinado/normal;

6. Condensadores com variação na especificação da tampa do condensador

7. Soldadura realizada com 3,2 ou 1 cabos de soldadura.

Assim, para cada um destes grupos são soldadas 10 peças com base em cada uma das premissas, por exemplo no caso 1, são soldadas 10 peças em que o operador esta a utilizar luvas para manusear os condensadores, são também soldadas 10 peças sem luvas para que seja possível compara os resultados obtidos.

Os grupos que mostraram uma maior variação, entres os dois conjuntos de peças, foram:

1. Peças manuseadas com/sem luvas: As peças manuseadas sem luvas possuem em média uma força de soldadura menor em 35,5% do que as peças que foram manuseadas com luvas. Isto ocorre pois a gordura presente nas mão adere ao condensador, o que provoca complicações aquando da soldadura, visto que existe uma camada de gordura que precisa de ser superada, adicionando variação ao sistema.

2. Coroas manuseadas com/sem luvas: De forma idêntica ao ponto anterior, as coroas manuseadas com luvas apresentam em média valores de força superiores do que se forem manuseadas com as mãos. A diferença entre a média dos valores de força medidos esta na ordem dos 17/

3. Condensador acima/abaixo da altura nominal: Este grupo revelou os resultados mais surpreendentes, dado que não era suposto ocorrer variação quando os condensadores essencialmente são similares, alterado a altura do condensador, mesmo estando dentro da especificação do mesmo. Isto ocorre devido as condições que o equipamento necessita para soldar a peça.

   Este equipamento, a *crown welding 9*, realiza a descarga quando o elétrodo positivo atinge um determinado valor de força dentro de uma tolerância em cima do condensador. De ressaltar que pelo motivo do elétrodo encontrar-se acoplado a um motor elétrico, a força é medida com base no binário que o motor está realizar. Assim, o equipamento, em primeiro lugar, envia o elétrodo para uma determinada posição com base na altura do condensador. Se quando chegar a essa posição o binário do motor estiver no intervalo pretendido, é produzida a descarga elétrica.

   A variação do processo surge, devido a este pormenor do motor se deslocar para uma posição com base na altura do condensador a fabricar. Isto ocorre pois, se um condensador estiver ligeiramente acima da especificação vai ficar a cima do ponto de soldadura definido no equipamento e assim quando o elétrodo chegar a essa posição efetivamente vai estar a realizar mais esforço sobre o condensador do que se este tivesse a especificação da altura a baixo. Desde que, a força esteja dentro dos limites quando o elétrodo esta na posição de soldadura esta é realiza. Como é permitida uma variação de 10% da força realizada sobre o condensador e usualmente esta força esta na ordem dos 285 Newton, efetivamente podem ser realizadas soldaduras com pressoes entre os 256 Newton e os 313,13 Newton.

   O que se verifica, no teste, é que os condensadores com a altura acima da especificação implicam valores médios de força superiores em cerca de 15% em relação aos que possuem a altura a baixo da especificação.

4. Soldadura realizada com 3,2 ou 1 cabos de soldadura: De longe o grupo mais intrigante da experiência, neste forma realizadas 10 soldaduras utilizando 3 cabos de soldadura, outras 10 utilizando dois cabos de soldadura e 10 utilizando 1 cabo de soldadura. Seria de esperar que qualidade da soldadura fosse decrescendo a medida que foram retirados cabos de soldadura, visto que a resistência global do sistema tende a subir apareceriam mais pontos onde seria possível se perder energia na forma de calor. No entanto, com a redução de cabos, a força da soldadura tendeu a subir. Uma possível justificação para este comportamento prende-se pela força que os cabos de soldadura fazer no elétrodo positivo, que no caso é o elétrodo móvel. São utilizados 4 cabos com 120 mm^2 de diâmetro para levar a potencia do controlador de soldadura ate ao elétrodo. Este tipo de cabos oferece muita resistência ao movimento e a dobragem assim muita resistência mecânica ao sistema e consequentemente desalinhado o próprio elétrodo .

Este estudo permitiu perceber que tipo de ações podem desde ja ser tomadas para diminuir a variação do processo, ações como o uso obrigatório de luvas por parte do operador, podem trazer melhorias imediatas no processo. Outras relações encontradas como da altura dos condensadores podem ajudar a melhor determinar os limites de controlo, visto que a altura do condensador é importante para o processo, pode também ser reduzida a tolerância de força permitida ao equipamento para realizar a soldadura para que seja garantido que a soldadura ocorreu num intervalo de força mais pequeno e consequentemente mais fiável.

Capítulo 6

Conclusão

O presente projeto de estágio teve como principal objetivo a integração do instrumento de monitorização de soldaduras nos equipamentos em que não estavam integrados e a criação de uma forma de gestão dos registos gerados por este.

Deste modo, existiu um processo prático que permitiu a aplicação de todas as medidas necessárias aquando das respetivas modificações a cada equipamento, de forma a tomar partido das funcionalidades disponibilizadas pelo instrumento de monitorização de soldadura. Tendo em conta o anterior, os instrumentos foram colocados na rede em prol da acessibilidade geral de todos os registos para, consequentemente, ser desenvolvimento um programa de gestão desses mesmos registos.

Assim, foi construído o programa de gestão de registos de soldadura que tira proveito dos registos de soldadura produzidos pelo instrumento de monitorização. Utilizando uma ampla lista de recursos disponibilizados para o *Python*, foram construídos algoritmos com diferentes objetivos, desde a criação de uma base de dados para guardar os registos como forma de extrair dados contidos na mesma, entre outros.

A primeira conclusão principal prende-se de que por meio da utilização do programa, as equipas da Engenharia do Processo, auferem uma nova dimensão no que se trata de dados relativos ao processo de soldadura por resistência elétrica, de modo a que o seu estudo seja facilitado. À vista disso, estas são atualmente utilizadas pelas equipas no seu dia a dia. As capacidades de rastrear a componente de soldadura por resistência do processo foram também expandidas visto que agora existe possibilidade de manter os registos em uma base de dados com outros parâmetros relevantes como o numero de lote ou a identificação do modelo possibilitando assim o acesso futuro a estes conjuntos de dados.

O cumprimento dos procedimentos de operação dos equipamentos estabelecidos é fundamental para o funcionamento eficiente do programa de gestão de registos de soldadura. Este é o principal entrave ao funcionamento correto do programa e como tal um dos objetivos futuros estabelecidos é reduzir ao máximo a quantidade de entradas que tem que ser preenchidas manualmente.

Outras melhorias que podem ser feitas de forma a simplificar o uso da aplicação. A possibilidade utilizar a aplicação na rede interna da fabrica, de forma a possibilitar o acesso aos dados a qualquer dispositivo na rede é uma proposta feita para uma melhor solução ao problema proposto.

Para tirar o melhor proveito do instrumento de monitorização e das suas novas capacidades de intervir no processo é necessário realizar mais estudos, principalmente sobre os produtos fabricados e o processo aplicado a esses produtos em especifico.

Os três principais objetos de estudo sugeridos no futuro são:

- Quais as principais variáveis da soldadura que influenciam a sua qualidade ;
- Caracterizar, para essas essas variáveis, os limites de controlo do processo para os diferentes modelos fabricados;
- Desenvolver conjunto de ações que, de forma mais eficiente, retornam o processo ao controlo no caso de desvio da qualidade.

Com estes vai ser possível ter um maior controlo sobre o processo , pressupondo uma monitorização constante e intervenção no mesmo por parte do instrumento de monitorização de soldadura que agora é possibilidade com as alterações que foram realizadas nos equipamentos.

Assim, o futuro, passa pelo estudo mais aprofundando do processo e este é simplificado devido ao acesso aos dados que apenas é possível devido ao programa desenvolvido durante o este relatório.

É ainda de ressaltar que todo o processo do programa estará aberto a novas reformas, exatamente pelo descrito anteriormente. A inovação tecnológica, dependendo do grau de aplicabilidade nos referidos equipamentos e condensadores, fará variar a necessidade da evolução do programa.

Bibliografia

[1] *csv — CSV File Reading and Writing.(Inglês) [Csv - Escrita e leitura de ficheiros CSV]*. URL: https://docs.python.org/3/library/csv.html.

[2] *DB-API 2.0 interface for SQLite databases.(Inglês) [Interface DB-API 2.o para bases de dados SQLite]*. URL: https://docs.python.org/3/library/sqlite3.html.

[3] MIYACHI EUROPE. *MG3D Digital Measuring Unit Operating Instruction .(Inglês) [Instruções de operação do instrumento de monitorização MG3D]*. v1.6. 2010. URL: https://amadaweldtech.com/wp-content/uploads/2019/02/76609409EN-BA-MG3D-V1_6.pdf.

[4] Donald E. Knuth. "Fundamental Algorithms". Em: Addison-Wesley, 1973. Cap. 1.2.

[5] *Miscellaneous operating system interfaces.(Inglês) [Interfaces de operações do sistema operativo]*. URL: https://docs.python.org/3/library/os.html.

[6] *NumPy Reference.(Inglês) [Manual de referencia do modulo Numpy]*. URL: https://numpy.org/doc/stable/reference/index.html#reference.

[7] *Python 3.11.2 documentation.(Inglês) [Documentação oficial do Python 3.11.2]*. URL: https://docs.python.org/3.

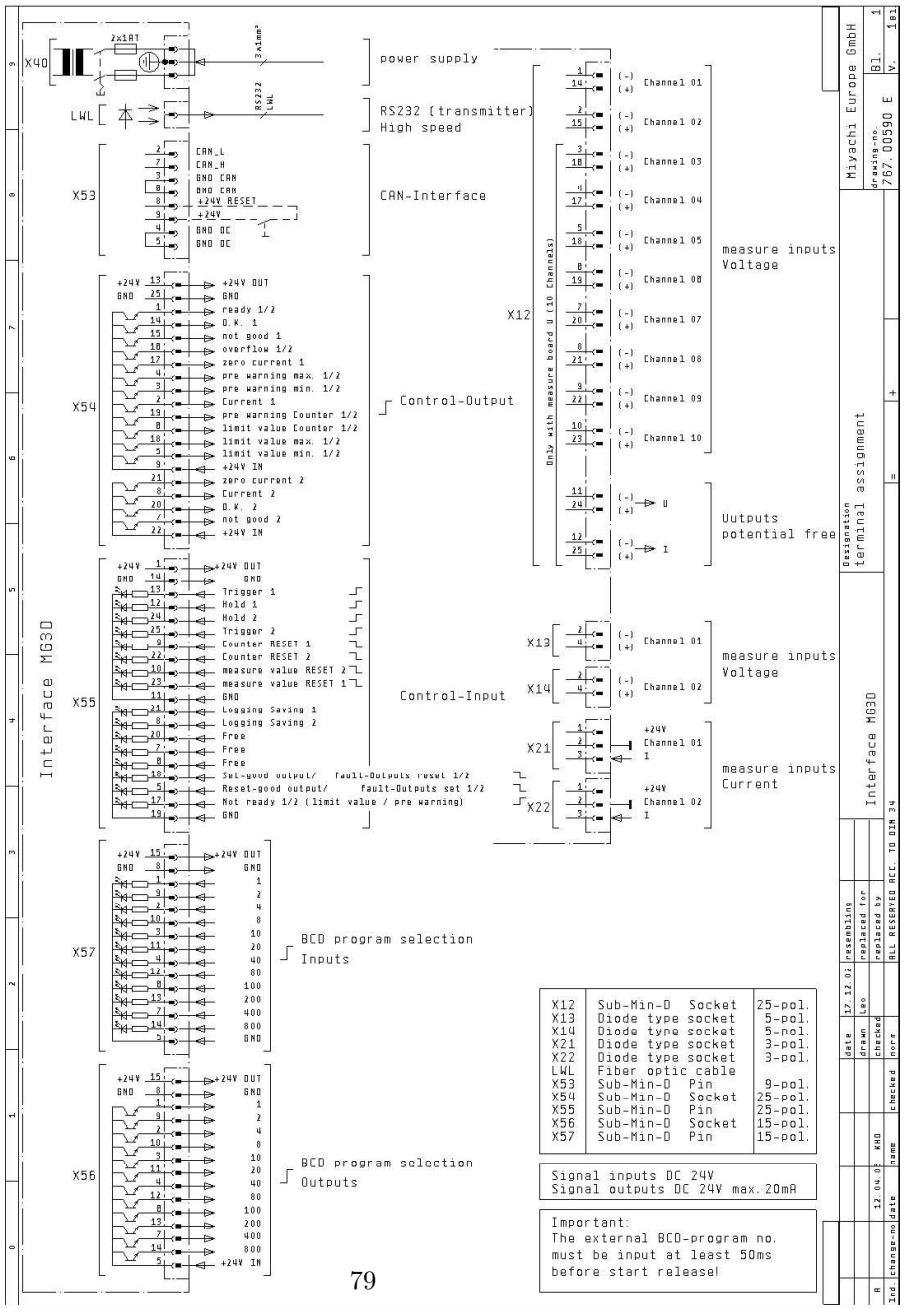
[8] *Python interface to Tcl/Tk.(Inglês) [Interface Tcl/Tk compativel com Python]*. Documentação oficial do modulo Tkinter. URL: https://docs.python.org/3/library/tkinter.html.

[9] *The Art of Computer Programming*. Four volumes. Seven volumes planned. Addison-Wesley, 1968.

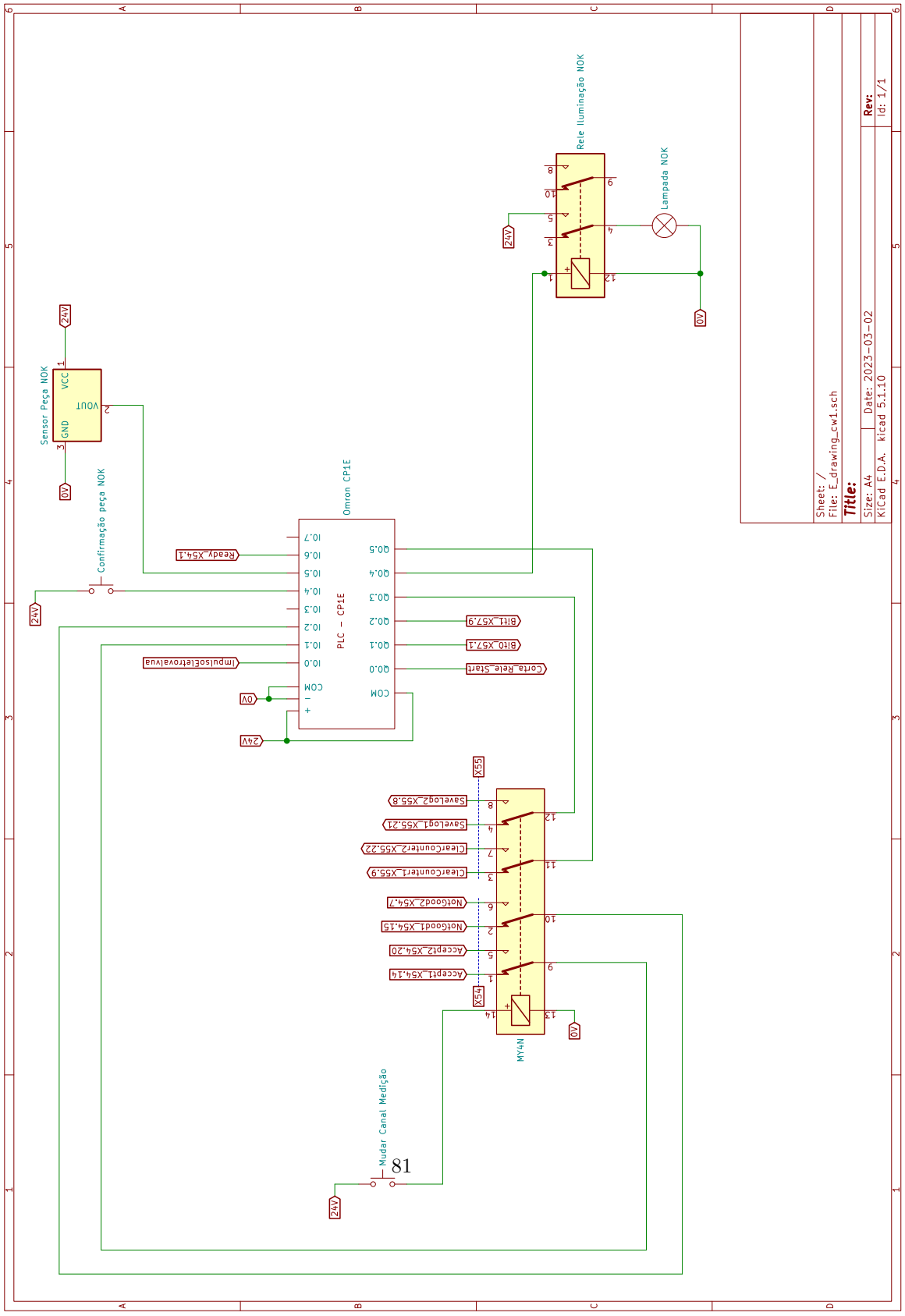
Appendix A: Esquema de interface do instrumento MG3D

7.3 Enclosures

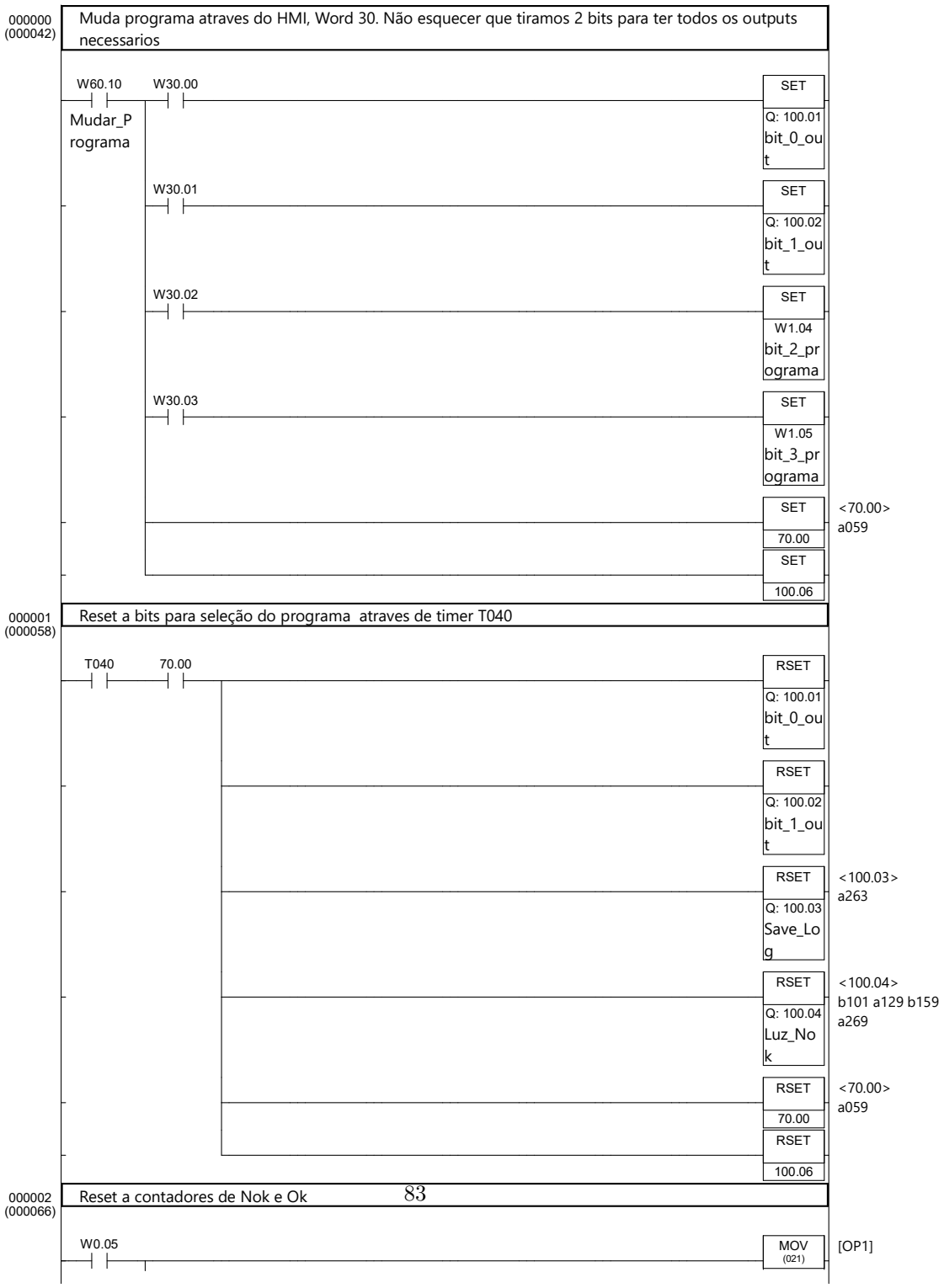
7.3.1 Connection diagram, Interface MG3D, 767.00590 E

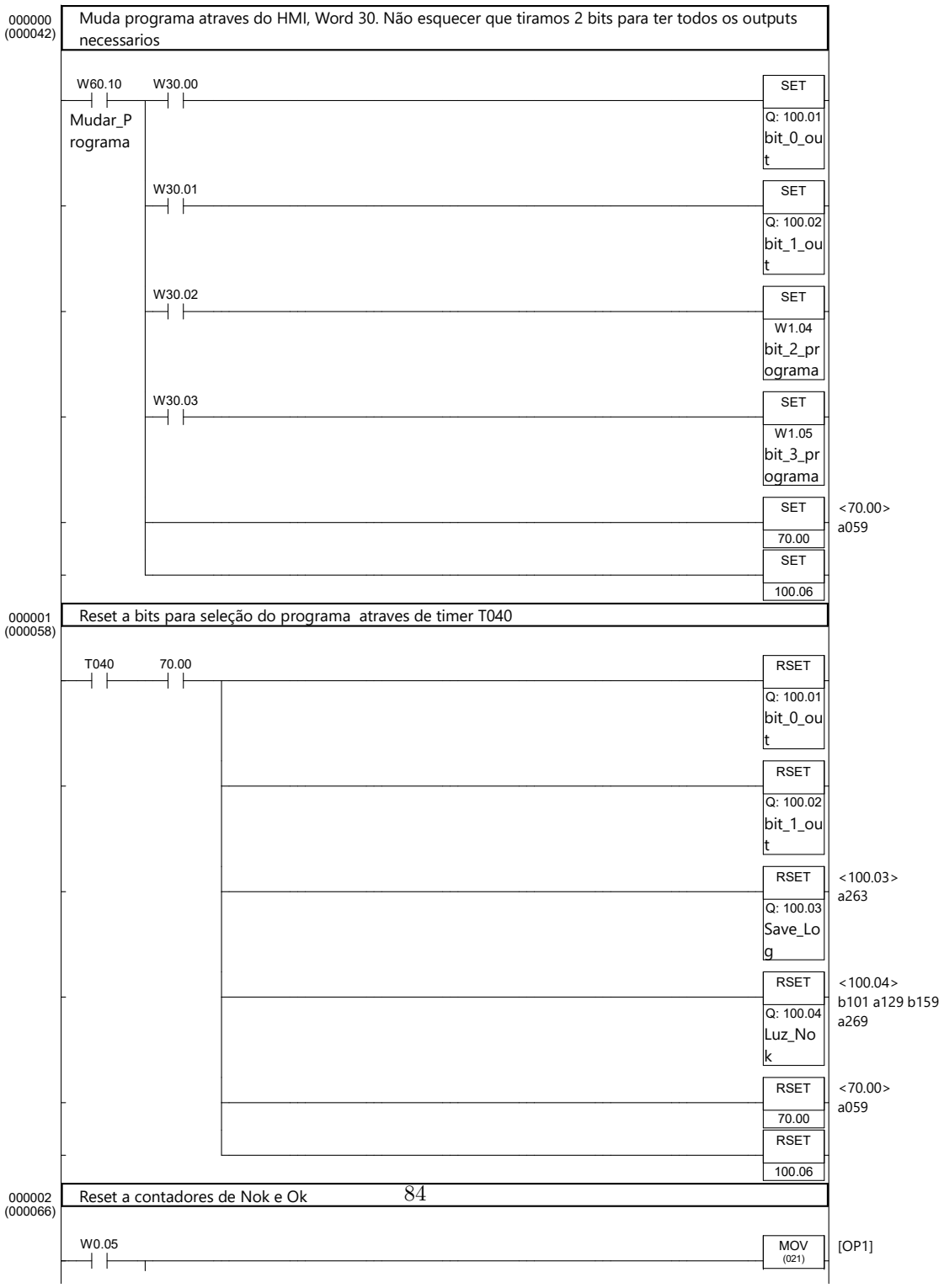


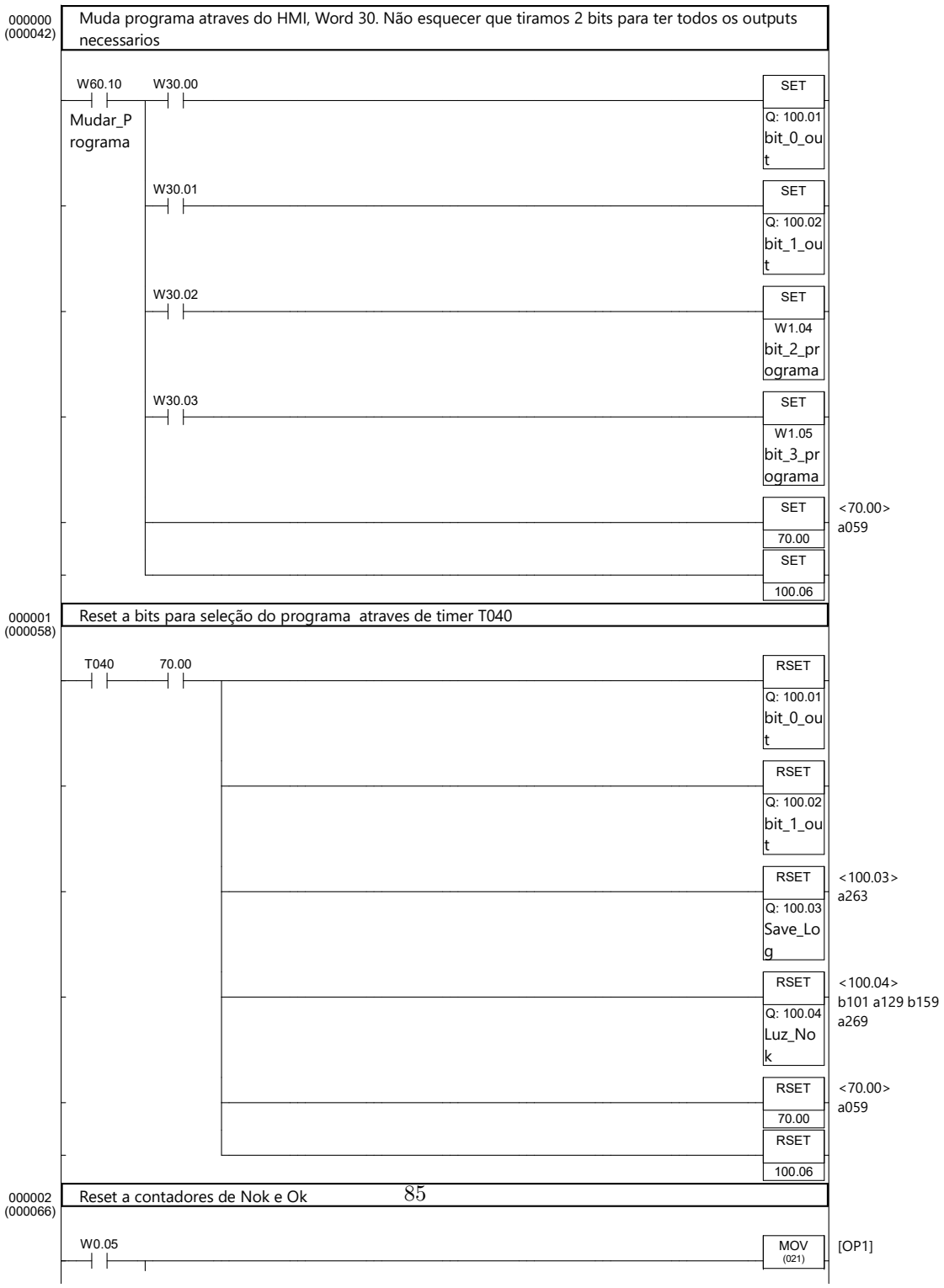
Appendix B: Esquema elétrico de alterações proposta - CW1

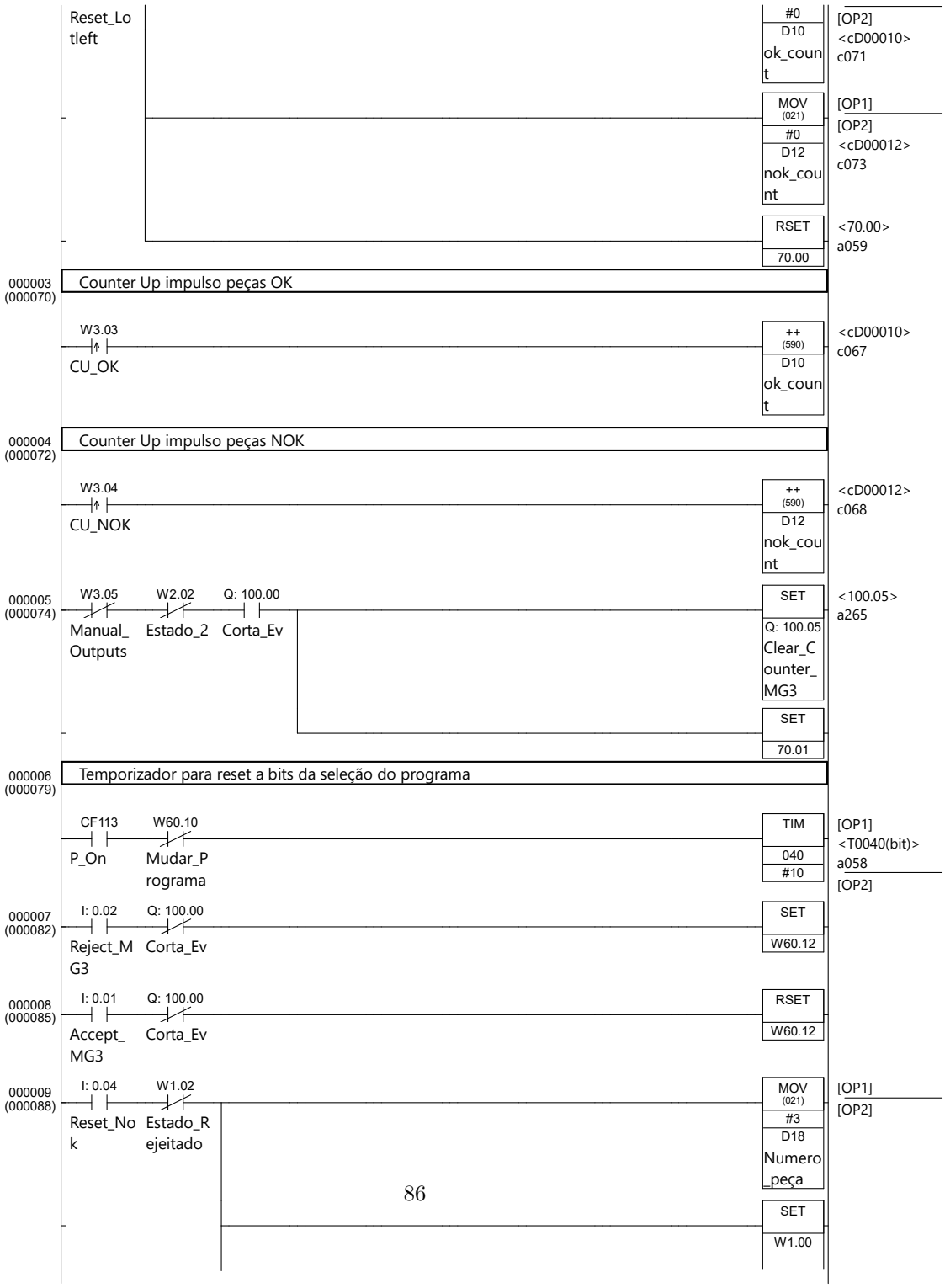

Rele Iluminação NOK
Lampada NOK
24V
0V
Sensor Peça NOK
VCC
VOUT
GND
Confirmação peça NOK
Omron CP1E
PLC - CP1E
Ready_X54.1
ImpulsoEletrovalva
Bit1_X57.9
Bit0_X57.1
Corta_Rele_Start
0V
24V
X55
X54
SaveLog2_X55.8
SaveLog1_X55.21
ClearCounter2_X55.22
ClearCounter1_X55.9
NotGood2_X54.7
NotGood1_X54.15
Accept2_X54.20
Accept1_X54.14
MY4N
Mudar Canal Medição
Sheet: /
File: E_drawing_cw1.sch
Title:
Size: A4
Date: 2023-03-02
KiCad E.D.A. kicad 5.1.10
Rev:
Id: 1/1

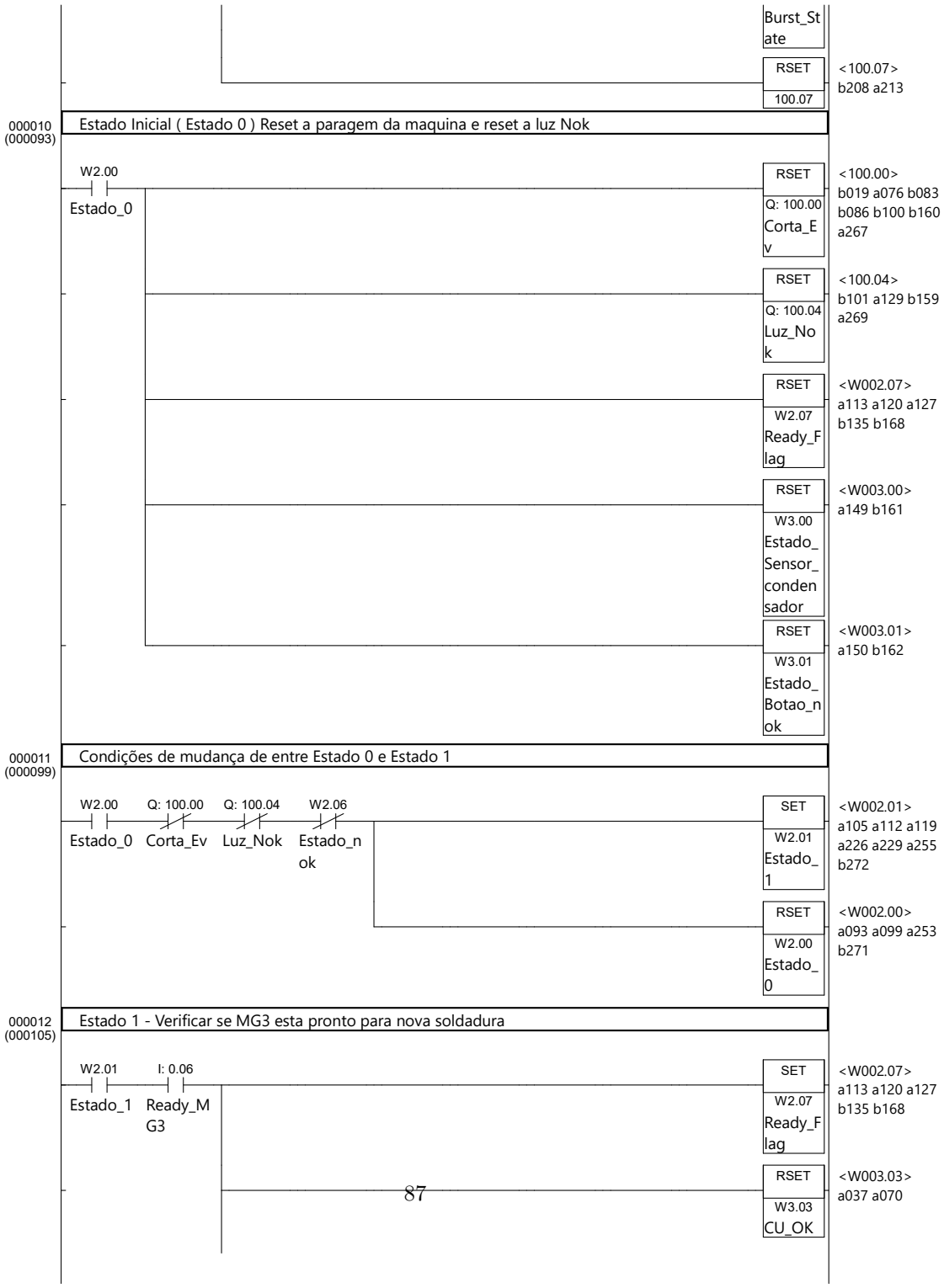
Appendix C: Diagrama Ladder das alterações realizadas ao equipamento CW1

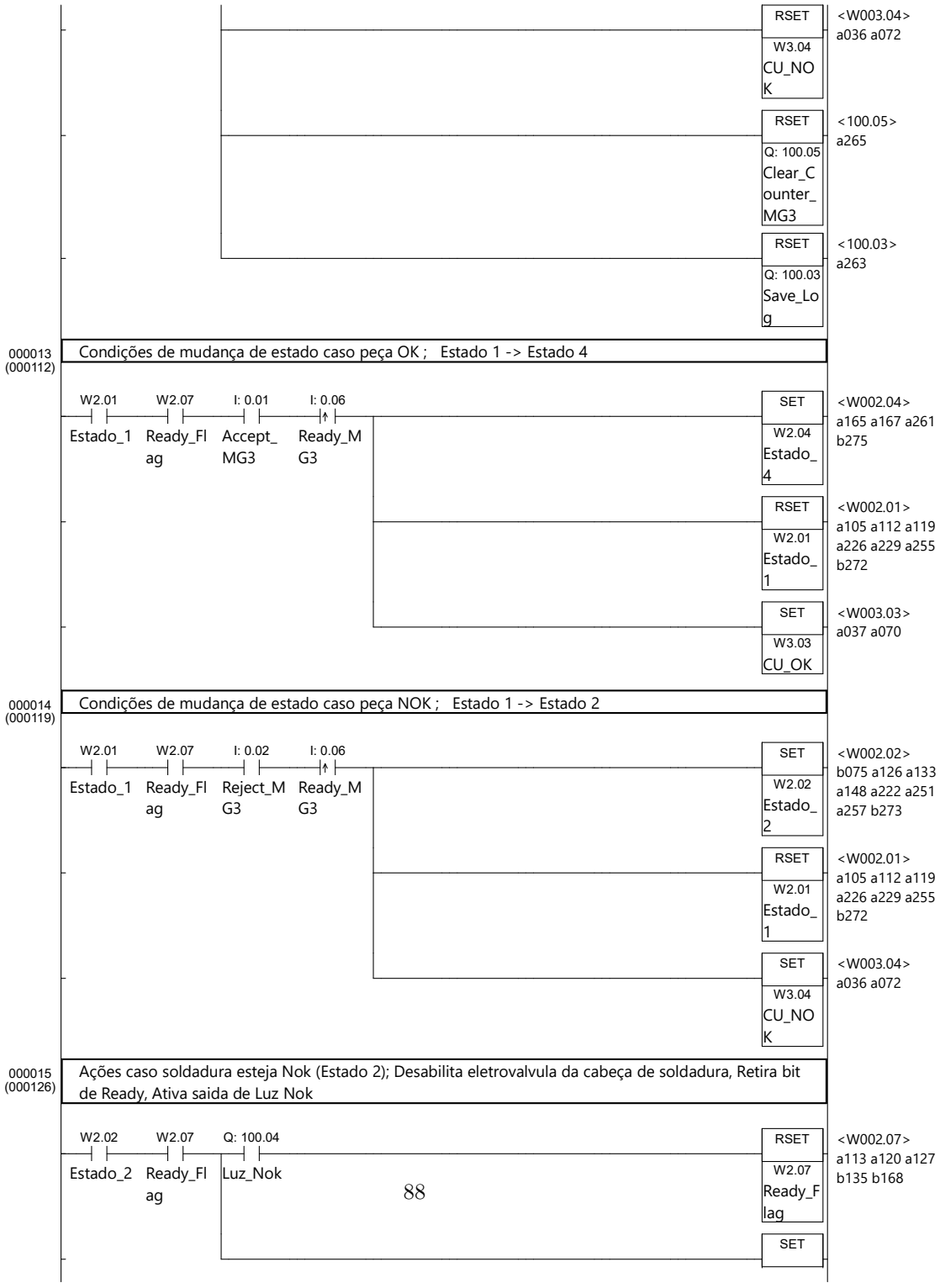

000000
(000042)
Muda programa atraves do HMI, Word 30. Não esquecer que tiramos 2 bits para ter todos os outputs necessarios
W60.10
W30.00
Mudar_Programa
SET
Q: 100.01
bit_0_out
W30.01
SET
Q: 100.02
bit_1_out
W30.02
SET
W1.04
bit_2_programa
W30.03
SET
W1.05
bit_3_programa
SET
70.00
<70.00>
a059
SET
100.06
000001
(000058)
Reset a bits para seleção do programa atraves de timer T040
T040
70.00
RSET
Q: 100.01
bit_0_out
RSET
Q: 100.02
bit_1_out
RSET
Q: 100.03
Save_Log
<100.03>
a263
RSET
Q: 100.04
Luz_Nok
<100.04>
b101 a129 b159
a269
RSET
70.00
<70.00>
a059
RSET
100.06
000002
(000066)
Reset a contadores de Nok e Ok
W0.05
MOV
(021)
[OP1]

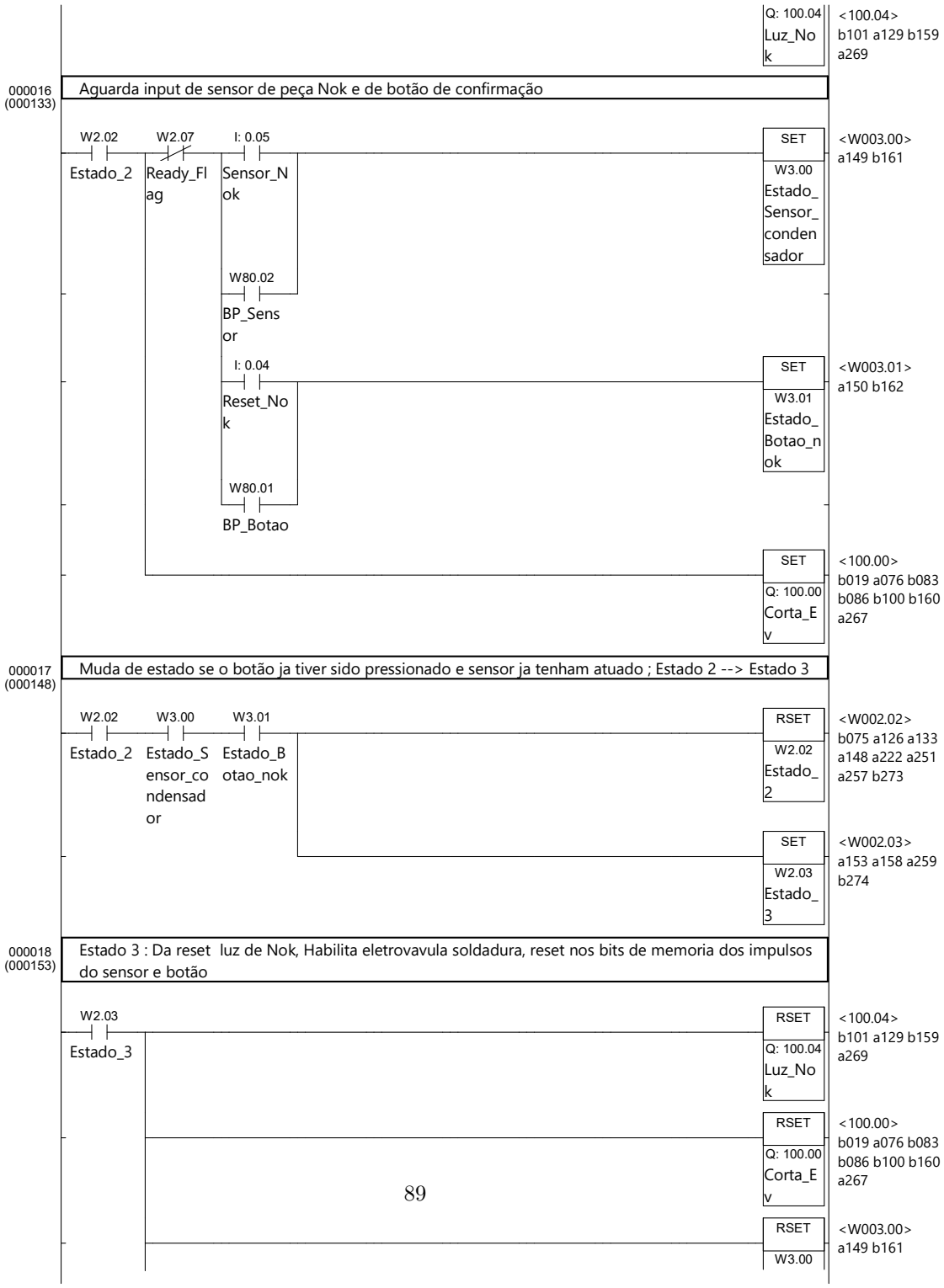

000000
(000042)
Muda programa atraves do HMI, Word 30. Não esquecer que tiramos 2 bits para ter todos os outputs necessarios
W60.10
W30.00
Mudar_Programa
SET
Q: 100.01
bit_0_out
W30.01
SET
Q: 100.02
bit_1_out
W30.02
SET
W1.04
bit_2_programa
W30.03
SET
W1.05
bit_3_programa
SET
70.00
<70.00>
a059
SET
100.06
000001
(000058)
Reset a bits para seleção do programa atraves de timer T040
T040
70.00
RSET
Q: 100.01
bit_0_out
RSET
Q: 100.02
bit_1_out
RSET
Q: 100.03
Save_Log
<100.03>
a263
RSET
Q: 100.04
Luz_Nok
<100.04>
b101 a129 b159
a269
RSET
70.00
<70.00>
a059
RSET
100.06
000002
(000066)
Reset a contadores de Nok e Ok
W0.05
MOV
(021)
[OP1]

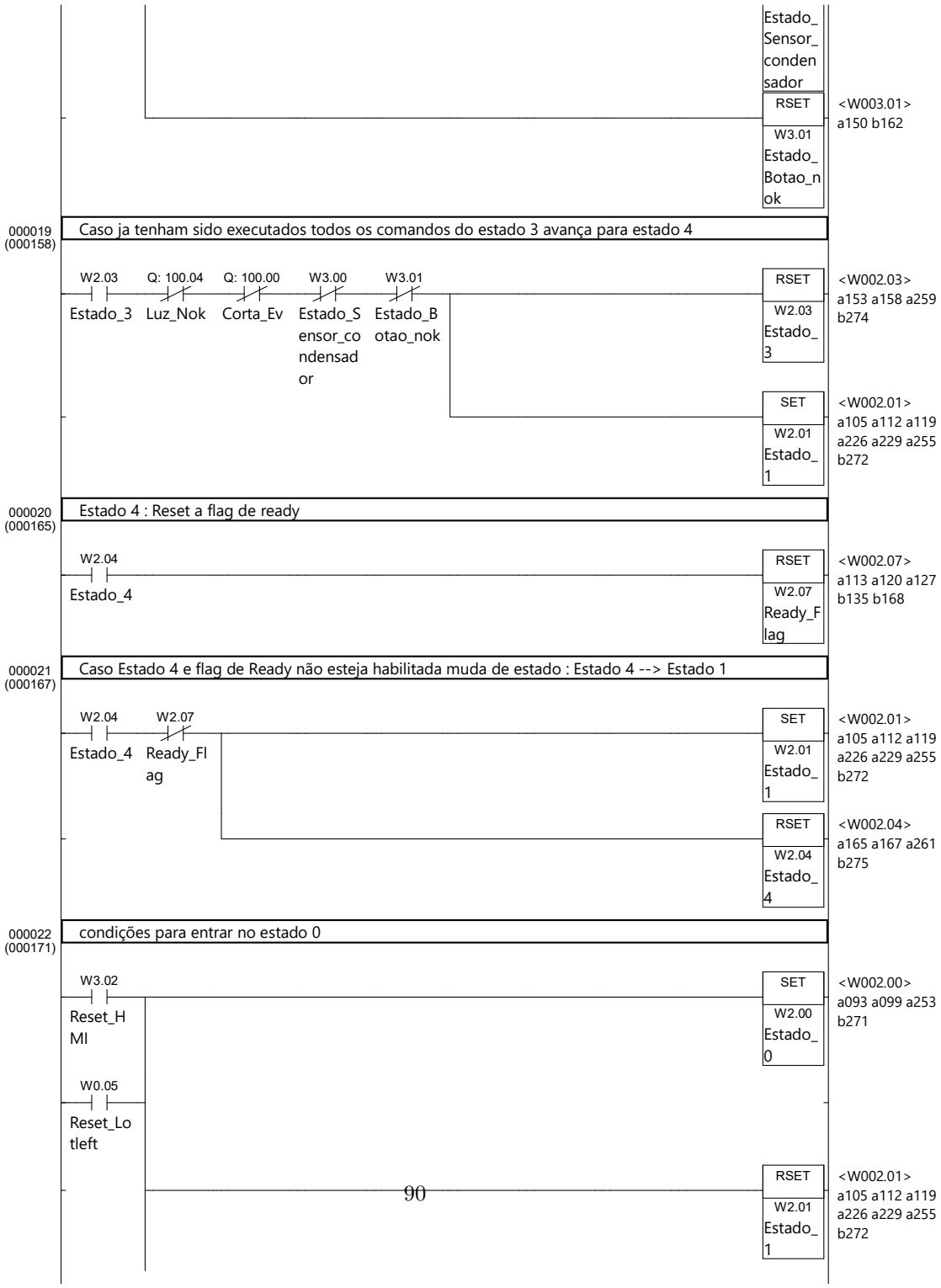

000000
(000042)
Muda programa atraves do HMI, Word 30. Não esquecer que tiramos 2 bits para ter todos os outputs necessarios
W60.10
W30.00
Mudar_Programa
SET
Q: 100.01
bit_0_out
W30.01
SET
Q: 100.02
bit_1_out
W30.02
SET
W1.04
bit_2_programa
W30.03
SET
W1.05
bit_3_programa
SET
70.00
<70.00>
a059
SET
100.06
000001
(000058)
Reset a bits para seleção do programa atraves de timer T040
T040
70.00
RSET
Q: 100.01
bit_0_out
RSET
Q: 100.02
bit_1_out
RSET
Q: 100.03
Save_Log
<100.03>
a263
RSET
Q: 100.04
Luz_Nok
<100.04>
b101 a129 b159
a269
RSET
70.00
<70.00>
a059
RSET
100.06
000002
(000066)
Reset a contadores de Nok e Ok
W0.05
MOV
(021)
[OP1]

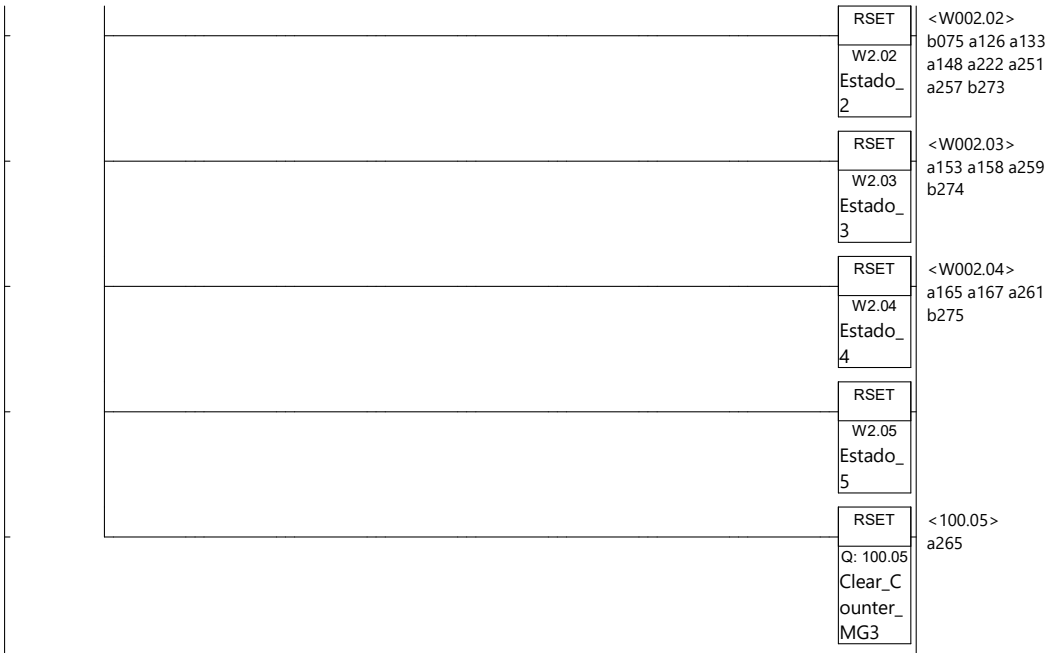

Reset_Lotleft
#0
D10
ok_count
[OP2]
<cD00010>
c071
MOV
(021)
[OP1]
[OP2]
#0
D12
nok_count
<cD00012>
c073
RSET
70.00
<70.00>
a059
000003
(000070)
Counter Up impulso peças OK
W3.03
CU_OK
++
(590)
D10
ok_count
<cD00010>
c067
000004
(000072)
Counter Up impulso peças NOK
W3.04
CU_NOK
++
(590)
D12
nok_count
<cD00012>
c068
000005
(000074)
W3.05
Manual_Outputs
W2.02
Estado_2
Q: 100.00
Corta_Ev
SET
Q: 100.05
Clear_Counter_MG3
<100.05>
a265
SET
70.01
000006
(000079)
Temporizador para reset a bits da seleção do programa
CF113
P_On
W60.10
Mudar_Programa
TIM
040
#10
[OP1]
<T0040(bit)>
a058
[OP2]
000007
(000082)
I: 0.02
Reject_MG3
Q: 100.00
Corta_Ev
SET
W60.12
000008
(000085)
I: 0.01
Accept_MG3
Q: 100.00
Corta_Ev
RSET
W60.12
000009
(000088)
I: 0.04
Reset_Nok
W1.02
Estado_Rejeitado
MOV
(021)
#3
D18
Numero_peça
[OP1]
[OP2]
SET
W1.00

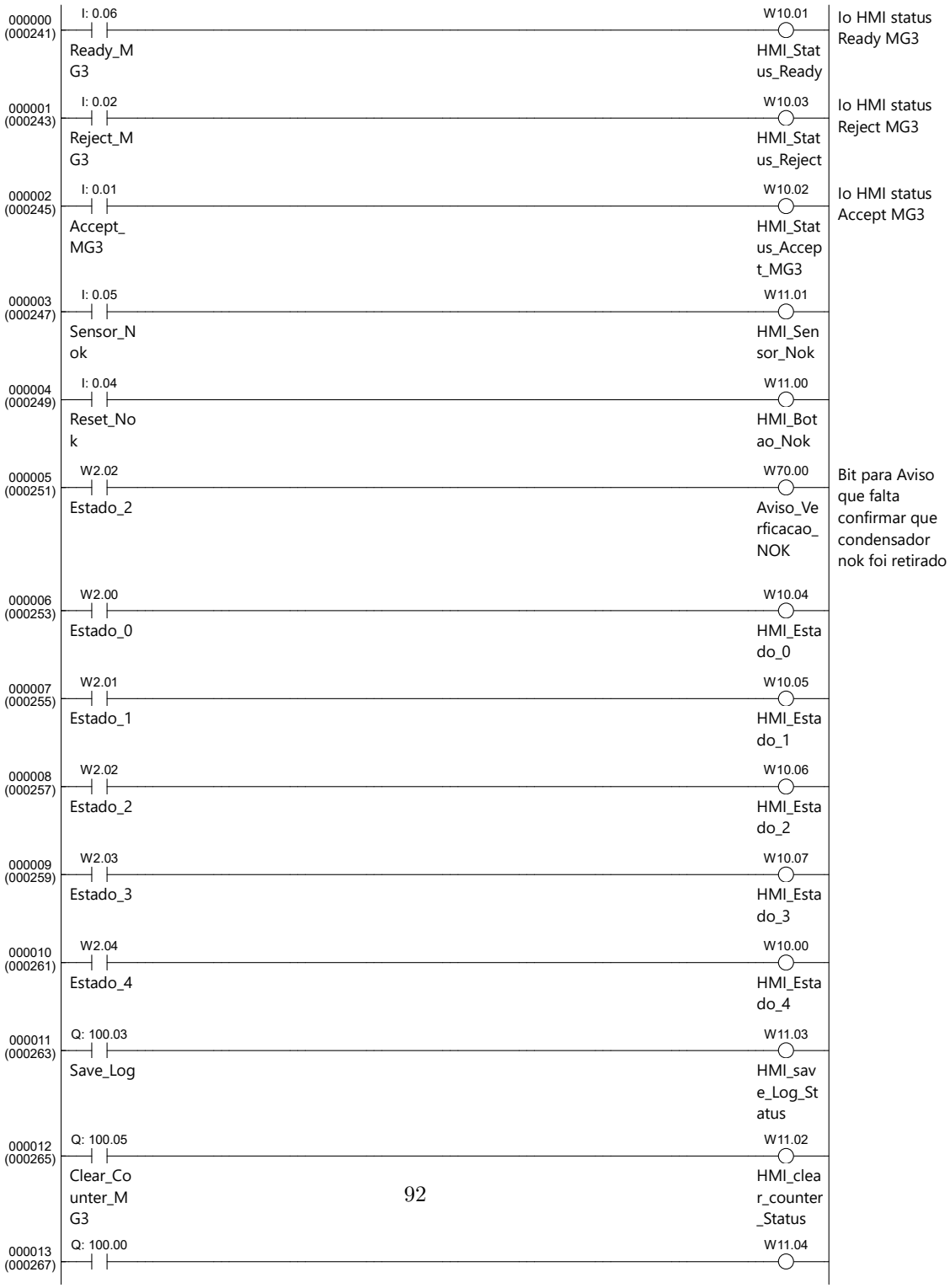

Burst_State
RSET
100.07
<100.07>
b208 a213
000010
(000093)
Estado Inicial (Estado 0) Reset a paragem da maquina e reset a luz Nok
W2.00
Estado_0
RSET
Q: 100.00
Corta_Ev
<100.00>
b019 a076 b083 b086 b100 b160 a267
RSET
Q: 100.04
Luz_Nok
<100.04>
b101 a129 b159 a269
RSET
W2.07
Ready_Flag
<W002.07>
a113 a120 a127 b135 b168
RSET
W3.00
Estado_Sensor_condensador
<W003.00>
a149 b161
RSET
W3.01
Estado_Botao_nok
<W003.01>
a150 b162
000011
(000099)
Condições de mudança de entre Estado 0 e Estado 1
W2.00
Estado_0
Q: 100.00
Corta_Ev
Q: 100.04
Luz_Nok
W2.06
Estado_nok
SET
W2.01
Estado_1
<W002.01>
a105 a112 a119 a226 a229 a255 b272
RSET
W2.00
Estado_0
<W002.00>
a093 a099 a253 b271
000012
(000105)
Estado 1 - Verificar se MG3 esta pronto para nova soldadura
W2.01
Estado_1
I: 0.06
Ready_MG3
SET
W2.07
Ready_Flag
<W002.07>
a113 a120 a127 b135 b168
RSET
W3.03
CU_OK
<W003.03>
a037 a070

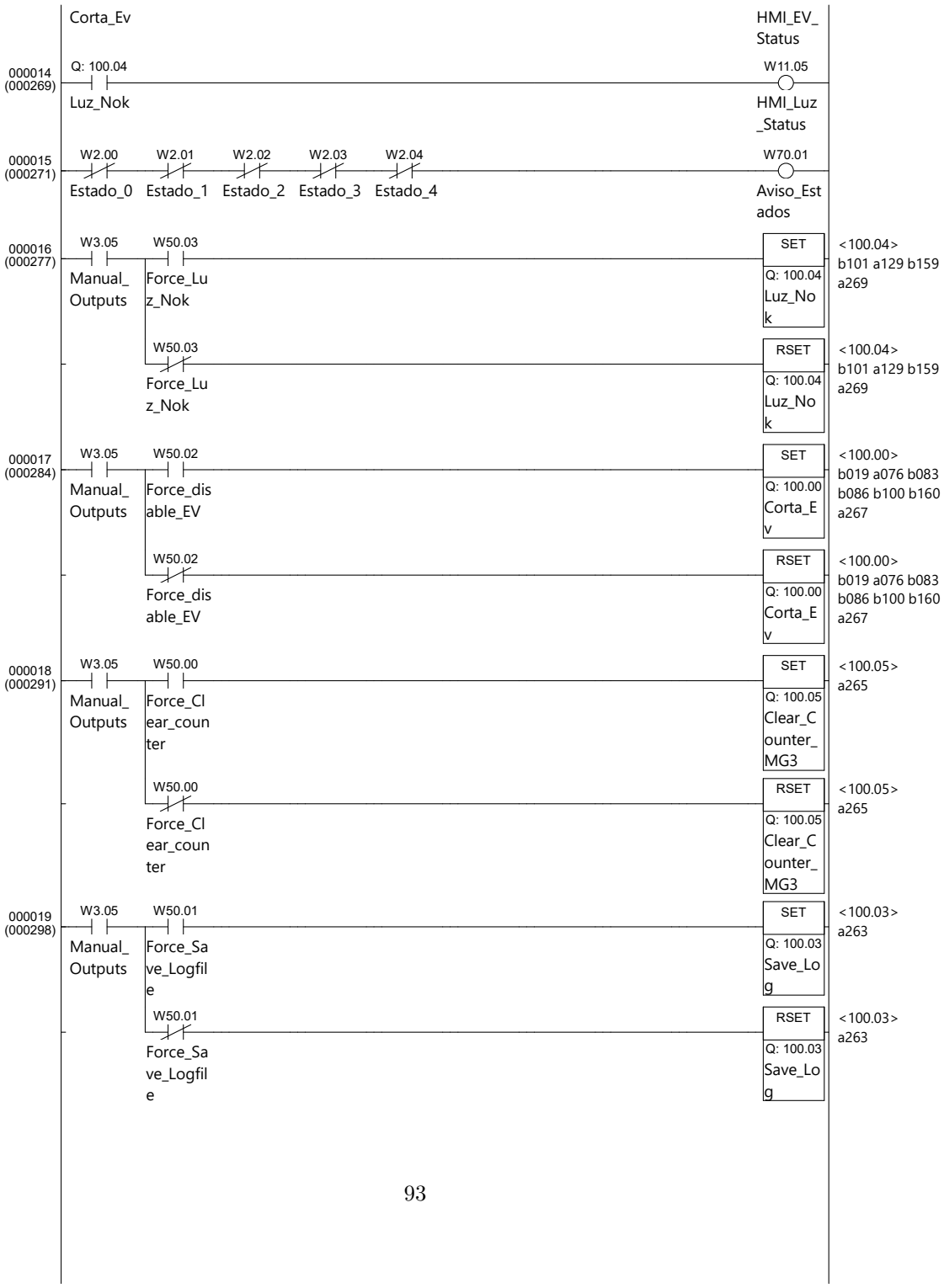

RSET
W3.04
CU_NOK
<W003.04>
a036 a072
RSET
Q: 100.05
Clear_Counter_MG3
<100.05>
a265
RSET
Q: 100.03
Save_Log
<100.03>
a263
000013
(000112)
Condições de mudança de estado caso peça OK ; Estado 1 -> Estado 4
W2.01
Estado_1
W2.07
Ready_Flag
I: 0.01
Accept_MG3
I: 0.06
Ready_MG3
SET
W2.04
Estado_4
<W002.04>
a165 a167 a261 b275
RSET
W2.01
Estado_1
<W002.01>
a105 a112 a119 a226 a229 a255 b272
SET
W3.03
CU_OK
<W003.03>
a037 a070
000014
(000119)
Condições de mudança de estado caso peça NOK ; Estado 1 -> Estado 2
W2.01
Estado_1
W2.07
Ready_Flag
I: 0.02
Reject_MG3
I: 0.06
Ready_MG3
SET
W2.02
Estado_2
<W002.02>
b075 a126 a133 a148 a222 a251 a257 b273
RSET
W2.01
Estado_1
<W002.01>
a105 a112 a119 a226 a229 a255 b272
SET
W3.04
CU_NOK
<W003.04>
a036 a072
000015
(000126)
Ações caso soldadura esteja Nok (Estado 2); Desabilita eletrovalvula da cabeça de soldadura, Retira bit de Ready, Ativa saida de Luz Nok
W2.02
Estado_2
W2.07
Ready_Flag
Q: 100.04
Luz_Nok
RSET
W2.07
Ready_Flag
<W002.07>
a113 a120 a127 b135 b168
SET

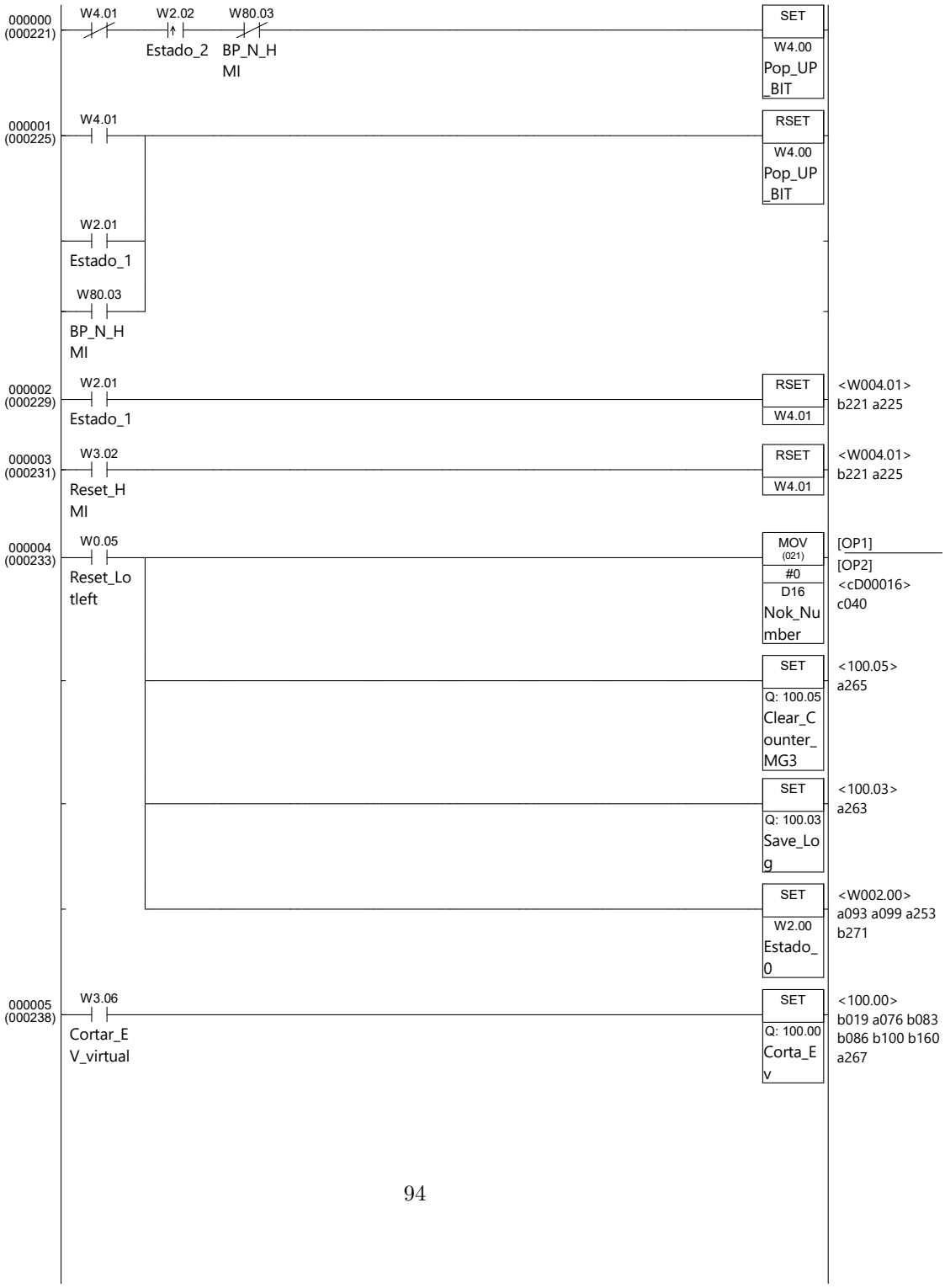

Q: 100.04
Luz_Nok
<100.04>
b101 a129 b159 a269
000016
(000133)
Aguarda input de sensor de peça Nok e de botão de confirmação
W2.02
Estado_2
W2.07
Ready_Flag
I: 0.05
Sensor_Nok
W80.02
BP_Sensor
SET
W3.00
Estado_Sensor_condensador
<W003.00>
a149 b161
I: 0.04
Reset_Nok
W80.01
BP_Botao
SET
W3.01
Estado_Botao_nok
<W003.01>
a150 b162
SET
Q: 100.00
Corta_Ev
<100.00>
b019 a076 b083 b086 b100 b160 a267
000017
(000148)
Muda de estado se o botão ja tiver sido pressionado e sensor ja tenham atuado ; Estado 2 --> Estado 3
W2.02
Estado_2
W3.00
Estado_Sensor_condensador
W3.01
Estado_Botao_nok
RSET
W2.02
Estado_2
<W002.02>
b075 a126 a133 a148 a222 a251 a257 b273
SET
W2.03
Estado_3
<W002.03>
a153 a158 a259 b274
000018
(000153)
Estado 3 : Da reset luz de Nok, Habilita eletrovavula soldadura, reset nos bits de memoria dos impulsos do sensor e botão
W2.03
Estado_3
RSET
Q: 100.04
Luz_Nok
<100.04>
b101 a129 b159 a269
Q: 100.00
Corta_Ev
<100.00>
b019 a076 b083 b086 b100 b160 a267
W3.00
<W003.00>
a149 b161

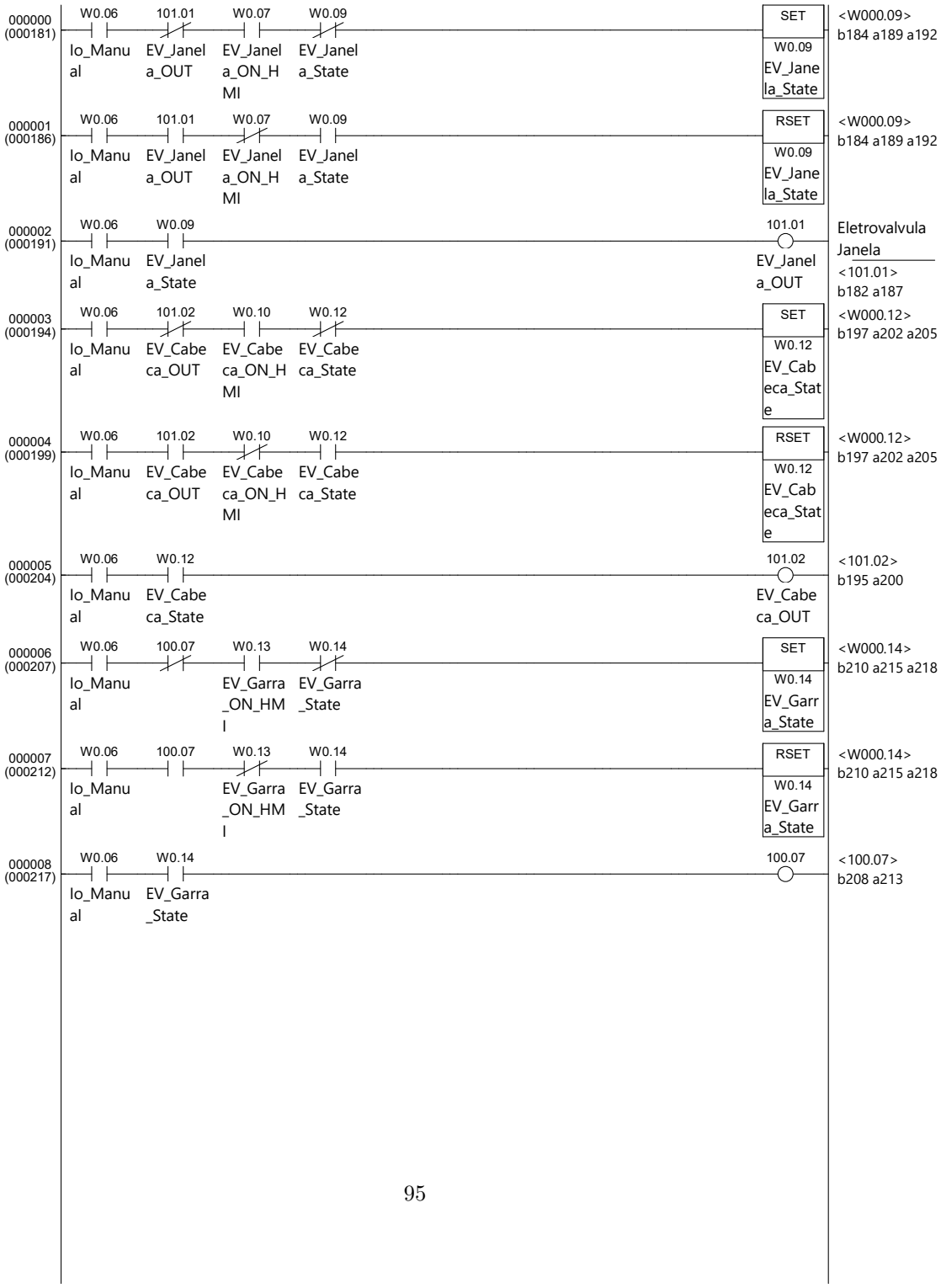

Estado_Sensor_condensador
RSET
W3.01
Estado_Botao_nok
<W003.01>
a150 b162
000019
(000158)
Caso ja tenham sido executados todos os comandos do estado 3 avança para estado 4
W2.03
Estado_3
Q: 100.04
Luz_Nok
Q: 100.00
Corta_Ev
W3.00
Estado_Sensor_condensador
W3.01
Estado_Botao_nok
RSET
W2.03
Estado_3
<W002.03>
a153 a158 a259 b274
SET
W2.01
Estado_1
<W002.01>
a105 a112 a119 a226 a229 a255 b272
000020
(000165)
Estado 4 : Reset a flag de ready
W2.04
Estado_4
RSET
W2.07
Ready_Flag
<W002.07>
a113 a120 a127 b135 b168
000021
(000167)
Caso Estado 4 e flag de Ready não esteja habilitada muda de estado : Estado 4 --> Estado 1
W2.04
Estado_4
W2.07
Ready_Flag
SET
W2.01
Estado_1
<W002.01>
a105 a112 a119 a226 a229 a255 b272
RSET
W2.04
Estado_4
<W002.04>
a165 a167 a261 b275
000022
(000171)
condições para entrar no estado 0
W3.02
Reset_HMI
W0.05
Reset_Lotleft
SET
W2.00
Estado_0
<W002.00>
a093 a099 a253 b271
RSET
W2.01
Estado_1
<W002.01>
a105 a112 a119 a226 a229 a255 b272


RSET
W2.02
Estado_2
<W002.02>
b075 a126 a133 a148 a222 a251 a257 b273
RSET
W2.03
Estado_3
<W002.03>
a153 a158 a259 b274
RSET
W2.04
Estado_4
<W002.04>
a165 a167 a261 b275
RSET
W2.05
Estado_5
RSET
Q: 100.05
Clear_Counter_MG3
<100.05>
a265

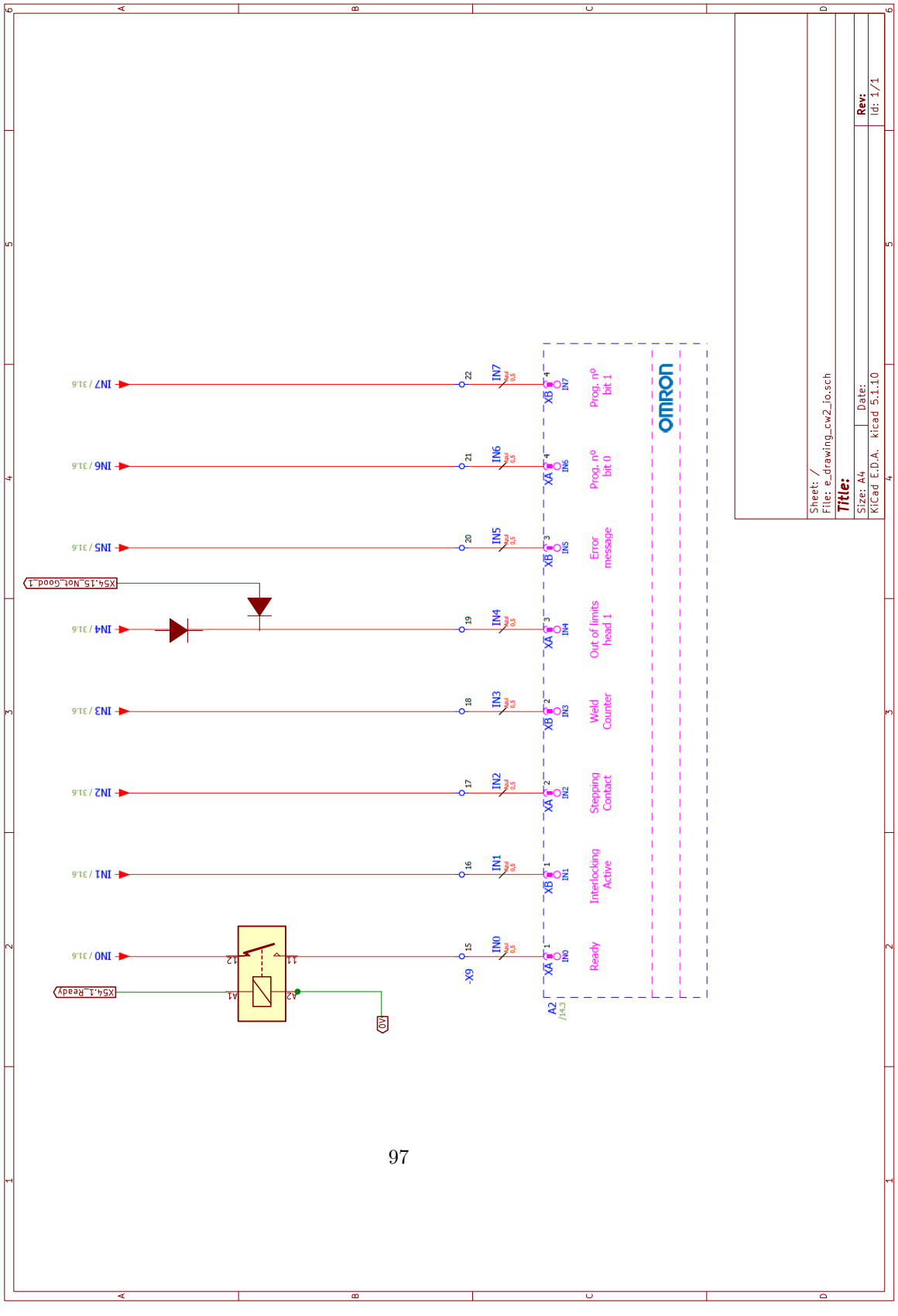

000000
(000241)
I: 0.06
Ready_MG3
W10.01
HMI_Status_Ready
Io HMI status Ready MG3
000001
(000243)
I: 0.02
Reject_MG3
W10.03
HMI_Status_Reject
Io HMI status Reject MG3
000002
(000245)
I: 0.01
Accept_MG3
W10.02
HMI_Status_Accept_MG3
Io HMI status Accept MG3
000003
(000247)
I: 0.05
Sensor_Nok
W11.01
HMI_Sensor_Nok
000004
(000249)
I: 0.04
Reset_Nok
W11.00
HMI_Botao_Nok
000005
(000251)
W2.02
Estado_2
W70.00
Aviso_Verficacao_NOK
Bit para Aviso que falta confirmar que condensador nok foi retirado
000006
(000253)
W2.00
Estado_0
W10.04
HMI_Estado_0
000007
(000255)
W2.01
Estado_1
W10.05
HMI_Estado_1
000008
(000257)
W2.02
Estado_2
W10.06
HMI_Estado_2
000009
(000259)
W2.03
Estado_3
W10.07
HMI_Estado_3
000010
(000261)
W2.04
Estado_4
W10.00
HMI_Estado_4
000011
(000263)
Q: 100.03
Save_Log
W11.03
HMI_save_Log_Status
000012
(000265)
Q: 100.05
Clear_Counter_MG3
W11.02
HMI_clear_counter_Status
000013
(000267)
Q: 100.00
W11.04

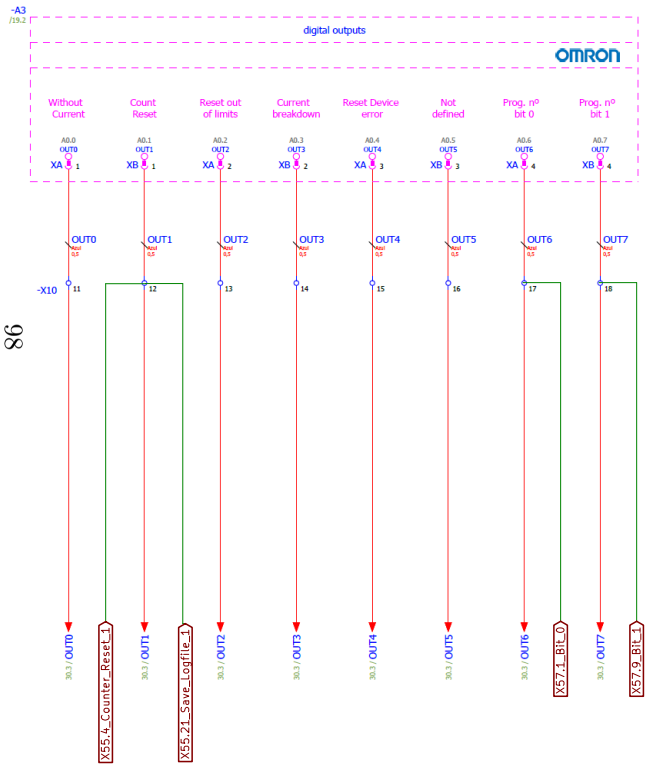

Corta_Ev
HMI_EV_Status
000014 (000269)
Q: 100.04
Luz_Nok
W11.05
HMI_Luz_Status
000015 (000271)
W2.00
Estado_0
W2.01
Estado_1
W2.02
Estado_2
W2.03
Estado_3
W2.04
Estado_4
W70.01
Aviso_Estados
000016 (000277)
W3.05
Manual_Outputs
W50.03
Force_Luz_Nok
SET
Q: 100.04
Luz_Nok
<100.04> b101 a129 b159 a269
W50.03
Force_Luz_Nok
RSET
Q: 100.04
Luz_Nok
<100.04> b101 a129 b159 a269
000017 (000284)
W3.05
Manual_Outputs
W50.02
Force_disable_EV
SET
Q: 100.00
Corta_Ev
<100.00> b019 a076 b083 b086 b100 b160 a267
W50.02
Force_disable_EV
RSET
Q: 100.00
Corta_Ev
<100.00> b019 a076 b083 b086 b100 b160 a267
000018 (000291)
W3.05
Manual_Outputs
W50.00
Force_Clear_counter
SET
Q: 100.05
Clear_Counter_MG3
<100.05> a265
W50.00
Force_Clear_counter
RSET
Q: 100.05
Clear_Counter_MG3
<100.05> a265
000019 (000298)
W3.05
Manual_Outputs
W50.01
Force_Save_Logfile
SET
Q: 100.03
Save_Log
<100.03> a263
W50.01
Force_Save_Logfile
RSET
Q: 100.03
Save_Log
<100.03> a263

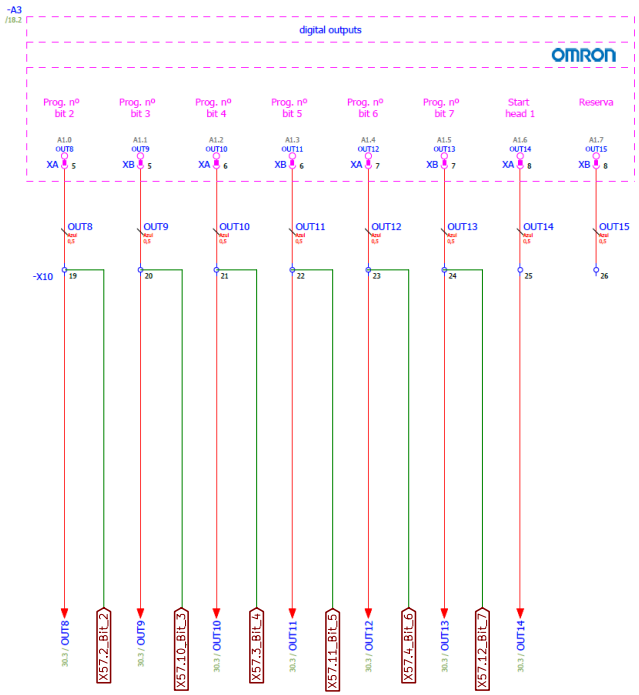

000000
(000221)
W4.01
W2.02
Estado_2
W80.03
BP_N_H
MI
SET
W4.00
Pop_UP
_BIT
000001
(000225)
W4.01
RSET
W4.00
Pop_UP
_BIT
W2.01
Estado_1
W80.03
BP_N_H
MI
000002
(000229)
W2.01
Estado_1
RSET
W4.01
<W004.01>
b221 a225
000003
(000231)
W3.02
Reset_H
MI
RSET
W4.01
<W004.01>
b221 a225
000004
(000233)
W0.05
Reset_Lo
tleft
MOV
(021)
#0
D16
Nok_Nu
mber
[OP1]
[OP2]
<cD00016>
c040
SET
Q: 100.05
Clear_C
ounter_
MG3
<100.05>
a265
SET
Q: 100.03
Save_Lo
g
<100.03>
a263
SET
W2.00
Estado_
0
<W002.00>
a093 a099 a253
b271
000005
(000238)
W3.06
Cortar_E
V_virtual
SET
Q: 100.00
Corta_E
v
<100.00>
b019 a076 b083
b086 b100 b160
a267


000000
(000181)
W0.06
Io_Manual
101.01
EV_Janela_OUT
W0.07
EV_Janela_ON_HMI
W0.09
EV_Janela_State
SET
W0.09
EV_Janela_State
<W000.09>
b184 a189 a192
000001
(000186)
W0.06
Io_Manual
101.01
EV_Janela_OUT
W0.07
EV_Janela_ON_HMI
W0.09
EV_Janela_State
RSET
W0.09
EV_Janela_State
<W000.09>
b184 a189 a192
000002
(000191)
W0.06
Io_Manual
W0.09
EV_Janela_State
101.01
EV_Janela_OUT
Eletrovalvula Janela
<101.01>
b182 a187
000003
(000194)
W0.06
Io_Manual
101.02
EV_Cabeca_OUT
W0.10
EV_Cabeca_ON_HMI
W0.12
EV_Cabeca_State
SET
W0.12
EV_Cabeca_State
<W000.12>
b197 a202 a205
000004
(000199)
W0.06
Io_Manual
101.02
EV_Cabeca_OUT
W0.10
EV_Cabeca_ON_HMI
W0.12
EV_Cabeca_State
RSET
W0.12
EV_Cabeca_State
<W000.12>
b197 a202 a205
000005
(000204)
W0.06
Io_Manual
W0.12
EV_Cabeca_State
101.02
EV_Cabeca_OUT
<101.02>
b195 a200
000006
(000207)
W0.06
Io_Manual
100.07
W0.13
EV_Garra_ON_HMI
W0.14
EV_Garra_State
SET
W0.14
EV_Garra_State
<W000.14>
b210 a215 a218
000007
(000212)
W0.06
Io_Manual
100.07
W0.13
EV_Garra_ON_HMI
W0.14
EV_Garra_State
RSET
W0.14
EV_Garra_State
<W000.14>
b210 a215 a218
000008
(000217)
W0.06
Io_Manual
W0.14
EV_Garra_State
100.07
<100.07>
b208 a213

Appendix D: Esquema elétrico de alterações proposto - CW2


Sheet: /
File: e_drawing_cw2_io.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 5.1.10
Id: 1/1
OMRON
Ready
Interlocking Active
Stepping Contact
Weld Counter
Out of limits head 1
Error message
Prog. n° bit 0
Prog. n° bit 1
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
X54.1_Ready
X54.15_Not_Good_1
0V
A2





86

Alteração do programa do MG3D

Sheet: /
File: e_drawing_cw2.sch

Title: Alterações Crown welding 2

Size: A4	Date:	Rev:
KiCad E.D.A. kicad 5.1.10		Id: 1/1

Appendix E: Função main_f

```
# funco  principal responsavel por criar a pagina principal
def main_f():

    #declarar varaiveis globais em uso
    global root
    global b_update_db
    global machine_list

    #destruir pagina com iamgem de arranque
    splash_root.destroy()

    #criar janela principal do programa atraves do metodo Tk()
    root = Tk()

    #alterar titulo da pagina
    root.title("RW Log Manager")

    #tentar centrar pagina no ecra
    root.eval('tk::PlaceWindow . center')

    #definir tamanho padro da pagina principal
    root.geometry("250x300")#"250x300
    multiprocessing.freeze_support()

    #abrir ficheiro com imagem da empresa
    fp = open("resources\Kemet.png", 'rb')

    #interpretar imagem com biblioteca PIl (pillow)
    img = ImageTk.PhotoImage(Image.open(fp))

    #criar objecto de etiqueta mas com a imagem da empresa e posicionamento no ecra
    label_main_program = Label(root, image = img)
    label_main_program.image = img
    label_main_program.grid(row = 0, column=2,pady=20,columnspan=3)

    #Criar e posicional botao de entrar no pagina para introduzir dados de lote
    b_data_enter = Button(root, text="Introduzir Dados \n Lote ", command=data_enter_click)
```

```
b_data_enter.grid(row=2,column=4,padx=15,pady=10)

#Criar e posicional botao de update  base de dados que abre janela de atualizao de
    dados
b_update_db = Button(root, text="Actualizar \n Base de
    Dados",command=b_update_db_click)
b_update_db.grid(row=2,column=2,padx=15,pady=10)


#Criar e posicional botao de update  base de dados que abre janela de atualizao de
    dados
b_extrair_data = Button(root, text="Extrair da \n Base de Dados", command =
    extract_click)
b_extrair_data.grid(row=4,column=2,padx=15,pady=10)

#Criar e posicional botao de update  base de dados que abre janela de extrair
    relatorios
b_configs = Button(root, text="Calcular", command =
    data_analisis_window,height=2,width=13,borderwidth=3)
b_configs.grid(row=4,column=4,padx=15,pady=10)

# barra de ferramentas
menubar=Menu(root) #criar objecto
root.config(menu=menubar)# configurar objecto

#adicionar menu  barra de ferramentas
file_menu = Menu(menubar,tearoff=False)

#introduzir escolhas possiveis do menu
file_menu.add_command(label='Machine Performance',command=machine_performance_w)
file_menu.add_command(label='Exit',command=root.destroy)

#adicionar menu construido  pagina
menubar.add_cascade(label="Menu",menu=file_menu)

#adicionar menu de equipamentos  barra de ferramentas
equipamento_menu=Menu(menubar,tearoff=False)

#introduzir escolhas e  aces  possiveis no menu
equipamento_menu.add_command(label='Adicionar',command=machine_list_add)
equipamento_menu.add_command(label='Alterar',command=config_change)
equipamento_menu.add_command(label='Listar',command=machine_list_click)

#introduzir menu na pagina
menubar.add_cascade(label="Equipamentos",menu=equipamento_menu)

#iniciar pagina - loop
root.mainloop()
```

Appendix F: Função get_log_file

```
# Funao  que abre ficheiro de registos e extrai o contedo
# Parmetro  = Caminho para o ficheiro de registos: varivel - path
#Abre ficheiro, verifica e formata registos
#Retorna lista formatada dos logs do ficheiro varivel - result

def get_log_file(path):

    result = []

    with open(path, "r") as file:
        #Abrir ficheiro csv atravs do mtodo reader da biblioteca csv
        csvreader = csv.reader(file,delimiter=';')

        for row in csvreader:

            if len(row) > 2 :
                #verificar se linha  o cabealho
                if str(row[0])==r"DATE":
                    pass
                #caso  no  seja
                else:
                    # Verificar se o registo  vlido  atravs  do contador interno
                    if row[2]!='00000000':

                        # Formataes
                        try:
                            #  formatao  do  nmero  do contador interno
                            row[2]=int(row[2].lstrip("0"))

                            #  formatao  da data do registo
                            row[0]=Format_date(row[0])

                            #verifica se a hora inclui milisegundos
                            if len(row[1])>8:
                                row[1]=row[1][:-3]
```

```
                        #Adiciona coluna vazia a registos que tenham falta de um
                            elemento EXT_INFO
                        if len(row) == 23:
                            row.append("")

                        #Adicionar registo  lista de registos
                        result.append(row)

                    # mostrar erro caso ocorra algum erro na formatao dos registo
                    # sem bloquear o programa
                    except:
                        print("Existe algum log incompatvel ",row[2]," | ",row[0]," a
                            ignorar!" )


    return result
```

Appendix G: Função download_data_db

Listing G.1: Codigo da função download_data_db

```
#Funcaoo responsvel por consultar a tabela de um equipamento por datas
#Inputs - equipamento (machine - string) & lista de datas (dates - lista)
#Retorna - lista de todos os itens na base de dados com base nas entradas

def download_db_data(machine,dates):

    #designar que a varivel que contm o caminho para a base de dados  global
    global path_db

    #criar varivel que guarda o objecto da ligao  a base de dados
    conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db")

    #criar varivel com o objecto do cursor na conexo estabelecida
    cursor_db = conn_db.cursor()

    #criar varivel para guardar resultados das consultas
    items=[]

    #ciclo que itera datas e cria consultas e executa-as com base
    for date in dates :

        #criar consulta no formato "SELECT * FROM cw9 WHERE DATE "10-01-2023" "
        query_download_db="SELECT * FROM "+str(machine)+" WHERE DATE = " + "\"" +
            str(date)+ "\""

        #Executar a consulta recorrendo ao cursor criado
        cursor_db.execute(query_download_db)

        #Adicionar os registos obtidos  lista de itens
        items = items + cursor_db.fetchall()

    #terminar a conexo    base de dados aps terminar o ciclo
```

```
    conn_db.close()

    #retornar registros encontrados
    return items
```

Appendix H: Função insert_db

Listing H.1: Codigo da função insert_db

```
# Funo que recebe lista de registos e adiciona base dados consoante mquina
#Input: identificao do equipamento (machine - string) ; data a introduzir na base de
    dados (data - lista de listas)
# No tem retorno

def insert_db(machine,data):

    #Definir caminho para a base de dados como varivel global do programa
    global path_db

    #Definir varivel que guarda objecto de ligao base de dados
    conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db")

    #Criar um objeto cursor sobre a conexo base de dados
    cursor_db = conn_db.cursor()

    #definir varivel temporria para guardar elementos da lista
    lista_para_tuple=[]

    #iterar registos
    for list in data:

        #caso o tamanho da lista seja 23, adicionar uma parcela para ficar similar a
            outros registos
        if len(list)==23 :

            list.append("")
            lista_para_tuple.append(list)

        #caso o tamanho da lista seja 24 diretamente adicionado lista temporaria
        elif len(list)==24:

            lista_para_tuple.append(list)

    # converso de lista de lista para lista de tuplos
```

```
tuple_to_db=[tuple(x)for x in lista_para_tuple]

# construo  da query base
query_insert = "INSERT INTO "+ str(machine).lower() + "
    (DATE,TIME,COUNTER,PROGRAM,Ipk,Upk,POT,R,
s1,s3,F,p,Irms,Q,Urms,Ut,W,tw,twcyc,AY,STATUS,DETAILS,PROG_NAME,EXT_INFO) VALUES
    (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"

#inserir na base de dados todas os tuplos da lista atravs do mtodo insert many
cursor_db.executemany(query_insert, tuple_to_db)

#Compromisso com a base de dados das alteraes feitas
conn_db.commit()

#terminar  ligao     base de dados
conn_db.close()
```

Appendix I: Função b_update_db_click

Listing I.1: Codigo da função b_update_db_click

```
Anexo 4 -  Funo  de controlo da interface grfica da pgina de  atualizao  de dados
#  Funo  que inicia a janela de atualizao dos dados na base de dados
def b_update_db_click():


    #  funo  que toma  ao  quando  pressionado o boto "Update"
    def  db_update_routine():

        #Indicar variveis em uso que so globais
        global strike_limit
        global list_intVar_machine
        global b_update_db

        #definir strike limit com base no input do utilizador, atravs do mtodo .get() da
            biblioteca tkinter
        strike_limit = e_strike.get()

        #criar lista de equipamentos atualizar
        machines =[]

        #ciclo que verifica que equipamentos esto selecionados para atualizao de

        for item in list_intVar_machine:

            print("to update: ", item[0]," yes/or no", item[1].get())

            if item[1].get() == 1:

                #Adicionar  lista se tiver selecionado
                machines.append(item[0])

        #criar uma thread que executa funo de  verificao  de registos
```

```
        thread_associar= threading.Thread(target=update_db_routine_final, args=(machines,))

        #elimina a janela de atualizao da base de dados
        data_update_wi.destroy()

        #Iniciar a thread criada anteriormente
        thread_associar.start()


    # funo que seleciona todos os equipamentos quando pressionado "Selecionar tudo"
    def select_all_click():

        global list_intVar_machine
        global list_check_box

        #utilizar o mtodo select() em cada caixa
        for item in list_check_box:

            item.select()

    # definio das variveis globais em uso
    global data_update_wi
    global machine_list
    global list_intVar_machine
    global b_upt_full #boto em global para dar para alterar nas funes
    global list_check_box

#definir objecto da janela, sendo uma janela secundria utiliza-se o mtodo Toplevel
    data_update_wi = Toplevel()

    #definir ttulo da pgina
    data_update_wi.title("Atualizar bases de dados")
    check_full= IntVar()

    #criar lista para guardar tuplos com o nome do equipamento
    list_intVar_machine = []

    # Criao de uma lista constituda por conjuntos de equipamento e sua varivel de uso no
        objecto.
    #Como estamos a utiliza caixas de confirmao, o resultado s pode ser 1 ou 0 e ento
        usado uma IntVar para guardar esse valor
    for item in machine_list:

        list_intVar_machine.append((item[0],IntVar()))

    # definio de variveis para posicionamento de objetos na janela
    row_i = 2
    column_i=0
    i = 0
    #definir lista para guardar objectos de equipamentos
```

```
    list_check_box = []

    #criar objectos de checkbox e nome do equipamento para cada equipamento na lista
    # Estes so criados no ciclo pois aparecem de forma dinamica, se adicionarmos ou
        removermos equipamentos
    #o programa tem que ser capaz de sempre mostrar como opo as mquinas disponveis
    for item in list_intVar_machine:

        #criar o objeto e definir variveis de interao
        list_check_box.append(Checkbutton(data_update_wi,text=list_intVar_machine[i][0],variable=list_intVar_ma
            onvalue=1, offvalue=0))
        #posicionar objeto na pgina
        list_check_box[-1].grid(row = row_i, column=column_i, padx=5,pady=10)

        # gesto de ndices de posicionamento dos objectos
        if column_i == 4:
            column_i=0
            row_i=row_i+1
        else:
            column_i =column_i+1

        i=i+1

    #Criar objecto de texto onde guardado o cabealho e posicionamento na janela
    Label(data_update_wi,text="Atualizar Bases de Dados\n Selecionadas").grid(row = 0,
        column=2,rowspan=2,pady=10)
    Label(data_update_wi,text="Strike Limit:").grid(row = row_i+1, column=0,pady=4, padx=5)

    #criar entrada para nmero mximo de ficheiros a analisa seguidos sem terminar a
        atualizao
    #ou seja, criar entrada para "strike limit"
    e_strike = Entry(data_update_wi,borderwidth=3,width=10)
    e_strike.grid(row = row_i+1, column=1)

    #inserir um valor padro entrada, 2 no caso
    e_strike.insert(END,2)

    #criar objecto de opo de uso de varivel limite
    Checkbutton(data_update_wi,text="Use strikes ",variable=check_full, onvalue=1,
        offvalue=0).grid(row = row_i+1, column=3)

    #criar e posicionar na janela objecto do boto de "Update" e "Selecionar todos "
    b_upt_full=Button(data_update_wi,text="Update",command=db_update_routine)
b_select_all=Button(data_update_wi,text="Selecionar todos",
command=select_all_click)
#  ao   (command)   chamar a  funo  select_all_click
    b_select_all.grid(row = row_i+1, column=3, padx=5,pady=5)
    b_upt_full.grid(row = row_i+1, column=4, padx=5,pady=5)

    #criar e posicionar na janela objecto do boto de "associar dados"
```

```
Label(data_update_wi,text="__"\*20).grid(row = row_i+2, column=0,columnspan=5)
button_data_association=Button(data_update_wi,text="Associao de Dados de
    Lote",command=associar_dados_click)# ao chama funo "associar_dados_click"
button_data_association.grid(row = row_i+3, column=1,columnspan=3,padx=5,pady=5)
```

Appendix J: Função verify_logs

```
#funcao que verifica registos de soldadura e entradas na base de dados
#cria lista de registos que devem ser adicionados a base de dados
#input - string - equipamento a verificar registos
#verifica se existe registos nos ficheiros a adicionar a base de dados
#se existirem executa  funo  que os insere na base dados
#nao tem retorno
def verify_logs(machine):

    #Declarar varaiveis globais em uso
    global strike_limit
    global machine_list
    global progress_bar_update
    global progress_bar_label

    #Obter caminho para pasta com registoss de soldadura
    for item in machine_list:
        if machine == item[0]:
            path=item[1]


    #Criar lista para guardar caminhos para ficheiros de registos
    list_logs_path=[]

    #ciclo que itera as pesquisas dos conteudos da pasta e os guarda na lista
         list_logs_path
    for folder in os.listdir(path):

        for file_name in os.listdir(path+ "\\" + str(folder)):
            list_logs_path.append(path+ "\\" + str(folder)+ "\\" + str(file_name))

    #inverter lista obtida para que ultima entrada passe para a primeira
    list_logs_path.reverse()

    #Definir variavel de limite
    strike=0
```

```
#Definir variavel para guardar numero de registos adicionados para mostrar ao
    utilizador
logs_print=0
#all_logs = []

#Criar o objeto na janela principal da barra de progresso
progress_bar_update =
    ttk.Progressbar(root,orient=HORIZONTAL,length=200,mode="determinate")

# Definir posicionamento da barra de progresso na janela principal
progress_bar_update.grid(row=6,column=2,padx=15,pady=2,columnspan=4)

#tentar destruir etiqueta previamente existente na pagina principa
try:
    progress_bar_label.destroy()
except:
    pass

#definir valor inicial para barra de progresso
progress_bar_update["value"]=0

#criar objecto com etiqueta na janela principal
progress_bar_label = Label(root,text="A verificar logs na base de dados")

#posicionar a etiqueta de texto na janela principal
progress_bar_label.grid(row=5,column=2,padx=15,pady=10,columnspan=4)
n_items_adicionados=0

#guardar variavel com tamanho da lista para realizar calculos
tamanho_list=len(list_logs_path)

#ciclo de  verificao  de ficheiros na lista de ficheiros
for log_file in list_logs_path:

    # Alteraes  iterativas na interface #
    logs_print=logs_print+1

    #Calculo do progresso feito
    progress_bar_update["value"]=(logs_print/tamanho_list)*100
    progress_bar_label["text"]="Log files verificados "+str(logs_print)+" de
        "+str(tamanho_list) +" | "+str(round((logs_print/tamanho_list)*100,0))+" % \n
        "+str(n_items_adicionados)+ " logs adicionados | "+str(machine).upper()
    print("Ficheiro de logs verificados",logs_print," de ",tamanho_list," no total
        para equipamento",machine)

    #Criar variavel para guardar lista de datas presentes no ficheiro log
    dates =[]

    logs = get_log_file(log_file)
```

```
print("tamanho do ficheiro de logs ", len(logs) , " logs no logfile",log_file)

#ciclo que verifica datas presentes num ficheiro de registos
for log in logs:
    #caso  no  tenha a data na lista deve adicionar
    if log[0] not in dates:
        dates.append(log[0])

#Ciclo que verifica as datas obtidas
if len(dates)>0 :

    #executar  funo  que obtem regitos da base de dados para uma determinada data e
        guardar resuktadis
    db_data =download_db_data(machine,dates)

    #print de debug a informar as dataas a verificar, o equipamento, os registos
        encontrados no ficheiro
    print("Datas a verificar na base de dados: ",dates, " equipamento", machine)
    print("Tamanho da BD : ",len(db_data)," |Tamanho do ficheiro com LOGS :
        ",len(logs), " | Referente s datas ",dates, "para a maquina" ,machine)


    # Caso o tamanho da lista de registos da base de dados seja igual ao tamanho da
        lista do ficheiro para estas datas
    #Considera-se que estes registos ja esto contidos na base de dados
    if len(db_data) == len(logs):

        #incrmentar varivel limite
        strike=strike +1

        #print de debug a informar que os registos ja pertecem  base de dados
        print("Option - Todos os ",len(logs)," logs ja estavem presentes na base de
            dados ")


    #Caso o numero de registos na base de dados seja diferente do numero de
        registos no ficheiro
    #    necessario compaarar os dois conjuntos
    elif len(db_data) != len(logs) and len(db_data)>0:

        #criar lista para guardar eventuais registos a adicionar  base de dados
        data_update_db=[]

        #variavel que vai guardar a data e hora de cada registo na base de dados
            para esta data.
        dates_db=[]

        #ciclo de obtencao da data e hora de cada registo
        for data in db_data:
```

```
    dates_db.append((data[7],data[8]))

#oprint de debug a informar quantos registos vai ser necessario verificar
print("Tamanho da lista de logs comparar ", len(dates_db)," para a maquina
    ",machine)

#ciclo que vai comparar cada registo no ficheiro com os contidos na base de
    dados
for item in logs:

    #caso o conjunto data e hora de um registo do ficheiro esteja contido na
        lista de conjuntos de data e hora da base de dados
    # consideramos que este ja esta na base de dados.
    # assim   removido da lista de forma a acelerar a verificao
    if (item[0],item[1]) in dates_db:
        dates_db.remove((item[0],item[1]))

    #caso  no  esteja presente na lista da base de dados significa que
        necessario adicionar a base de dados
    else:
        #adicionamos o registo  lista a adicionar a base de dados
        data_update_db.append(item)

#caso a nao eencontre registos no ficheiro necessarios adicionar a na base
    de dados
if len(data_update_db)==0:

    #   incrmentada a varivel de limite
    strike=strike +1

#caso existam tenham sido encontrados registos a adicionar  base de dados
else:

    #limpar variavel de limite
    strike = 0

    #print de debug com  informao  do numero de registos a adicionar
    print("inserir :", len(data_update_db), " logs")

    #executar funcao que adiciona conjuntos de registos  base dados
    insert_db(machine,data_update_db)

    #incrementar variavel que regista numero de registos adicionados  base
        de dados
    n_items_adicionados=n_items_adicionados+len(data_update_db)


#print de debug a infar registos ja adicionados  base de dados
print("Numero de logs a adicionar  base de dados ",len(data_update_db), "
    do equipamento ", machine," e n strikes: ", strike,"\n")
```

```
            #caso quando seja consutada a base de dados no sejam retornados registos nenhums
            #considera-se que  necessario inserir tudo o que esta no ficheiro de registo na
                base de dados
            elif len(db_data)==0:

                #limpar variavel de limite
                strike=0

                #incrementar variavel que regista numero de registos adicionados  base de
                    dados
                n_items_adicionados=n_items_adicionados+len(logs)

                #executar funcao que adiciona conjuntos de registos  base dados
                insert_db(machine,logs)

        #caso o numero da variavel limite chegue ao valor maximo definido
        if strike == int(strike_limit):

            #print de debug a informar fim da atualizao
            print("fim update  maquina ", machine, " devido ao limite de strikes\n " )

            #introduzir  informao  na etiqueta a colocar na janela principal
            progress_bar_label["text"]="Log files verificados "+str(logs_print)+" de
                "+str(tamanho_list) +" | "+str(round((logs_print/tamanho_list)*100,0))+"
                %\n"+str(strike)+" em "+str(strike_limit)+ " strikes |
                "+str(n_items_adicionados)+ " logs adicionados \n no equipamento
                "+str(machine).upper()

            #interromper ciclo de  verificao  de ficheiros de registos
            break

    #Destruir barra de progresso caso verificao termine
    try:
        progress_bar_update.destroy()
    except:
        pass
```

Appendix K: Função data_enter_click

Listing K.1: Codigo da função data_enter_click

```
#funcao responsavel por gerar janela e objectos de introduo de dados relativos a produo
#contem as  funes  das  aces  dos botoes contidos na janela
#
def data_enter_click():

    #Gerar uma janela de Top level, secundaria  main
    data_enter_wi = Toplevel()
    #alterar titulo da janela
    data_enter_wi.title("Introduzir dados")

    #indicar o uso de variaveis globais
    global e_nlote
    global machine_list
    global clicked_maq_drop

    # definio  de variavel que vai guardar lista de equipamentos
    lista_maquinas=[]

    # definio  de variavel que vai guardar lista de equipamentos para utilizar em menu
        colapsavel
    lista_maquinas_drop=[]

    #ciclo para atribrir equipamentos as slistas
    for items in machine_list:
        lista_maquinas.append(items[0])
        lista_maquinas_drop.append(items[0].upper())

    #criar variavel do tkinter para guardar variavel string
    clicked_maq_drop = StringVar()
    #definir item padro da lista
    clicked_maq_drop.set(lista_maquinas_drop[0])
    #criar lista colapsavel
```

```
maq_drop=OptionMenu(data_enter_wi, clicked_maq_drop,*lista_maquinas_drop ).grid(row =
    1, column=2,pady=4, padx=5)


#objecto e posicionamento de etiqueta
Label(data_enter_wi,text="Maquina:").grid(row = 1, column=1,pady=4, padx=5)

#Criar objecto e etiqueta da entrada N de lote e posicionamento na janela
Label(data_enter_wi,text="N Lote:").grid(row = 2, column=1,pady=4, padx=5)
e_nlote = Entry(data_enter_wi,borderwidth=2,width=25)
e_nlote.grid(row = 2, column=2,pady=4, padx=5)

#Criar objecto e etiqueta da entrada Descrio e posicionamento na janela
Label(data_enter_wi,text="Descrio:").grid(row = 3, column=1,pady=4, padx=5)
e_descritption = Entry(data_enter_wi,borderwidth=2,width=25)
e_descritption.grid(row = 3, column=2,pady=4, padx=5)

#Criar objecto e etiqueta da entrada Part number e posicionamento na janela
Label(data_enter_wi,text="Part Number:").grid(row = 4, column=1,pady=4, padx=5)
e_partnum = Entry(data_enter_wi,borderwidth=2,width=25)
e_partnum.grid(row = 4, column=2,pady=4, padx=5)

#Criar objecto e etiqueta da entrada setpoint do controlador e posicionamento na janela
Label(data_enter_wi,text="Setpoint Corrente [kA]:").grid(row = 5, column=1,pady=4,
    padx=5)
e_setpoint = Entry(data_enter_wi,borderwidth=2,width=25)
e_setpoint.grid(row = 5, column=2,pady=4, padx=5)

#Criar objecto e etiqueta da entrada item da coroa e posicionamento na janela
Label(data_enter_wi,text="Item Coroa:").grid(row = 1, column=3,pady=4, padx=5)
e_crownitem = Entry(data_enter_wi,borderwidth=2,width=25)
e_crownitem.grid(row = 1, column=4,pady=4, padx=5)

#Criar objecto e etiqueta da entrada numero de lote da coroa e posicionamento na janela
Label(data_enter_wi,text="Lote da Coroa:").grid(row = 2, column=3,pady=4, padx=5)
e_crownlot = Entry(data_enter_wi,borderwidth=2,width=25)
e_crownlot.grid(row = 2, column=4,pady=4, padx=5)

#Criar objecto e etiqueta da entrada referencia do eletrodo e posicionamento na janela
Label(data_enter_wi,text="Referencia Eletrodo").grid(row = 3, column=3,pady=4, padx=5)
e_eletrode = Entry(data_enter_wi,borderwidth=3,width=25)
e_eletrode.grid(row = 3, column=4,pady=4, padx=5)

#Criar objecto do boto "Iniciar lote" - aco quando pressionado: executa funcao
    start_lot
b_start_lot=Button(data_enter_wi, text="Iniciar Lote",command=start_lot )
b_start_lot.grid(row = 6, column=3,pady=4, padx=5)

#Criar objecto do boto "Terminar Lote" - aco quando pressionado: executa funcao
    end_lot
```

```
    b_end_lot=Button(data_enter_wi, text="Terminar Lote ",command=end_lot,state=
        'disabled')
    b_end_lot.grid(row = 6, column=4,pady=4, padx=5)


    #Criar objecto do boto "Limpar dados" - aco quando pressionado: executa funcao
        clear_fields_data_enter
    b_clear_data=Button(data_enter_wi, text="Limpar dados",
        command=clear_fields_data_enter)
    b_clear_data.grid(row = 6, column=2)

    #criar um menu na barra de ferramentas
    menubar = Menu(data_enter_wi)
    data_enter_wi['menu']= menubar

    #criar um submenu
    subMenu = Menu(menubar)

    #adicionar opes ao menu Adicionar valor de fora de trao , executa funo
    menubar.add_command(label="Adicionar valor Pull Force",command=add_pull_force_value)

    #verificar se existem lotes por terminar
    if os.listdir(path_app + "\\" +"tmp" + "\\") !=[]:

            #criar boto existam logs por terminar
            b_continue_lot=Button(data_enter_wi, text="Continuar
                Lote",command=continue_lote)
            b_continue_lot.grid(row = 6, column=1)
```

Listing K.2: Codigo da função start_lot

```
#Funcao que guarda variaveis introduzidas nos campos em lista
    # iniciada quando o boto "Iniciar lote" pressionado
    def start_lot():

        #declarar variaveis globais em uso
        global init_lot_time
        global in_prod_lot_label
        global in_prod_label

        #print de debug a indicar a hora guardada
        print("Lot started at:",datetime.now())

        #desabilitar o botao "Iniciar lote"
        b_start_lot['state']='disabled'

        #habilitar boto de terminar o lot
        b_end_lot['state']='active'
```

```
# tenta disabilitar o boto de continuar lote, este pode no existir
#caso  no  hajam registos anteriores por isso o uso de try;except
try:
    b_continue_lot['state']='disabled'
except:
    pass
b_clear_data['state']='disabled'

#guardar hora atual em variavel
init_lot_time= datetime.now()

#criar objecto de texto indicando que um lote foi iniciado, posicionamento no ecra
     com metodo .grid()
in_prod_label=Label(data_enter_wi,text="!! EM PRODUO !!")
in_prod_label.grid(row = 4, column=4,pady=4, padx=5)
N_lote_label="LOTE: " + str(e_nlote.get())
in_prod_lot_label=Label(data_enter_wi,text=N_lote_label)
in_prod_lot_label.grid(row = 5, column=4,pady=4, padx=5)

#guardar conteudo inserido em lista.
#obtem os dados introduzidos nos campos atraves do metodo get sobre as variaveis
#dos correspondentes objectos
date=[
    clicked_maq_drop.get(),
    str(init_lot_time),
    str("end time"),
    e_nlote.get(),
    e_descritption.get(),
    e_partnum.get(),
    e_setpoint.get(),
    e_crownitem.get(),
    e_crownlot.get(),
    e_eletrode.get(),

]

#guardar caminho para ficheiro em variavel
caminho = path_app + "\\" +"tmp" + "\\" + str(e_nlote.get())+ ".txt"

#guardar em ficheiro csv, ao utilizar o metodo open do python, com os argumentos
#caminho a indicar o nome do ficheiro e onde deve ser guardado, em string
# w indica o metodo de abertura do ficheiro
# newline diz para no terminar o ficheiro com enter "\n" e ser apenas de uma linha
     o ficheiro utilizando ""
with open(caminho, "w", newline="") as file:

    csvwrite = csv.writer(file,delimiter=",")

    csvwrite.writerow(date)
```

```
    #desabilitar entradas de texto
    e_crownitem.config(state='disabled')
    e_crownlot.config(state='disabled')
    e_descritption.config(state='disabled')
    e_nlote.config(state='disabled')
    e_partnum.config(state='disabled')
    e_setpoint.config(state='disabled')
    e_eletrode.config(state='disabled')
```

Listing K.3: Codigo da função end_lot

```
#Funcao terminar o lote e guardar os dados na base de dados
#funcao   executada ao pressionar boto "terminar lote"
def end_lot():

    #declarar variaveis globais em uso
    global threadDB
    global in_prod_lot_label
    global init_lot_time
    global in_prod_label

    #print de debug a indicar hora registada
    print("Lot ended at:",datetime.now())

    #verifica se base de dadosesta a ser acessada atraves do metodo is_alive da
        biblioteca threading
    if threadDB.is_alive():
        #mensagem de erro
        messagebox.showinfo("Alerta","A Aguardar disponibilidade acesso a base de
            dados")

        #esperar que thread acabe atraves do metodo join
        threadDB.join()

    #caso o acesso  base de dados esteja disponivel
    else:

        #actvar boto de iniciar lote
        b_start_lot['state']='active'

        #desabilitar boto de terminar lote
        b_end_lot['state']='disabled'

        #tenta habilitar o boto de continuar lote caso este exista
        try:
            b_continue_lot['state']='active'
        except:
```

```
            pass

        #habilita boto de limpar campos de introdo de dados
        b_clear_data['state']='active'

        #guardar hora de fim em variavel
        final_lot_time = datetime.now()

        #destroi indicaes na interface sobre lotes em produo
        in_prod_label.destroy() # Elimina labels quando o programa tirado do modo de
            produo
        in_prod_lot_label.destroy()
        e_crownitem.config(state='normal')
        e_crownlot.config(state='normal')
        e_descritption.config(state='normal')
        e_nlote.config(state='normal')
        e_partnum.config(state='normal')
        e_setpoint.config(state='normal')
        e_eletrode.config(state='normal')

    #guardar conteudo inserido em lista.
    #obtem os dados introduzidos nos campos atraves do metodo get sobre as variaveis
    #dos correspondentes objectos
        date=[
            clicked_maq_drop.get(),
            str(init_lot_time)[:-7],
            str(final_lot_time)[:-7],
            e_nlote.get(),
            e_descritption.get(),
            e_partnum.get(),
            e_setpoint.get(),
            e_crownitem.get(),
            e_crownlot.get(),
            e_eletrode.get(),
        ]

        try:
            #executa funo insert_time_db para introduzir na base de dados os detalhes
                da produo
            insert_time_db(date,path_db)

            #remove ficheiro da pasta com metodo remove da biblioteca os
            os.remove(path_app+"\\"+"tmp"+"\\"+str(e_nlote.get())+".txt")

        #mensagem de erro caso introduo na base de dados no tenha sucesso
        except Exception as e:
            messagebox.showerror("Erro", e)
```

Listing K.4: Codigo da função continue_lot

```
#Funcao para recuperar dados guardados em ficheiro
#funcao   executada ao pressionar boto "continuar lote"
def continue_lote():

    #declarar variaveis globais em uso
    global init_lot_time
    global in_prod_lot_label
    global in_prod_label

    #obter caminho para ficheiro atraves de explorador de ficheiros do windows
    file = filedialog.askopenfilename(initialdir=path_app+"\\"+"tmp",title =
        "Selecionar ficheiro do lote")

    #com o ficheiro
    try:

        #utilizar funcao path config para obter detalhes do ficheiro
        dados_recuperados = path_config(file)[0]

        #atribuir data de inicio atrves do dos dados obtidos
        init_lot_time = dados_recuperados[1]

        #apagar o que estiver nas entradas de texto
        e_crownitem.delete(0, END)
        e_crownlot.delete(0, END)
        e_descritption.delete(0, END)
        e_nlote.delete(0, END)
        e_partnum.delete(0, END)
        e_setpoint.delete(0, END)
        e_eletrode.delete(0, END)

        #inserir texto nas entradas conforme recuperado
        e_crownitem.insert(0, dados_recuperados[7])
        e_crownlot.insert(0, dados_recuperados[8])
        e_descritption.insert(0, dados_recuperados[4])
        e_nlote.insert(0, dados_recuperados[3])
        e_partnum.insert(0, dados_recuperados[5])
        e_setpoint.insert(0, dados_recuperados[6])
        e_eletrode.insert(0, dados_recuperados[9])

        #desabilitar entradas de texto
        e_crownitem.config(state='disabled')
        e_crownlot.config(state='disabled')
        e_descritption.config(state='disabled')
        e_nlote.config(state='disabled')
        e_partnum.config(state='disabled')
        e_setpoint.config(state='disabled')
        e_eletrode.config(state='disabled')
```

```
            #desativar boto de inicio de lote
            b_start_lot['state']='disabled'

            #activar boto de terminar lote
            b_end_lot['state']='active'

            #tenta desabilitar o boto de continuar lote caso este exista
            try:
                b_continue_lot['state']='disabled'
            except:
                pass

            #desabilitar boto de limpar os dados
            b_clear_data['state']='disabled'

            #definir  opo  de menu colapsavel conforme dados recuperados
            clicked_maq_drop.set(dados_recuperados[0])

            #criar texto indicando que esta em produo e numero de lote associado
            in_prod_label=Label(data_enter_wi,text="!! EM PRODUO !!")
            in_prod_label.grid(row = 4, column=4,pady=4, padx=5)
            N_lote_label="LOTE: " + str(e_nlote.get())
            in_prod_lot_label=Label(data_enter_wi,text=N_lote_label)
            in_prod_lot_label.grid(row = 5, column=4,pady=4, padx=5)

        #caso acontea algum erro no processo anterior
        except Exception as e:
            messagebox.showerror("Erro", e)
        pass
```

Listing K.5: Codigo da função clear_fields_data_enter

```
#funcao que limpa os campos quando se carrega no botao para "limpar dados"
   def clear_fields_data_enter():

        #utiliza o metodo delete sobre o objecto de entrada de texto criada
        e_crownitem.delete(0, END)
        e_crownlot.delete(0, END)
        e_descritption.delete(0, END)
        e_nlote.delete(0, END)
        e_partnum.delete(0, END)
        e_setpoint.delete(0, END)
        e_eletrode.delete(0, END)
```

Appendix L: Função insert_time_db

```
# Funo  que insere dados na base de dados na tabela de dados para atualizar registos de
    soldadura
# inputs: data - dados a adicionar na base de dados (lista)| [machine,data inicial,data
    final ,Numero Lote ,Descrition,PartNum,SetPoint,ItemCrown,LotCrown,RefEletrode]
#       path_db - caminho para base de dados (string)
# No tem retorno
def insert_time_db(data,path_db):

    #iniciar conexo a base de dados
    conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db")

    #criar um cursor sobre essa conexo
    cursor_db = conn_db.cursor()

    #transformar dados em formato de tuplo para consulta
    tuple_db_data=tuple(data)

    #construir consulta e executar com mtodo execute do sqlite , parmetros (string
        consulta,tuplo dados)
    cursor_db.execute("INSERT INTO updatedata
        (machine,datai,dataf,NLot,Descrition,PartNum,SetPoint,ItemCrown,LotCrown,RefEletrode)
        VALUES (?,?,?,?,?,?,?,?,?,?)", tuple_db_data)

    #compromisso das alteraes
    conn_db.commit()

    #print de debug a informar que dados foram adicionados tabela
    print("Adicionado tabela para updatedata: ",tuple_db_data)

    #terminar ligao com base de dados
    conn_db.close()
```

Appendix M: Função add_pull_force_value

Listing M.1: Codigo da função add_pull_force_value

```
#Funcao responsavel por abrir janela para introduizr dados relativos a testes destrutivos
    de resistencia  trao
#Esta  funo     executada quando  pressionada a  opo  "inserir dados de pull force"
    dentro da janela de "introduzir dados de lote "
def add_pull_force_value():

    #Declarar uso de variaveis globais
    global e_nlote
    global machine_list
    global clicked_maq_drop

    #definir variaveis para guardar lista de equipamentos
    lista_maquinas=[]
    lista_maquinas_drop=[]

    #introduzir equipamentos na lista
    for items in machine_list:
        lista_maquinas.append(items[0])
        lista_maquinas_drop.append(items[0].upper())

    #criar a janela de  introduo  de dados de esforo de   trao   axial
    pull_force_w = Toplevel()
    pull_force_w.title("Introduzir valores de Pull Force")

    #definir varivel a utilizar em caixa de seleo do tipo stringVar - Tkinter
    maq_drop = StringVar()
    maq_drop.set(lista_maquinas_drop[0]) #definir item padrao da lista
    #criar e posicionar caixa de  seleo  ddo equipamento
    drop=OptionMenu(pull_force_w, maq_drop,*lista_maquinas_drop ).grid(row = 1,
        column=0,pady=4, padx=5)

    #definir entrada igual  da janela de  insero  de dados de  produo
```

```
    maq_drop.set(clicked_maq_drop.get())

    #criar e posicionar etiqueta para caixa de insero criada e
    Label(pull_force_w,text="Equipamento").grid(row = 0 , column=0,pady=4, padx=5)

    #Criar e posicionar na janela etiqueta e entrada para introduzir numero interno do
        instrumento de monitorizao
    Label(pull_force_w,text="Contador Interno MG3: ").grid(row = 0, column=2,pady=4,
        padx=5)
    e_numeromg3 = Entry(pull_force_w,borderwidth=2,width=25)
    e_numeromg3.grid(row = 1, column=2,pady=4, padx=5)


    #Criar e posicionar na janela etiqueta e entrada para introduzir numero de lote
    Label(pull_force_w,text="Lote").grid(row = 0, column=1,pady=4, padx=5)
    e_lot = Entry(pull_force_w,borderwidth=2,width=25)
    e_lot.grid(row = 1, column=1,pady=3, padx=5)

    #definir numero de lote igual  da janela de insero de dados de  produo  por defeito
    e_lot.insert(0,e_nlote.get())

    #Criar e posicionar na janela, etiqueta e entrada para introduzir valor de fora medido
    Label(pull_force_w,text="Valor de Pull Force medido [KgF]").grid(row = 0,
        column=3,pady=4, padx=5)
    e_pullforce = Entry(pull_force_w,borderwidth=2,width=25)
    e_pullforce.grid(row = 1, column=3,pady=4, padx=5)

    #Criar e posicionar na janela boto que quando pressionado executa a funo
        save_pullforce_click
    b_save_pullforce = Button(pull_force_w,text="Guarda valor de pull force
        ",width=25,command=save_pullforce_click)
    b_save_pullforce.grid(row = 2, column=3,pady=10, padx=6,columnspan=1)
```

Listing M.2: Codigo da função save_pullforce_click

```
# funo   executada quando boto "Guarda valor de pull force "  pressionado
    def save_pullforce_click():

        #definir variaveis globais em uso
        global path_app
        global path_db
        global threadDB

        #obter dados inseridos nos campos e introduzir em tuplo atraves do metodo .get
            sobre os objectos das entradas
        data_to_add_pullforce =
            (maq_drop.get(),e_lot.get(),e_numeromg3.get(),e_pullforce.get())
```

```
#verificar se nenhuma entrada  nula
if data_to_add_pullforce[1] != '' and data_to_add_pullforce[2] != '' and
    data_to_add_pullforce[3] != '':

    #criar consulta  base de dados com  instro  para introduzir os dados obtidos
        nos campos, atraves da propria query por isso o uso da string
    query_pullforce = "INSERT INTO pullforcedata
        (machine,nlote,countermg3,pullforcevalue) VALUES " +
        str(data_to_add_pullforce)

    #verificar se base de dados no esta a ser utilizada
    if threadDB.is_alive():
        threadDB.join()

    #inicar  ligao  a base de dados
    conn_db = sqlite3.connect(path_db + "\\"+"RW_DB.db",check_same_thread=False)

    #criar cursor
    cursor_db = conn_db.cursor()

    #executar consulta
    cursor_db.execute(query_pullforce)

    #comprometer com  alteraes
    conn_db.commit()

    #print de debug a informar dados inseridos
    print("data para db", data_to_add_pullforce)
```

Appendix N: Função date_log_match

```
#Funcao que faz a  associao  dos dados da  produo  e registos de soldadura
#Input: path_db (caminho para abase de dados) - string
#  No  tem retorno
def date_log_match(path_db):

    #declarar uso de variaveis globais
    global progress_bar_update
    global threadDB
    global path_app
    global b_update_db
    global progress_bar_label
    global root
    global stop_thread_val



    #Declarar varaiveis com objeto da barra de progresso e posicionamento no ecra atraves
        do metodo .grid
    progress_bar_update =
        ttk.Progressbar(root,orient=HORIZONTAL,length=200,mode="determinate")
    progress_bar_update.grid(row=6,column=2,padx=15,pady=4,columnspan=4)

    #tentar eliminar alguma informo exposta no ecra de modo a imprimir outra
    #    utilizado desta forma para o programa no bloquear caso no exista ainda informao
    #mas tenta-se elimiar uma variavel que no existe ainda
    try:
        progress_bar_label.destroy()
    except:
        pass

    #Declarar varaiveis com objeto da mensagem de informao e posicionamento no ecra
        atraves do metodo .grid
    progress_bar_label=Label(root,text="A atualizar datas")
    progress_bar_label.grid(row=5,column=2,padx=15,pady=10,columnspan=4)
```

```
#alterar  configuraes  do botoes
b_update_db['state']='disabled'
b_update_db['text']='A associar\n dados..'

#iniciar  ligao     base de dados e associar a uma variavel
conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db",check_same_thread=False)

#criar um cursor sobre essa ligao
cursor_db = conn_db.cursor()#criar um cursor

#criar e executar consulta a todos os elementos da tabela update data
cursor_db.execute("SELECT * FROM updatedata")

#guardar dados obtidos pela consulta em variavel
data_para_update= cursor_db.fetchall()

# declarao  de variaveis utilizadas para debug
inicio = datetime.now()
e=0
i=0
tamanho_datas_update=len(data_para_update)

#iterar sobre dados de  produo
for datas in data_para_update:


    # Verificao  se a thread que acessa a base de dados esta activa ou inactiva
    if threadDB.is_alive():
        stop_thread_val = True
        threadDB.join()

    #incrementar variavel para utilizar na barra de progresso e informacao
    i = i+1

    #definir novo valor para barra de progresso
    progress_bar_update['value']=(i/tamanho_datas_update)*80

    #definir nova mensagem com informao do progresso
    progress_bar_label['text']= "A corresponder lotes - " +
        str(round(((i/tamanho_datas_update)*100),2))+" %"

    #Atualizar  informaes  na interface grafica -  necessario porque estamos em uma
        janela secundaria toplevel
    root.update_idletasks()


    #inicar  ligao     base de dados
    conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db",check_same_thread=False)

    #criar cursor sobre objeto criado de ligao   base de dados
```

```
cursor_db = conn_db.cursor()

print("\n","*"*40," ",i ," of ",len(data_para_update),"*"*40)


#ano data[1][:4]
#mes data[1][5:7]
#dia data[1][8:10]
#hora data[1][11:13]
#minuto data[1][14:16]
#Segundo data[1][17:19]

#criar objetos para a hora inicial e final no formato datetime para que possam ser
    comparados com hora do registo
#                               Ano                 mes                dia
              hora                  minuto            segundo
data_inicio_lote = datetime(int(datas[1][:4]), int(datas[1][5:7]),
    int(datas[1][8:10]), int(datas[1][11:13]), int(datas[1][14:16]),
    int(datas[1][17:19]))
data_fim_lote = datetime(int(datas[2][:4]), int(datas[2][5:7]),
    int(datas[2][8:10]), int(datas[2][11:13]), int(datas[2][14:16]),
    int(datas[2][17:19]))

#construir consulta variaveis para utilzar na consulta  base de dados
data_inicio_lote_query=str(datas[1][8:10])+"-"+str(datas[1][5:7])+"-"+str(datas[1][:4])
data_fim_lote_query=str(datas[2][8:10])+"-"+str(datas[2][5:7])+"-"+str(datas[2][:4])

#Query para obter a  informao  sobre os registos que no tem numero lote e part number
sql_query="SELECT * FROM " + str(datas[0]).lower()+ " WHERE (N_Lote IS NULL AND
    (DATE = '"+data_inicio_lote_query+"' OR DATE = '"+data_fim_lote_query+"'))"

#print de debug a indicar intervalo de datas utilizado
print("Searching for logs in dates :
    ",str(int(datas[1][8:10]))+"-"+str(int(datas[1][5:7]))+"-"+str(int(datas[1][:4])),"
    and
    ",str(int(datas[1][8:10]))+"-"+str(int(datas[1][5:7]))+"-"+str(int(datas[1][:4])))

#executar a consulta  base de dados atraves do metodo execute da biblioteca sql
cursor_db.execute(sql_query)

#obter resultados da consulta atraves do metodo fetchall sobre o cursor
data_db=cursor_db.fetchall()

#print de debug que informa dados encontrados
print("Found: ",len(data_db)," Logs in this date needing Lot number")

#varaivel para guardar hora para calcular tempo que demorou o subciclo
time_inicial = datetime.now()

#Criamos uma lista para guardar LogId de registos que necessitem ser atualizados
```

```
lista_ID=[]

#Criar a query de  atualizao  que contem os dados da  produo
sql_update_query_2= "UPDATE " + str(datas[0]).lower() + " SET N_lote =
    '"+datas[3]+"', DESCRIPTION ='"+datas[4]+"', PART_NUM = '"+datas[5]+"',
    ITEM_CROWN = '"+datas[7]+"', LOT_CROWN ='"+datas[8]+"' ,I_SET =
    '"+datas[6]+"', ELECTRODE_REF = '"+datas[9]+"' WHERE (LogId = ? )"

#Iterar registos obtidos pela consulta  base de dados
for item in data_db:

    # formatao  se necessario - introduzir millisegundos porque existem registos
        que incluem
    if item[8][6:8]=='':

        hora_log = datetime(int(item[7][6:10]), int(item[7][3:5]),
            int(item[7][:2]), int(item[8][:2]), int(item[8][3:5]), int(0))
    else:
        hora_log = datetime(int(item[7][6:10]), int(item[7][3:5]),
            int(item[7][:2]), int(item[8][:2]), int(item[8][3:5]),
            int(item[8][6:8]))


    #se a data de um registo estiver contida entre a hora inicial e final da produo
    if hora_log > data_inicio_lote and hora_log < data_fim_lote:

        #incremento de variavel de debug
        e=e+1

        #adicionar  lista o identificador do registo
        lista_ID.append([item[-1]])

#print de debug no fim do ciclo a inidicar numero de registos que necessitam ser
    atualizados na base de dadps
print("Logs Matched - ", e)

#se a lista de registos a atualizar tiver elementos
if len(lista_ID)>0:

    #executar query a indicar que transao vai  comear  - melhorou performance de
        update
    cursor_db.execute('BEGIN TRANSACTION')

    #print de debug a indicar que processo de
    print("Updating DB - please wait - ",len(lista_ID)," Items to edit in DataBase")

    # introduo  de dados na base de dados atraves do metodo execute many
        utilizando sempre a query feita e iterando a lista de logid's a atualizar
    cursor_db.executemany(sql_update_query_2,lista_ID)
```

```
            #comprometer  alteraes    base de dados
            cursor_db.execute('COMMIT')

            #print de debug a indicar tempo que demorou a atualizao dos registos
            print("Update DB complete took: ",datetime.now()-time_inicial)

        else:
            #print de debug a indicar tempo que demorou a verifcao  base de dados
            print("No items to update in DB - elapsed time: ",datetime.now()-time_inicial)

        print("*"*95,"\n")


    #apos  verificao  acabar, tentar destruir objeto de barra de progresso
    try:
        progress_bar_update.destroy()
        progress_bar_label.destroy()
    except:
        pass


    #print de debug a indicar tempo total que demorou a atualizar a base de dados e
        quantos registos foram atualizados
    print("**"*20)
    print("\n Matching Done -- Tempo de decorrido = ",datetime.now() - inicio,"\n
        Atualizados ", e," registos" )
    print("\n","**"*20)
```

Appendix O: Função pull_force_update

```
#Funcao para fazer update da base dos dados de pullforce a registos de soldadura
#executada em thread
def pull_force_update():

    #declarar variaveis globais em uso
    global root
    global path_db
    global path_app
    global progress_bar_label
    global progress_bar_update

    #contruir e posicionar na janela principal objeto com barra de progresso
    progress_bar_update =
        ttk.Progressbar(root,orient=HORIZONTAL,length=200,mode="determinate")
    progress_bar_update.grid(row=6,column=2,padx=15,pady=4,columnspan=4)

    #tenta apagar possivel etiqueta da barra de progresso
    try:
        progress_bar_label.destroy()
    except:
        pass

    #Contruir e posicionar na janela de progresso etiqueta da barra de progresso
    progress_bar_label=Label(root,text="A atualizar datas")
    progress_bar_label.grid(row=5,column=2,padx=15,pady=10,columnspan=4)

    #iniciar  ligao     base de dados
    conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db")# connectar  base de dados

    #criar cursor para o objecto da ligao a base de dados
    cursor_db = conn_db.cursor()

    #executar consulta  base de dados relativamente a todos dados presentes na tabela
        pullforcedata
```

```
cursor_db.execute("SELECT * FROM pullforcedata")

#guardar resultados obtidos pela consulta em variavel
items_por_update = cursor_db.fetchall()

#atualizar etiqueta da barra de progresso para informar sonre n de dados encontrados
progress_bar_label["text"]="Pull forces registados " + str(len(items_por_update))

#criar varaivel para uso em barra de progresso etiqueta
i = 0

#ciclo de  construo  e submissao de consulta  base de dados
for item in items_por_update:

    try:
        # alteraes     barra de progresso e etiqueta
        progress_bar_update["value"]=(i/len(items_por_update))*100
        progress_bar_label["text"]="Submetidas "+ str(i) + " introdues de pull force "

    except:

        pass

    #incrementar variavel de progresso
    i = i + 1

    #criar e executar consulta de alterao da  descrio  de registos especificos para
        valor de fora medido
    cursor_db.execute("UPDATE " +str(item[0]).lower() + " SET DESCRIPTION = " +"'"+
        str(item[3])+"'" + " WHERE COUNTER = '"+ str(item[2])+ "' AND N_lote =
        '"+str(item[1]) + "'")

# alterao  de mensagem informativa na pagina principal
progress_bar_label["text"]="A gravar "+ str(i) + " alteraes "

#compremeter com  alteraes  feitas   base de dados
conn_db.commit()

#Print de debug a informar items atualizados
print("  Items com pullforce associado ", i)

#Destruir a barra de progresso
try:
    progress_bar_update.destroy()
except:
    pass

#atualizar  informao  da etiqueta na pagina inicial
progress_bar_label["text"]="Atualizadas "+ str(i) + " entradas de pull force "
```

Appendix P: Função match_routine

```
# Funo  que vai executar funes relacionadas com a associao de registos de soldadura
#Esta   executada em thread
def match_routine():

    #Declarar variaveis globais em uso
    global path_db
    global b_update_db
    global data_update_wi
    global progress_bar_update
    global progress_bar_label


    #alterar  configuraes  do botao de atualizar dados
    # de forma a nao permitir abrir a janela de atualizar dados
    b_update_db['state']='disabled'
    b_update_db['text']='A associar\n dados..'

    #Executar  funes
    try:

        #fechar a janela de inserir dados
        data_update_wi.destroy()

        #executar  funo  responsavel por associar registos de soldadura
        #a dados relativos   produo
        date_log_match()

        #executar  funo  que introduzir registos de teste destrutivo
        #em registos de soldadura
        pull_force_update()

    #caso alguma das  funes  de erro fazer print de debug
    except Exception as e:
```

```
        print("Problemas ao executar " ,e)

    #restaurar estado do boto de atualizar dados
    b_update_db['text']='Atualizar\nBase de Dados'
    b_update_db['state']='normal'
```

Appendix Q: Função extract_click

Listing Q.1: Codigo da função extract_click

```
# Funo  responsavel por gerar janela de extrao de dados da base de dados para csv
#Executada quando  pressionado o botao "Extrair da base de dados" na pagina principal
def extract_click():

    #declarar variaveis globais em uso
    global path_db
    global path_app
    global machine_list



    #Criar janela secundaria utilizando metodo toplevel da biblioteca Tkiner
    extract_w = Toplevel()

    #alterar titulo da pagina
    extract_w.title("Extrair dados para ficheiro .csv")

    #alterar  configuraes  de  arrumao  da pagina
    extract_w.columnconfigure(0, pad=3)
    extract_w.columnconfigure(1, pad=3)
    extract_w.rowconfigure(0, pad=3)

    #definir listas para serem utilizadas em menus colapsaveis
    lista_keyword=["N_lote","DESCRIPTION","PART_NUM","ITEM_CROWN","LOT_CROWN","ELECTRODE_REF"]
    lista_maquinas = []

    #adicionar apenas nome de equipamento  lista dos equipamentos
    for item in machine_list:
        lista_maquinas.append(item[0])
```

```
#declarar variavel para guardar numero de registos encontrados
size_f=str(0)

#declarar variaveis do tipo string do tkinter para usar para guardar varavel
    selecionada dos menu colapsaveis
keyword_input = StringVar()
machine_select = StringVar()

#definir entradas pardro de listas colapsaveis
machine_select.set(lista_maquinas[0])
keyword_input.set(lista_keyword[1])

#Criar e posicionar variavel que contem objecto do etiqueta e caixa de seleo do
    equipamento
Label(extract_w,text="Maquina:").grid(row=0, column=0,pady=4, padx=5)
select_machine=OptionMenu(extract_w, machine_select,*lista_maquinas )
select_machine.grid(row=0, column=1,pady=4, padx=5,ipadx=20)

#Criar e posicionar variavel que contem objecto da etiqueta e caixa de entrada da data
Label(extract_w,text="Data:").grid(row=1, column=0,pady=4, padx=5)
e_data = Entry(extract_w,borderwidth=3,width=15)
e_data.grid(row = 1, column=1)

#Criar e posicionar variavel que contem objecto da etiqueta e caixa de entrada do
    numero inicial do contador interno do instrumento de monitorizao
Label(extract_w,text="Contador Inicio:").grid(row=2, column=0,pady=4, padx=5)
e_nf = Entry(extract_w,borderwidth=3,width=15)
e_nf.grid(row = 2, column=1)

 #Criar e posicionar variavel que contem objecto da etiqueta e caixa de entrada do
     numero finaldo contador interno do instrumento de monitorizao
Label(extract_w,text="Contador Fim:").grid(row=3, column=0,pady=4)
e_nl = Entry(extract_w,borderwidth=3,width=15)
e_nl.grid(row=3, column=1,pady=4, padx=5)


#Criar e posicionar variavel que contem objecto do etiqueta e caixa de seleo da keyword
Label(extract_w,text="Filtrar por \nKeyword:").grid(row=4, column=0,pady=4, padx=5)
select_keyword=OptionMenu(extract_w, keyword_input,*lista_keyword )
select_keyword.grid(row=4, column=1,pady=4, padx=5)

#Criar e posicionar variavel que contem objecto da etiqueta e caixa de entrada da
    keyword
e_keyword_value = Entry(extract_w,borderwidth=3,width=30)
e_keyword_value.grid(row=5, column=0, columnspan=2,pady=4, padx=5)

#Criar e posicionar variavel que contem objecto do boto que inicia o processo de gerar
    o ficheiro
b_generate_file=Button(extract_w, text="Gerar
    Ficheiro",borderwidth=3,command=generate_file)
```

```
b_generate_file.grid(row=7, column=4,pady=4, padx=5,columnspan=2,ipadx=50,ipady=5)

#Criar e posicionar variavel que contem objecto da etiqueta com o numero de registos
    encontrados durante a consulta  base de dados
Label(extract_w,text="Items encontrados:").grid(row=7, column=0,pady=4,padx=5)
files_found=Label(extract_w,text=size_f)
files_found.grid(row=7, column=1,pady=4,sticky="W")
```

Listing Q.2: Codigo da função generate_file

```
# Funo  responsavel por obter a informao das entradas, realizar a consulta e gerar um
    ficheiro csv com resultados obtidos
def generate_file():

    #executar funcao query_ builder que constroi consulta e executa-a na base de dados
    data =
        query_builder(path_db,machine_select.get(),e_data.get(),e_data.get(),keyword_input.get(),e_keyword_

    #guardar numero de registos encontrados
    size_f=str(len(data))

    #alterar numero na interfacce
    files_found.config(text=size_f)

    #criar string com mensagem para ser utilizada em caixa de aviso
    string= "Items encontrados: " + str(size_f) +"\nGerar ficheiro ?\n"

    #Caixa de aviso a perguntar se deve gerar ficheiro
    #caso a resposta seja sim
    if messagebox.askokcancel("Aviso",string) == 1:

        #mensagem de debub
        print("Gerar ficheiro!")

        try:
            #criar nome de ficheiro
            file_name = str(data[0][0])+"_"+str(data[0][1])+".csv"

            #criar duas primeiras linhas do ficheiro
            maquina_enter=['Maquina:',str(machine_select.get()),]
            header_file=["Numero lote","Descrio","Part Number","Item Coroa","Lote
                Coroa","Setpoint Corrente[kA]","Referencia Eletrodo",
                "Data","Hora","Contador","Programa","Ipk
                [kA]","Upk[V]","POT[kW]","R[Ohm]","s1[m]","s3[m]","F[N]","p[bar]","Irms[kA]","Q[Asec]","Urms
                "PROG_NAME","EXT_INFO",]

            #abrir ficheiro em modo w significa escrever ficheiro
            with open(file_name, "w",newline='') as file:
```

```
                #utilizar metodo writer da biblioteca csv para criar objeto
                Ficheiro = csv.writer(file)

                #inserir duas primeiras linhas com metodo writerow em cada lista
                Ficheiro.writerow(maquina_enter)
                Ficheiro.writerow(header_file)

                #inserir todos os registos com metodo writerows em lista de listas
                Ficheiro.writerows(data)

        #Caso ocorra um problema a gerar o ficheiro
        except Exception as e:

            #abrir caixa com mensagem de erro
            messagebox.showerror("Erro a criar o ficheiro",e)

    else:
            #print de debug caso seja respondido no a quando de gerar o ficheiro
            print(" No  gerar ficheiro")
```

Appendix R: Função query_builder

```
# funo  responsavel por gerar consulta  base de dados com base no conjunto de parametros
    introduzido
#Entradas: machine - string - identificao do equipamento
#         initialdate - string - data inicial inserida
#         finaldate - string - data final inserida
#         keyword - string   - verifica qual o tipo de keyword a usar
#         keywordvalue - string -   indica o valor da keyword em si
#         inicial_counter - string - valor inicial do contador do instrumento de
    monitorizao
#         final_counter - string - valor final do contador do instrumento de monitorizao
# Retorna : lista com resultados obtidos pela consulta
def
    query_builder(machine,initialdate,finaldate,keyword,keywordvalue,inicial_counter,final_counter):

    #Declarar variaveis globais em utilizao
    global path_app
    global path_db


    #definir base da consulta : Selecionar tudo do equipamento
    query="SELECT * FROM "+str(machine).lower()

    #criar lista para a guardar consultaas a realizar - para varias datas
    query_list=[]


    # verificao  se foi inserido keyword
    if keywordvalue != '':
        query = query + " WHERE ( "+ str(keyword) + " = '"+str(keywordvalue) + "'"

    #caso  no  ter sido inserido keyword mas seja contador
    elif inicial_counter != '' and final_counter != '':
        query = query + " WHERE ( COUNTER BETWEEN "+ str(inicial_counter) + " AND
            "+str(final_counter)


    # se existir  introduo  de keyword e de contador do instrumento de monitorizao
```

```
if inicial_counter != '' and final_counter != '' and keywordvalue != '':

    query = query + " AND COUNTER BETWEEN "+ str(inicial_counter) + " AND
        "+str(final_counter)


#Se tiverem sido inseridas entradas da data
if initialdate != '' and finaldate != '':

    #criar variaveis do tipo datetime com base em entradas
    start_date_datetime= datetime.strptime(initialdate,'%d-%m-%Y')
    final_date_datetime= datetime.strptime(finaldate,'%d-%m-%Y')

    #guardar intervalo de tempo entre datas
    delta = final_date_datetime - start_date_datetime

    #lista para guardar datas
    dates_list=[]

    #obter todas as datas entre as datas introduzidas
    for i in range(delta.days + 1):

        #utilizado o metoodo timedelta() para aumentar a variavel do tipo datetime para
            obter data
        day = start_date_datetime + timedelta(days=i)

        #adcionar string  lista com data correspondente no formato dd-mm-yyyy
        dates_list.append(day.strftime('%d-%m-%Y'))

    #para cada data obtida vamos criar uma consulta
    for item in dates_list:

        #finalizar a consulta executada ate agora com data
        #caso a data seja a unica entrada
        if inicial_counter == '' and final_counter == '' and keywordvalue == '':

            temp_query = query + " WHERE ( DATE = '"+str(item) + "')"
            query_list.append(temp_query)

        # caso a consulta ja tenha sido alterada
        else:
            temp_query = query + " AND DATE = '"+str(item)+"')"
            query_list.append(temp_query)

#adicionar a entrada a lista caso seja para pesquisar tudo
#essecialmente a consulta inicial
elif inicial_counter == '' and final_counter == '' and keywordvalue == '':
    query_list.append(query)

#caso tenha sido introduzido algum filtro
```

```
#terminar a consulta introduzindo ")" e introduzir na lista das consultas
else:
    query = query + ")"
    query_list.append(query)



#iniciar a  ligao     base de dados e criar cursor sobre essa ligao
conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db")
cursor_db = conn_db.cursor()

#criar variavel para guardar resultados das consultas realizadas
data_list = []

#iterar consultas da lista
for consulta in query_list:

    #executar consulta
    cursor_db.execute(consulta)

    #recolher dados obtidos pela consulta
    data = cursor_db.fetchall()

    #introduzir cada um desses registos em lista unica
    for item in data:
        data_list.append(item)

#retornar a lista com os registos obtidos
return data_list
```

Appendix S: Função data_report_window

Listing S.1: Codigo da função data_report_window

```
# Funo  reponsavel criar a janela e objecto relativos a gerar o relatorio
#Esta pagina disponibliza uma entrada e uma caixa de multipla seleo para definir os
    criteiros do relatorio
def data_report_window():


    #declarar variaveis globais em uso
    global lista_maquinas

    #criar janela secundaria para a pagina de relatorios
    data_a_w = Toplevel()

    #definir titulo
    data_a_w.title("Data window")

    #configurar linhas e colunas
    data_a_w.columnconfigure(0, pad=3)
    data_a_w.columnconfigure(1, pad=3)
    data_a_w.rowconfigure(0, pad=3)

    #criar variavel que vai guardar o tipo de condicionante do relatorio
    clicked_select = StringVar()

    #definir lista de possibilidades e valor padro
    lista_select=["N_lote","DESCRIPTION","PART_NUM","ITEM_CROWN","LOT_CROWN","ELECTRODE_REF"]
    clicked_select.set(lista_select[1])

    #criar objecto de menu colaspsvel utilizando a lista e variavel anterior
    select_drop=OptionMenu(data_a_w, clicked_select,*lista_select )
    select_drop.grid(row = 0, column=0,padx=5,pady=5)

    #implementar entrada de texto na janela e poscicionamento da mesma
```

```
e_pn=Entry(data_a_w,borderwidth=3,width=10)
e_pn.grid(row = 0, column=1,padx=10,pady=10)

#implementar o boto de calcular na pagina dos relatorios
b_calc=Button(data_a_w, text="Calcular",borderwidth=3,command=calc_click)
b_calc.grid(row = 0, column=2,padx=10,pady=10)

#criar variavel que depois vai apresentar os resultados obtidos
l_results = Label(data_a_w, text="")
l_results.grid(row = 2, column=0,rowspan=3)
```

Listing S.2: Codigo da função calc_click

```
# Funo  executada quando boto "calcular" na janela dos relatorios  pressionada
  #funcao que reponsavel por agrupar os dados necessario para criar os graficos e o
      relatorio em si
  def calc_click():

      #declarar variaveis globais em uso
      global machine_list
      global progress_bar_label
      global root
      global progress_bar_update

      #criar barra de progresso na pagina principal, posicionar na janela e definir
          valor inicial
      progress_bar_update =
          ttk.Progressbar(root,orient=HORIZONTAL,length=200,mode="determinate")
      progress_bar_update.grid(row=6,column=2,padx=15,pady=4,columnspan=4)
      progress_bar_update["value"]=0

      #tenta apagar etiqueta da pagina principal utilziada anteriormente
      try:
          progress_bar_label.destroy()
      except:
          pass

      #Criar objeto com a etiqueta da pagina principal
      progress_bar_label=Label(root,text="A gerar graficos")
      progress_bar_label.grid(row=5,column=2,padx=15,pady=10,columnspan=4)


      #criar variavel para guardar nomes dos equipamentos
      machines = []

      #ciclo de  obteno  de equipamentos
      for item in machine_list:
          machines.append(item[0].upper())
```

```
#definir variaveis e unidades
normal_items =
    ["Ipk","Upk","POT","R","s1","s3","F","p","Irms","Q","Urms","Ut","W","tw","twcyc","AY"]
units_items = ["[kA]", "[V]","[kW]",
    "[Ohm]","[m]","[m]","[N]","[bar]","[kA]","[As]","[V]","[Vs]","[Ws]","[ms]","[cyc]","[deg]"]

#defenir variaveis a utizar durante a organizao dos dados
count = 0
string_results=''

#variavel que armazena dados a introduzir no PDF
data_pdf=[]
last_item=('','','','','','','')

#lista de dados com informaes do lote
about_data=[]

#lista de dados da base de dados
data = []

#para cada equipamento regitado
for machine in machines:

    #realizar consulta base de dados utilizando apenas o equipamento a keyword
        como argumento e guardar em variavel
    data_db = query_builder(machine,'','',clicked_select.get(),e_pn.get(),'','')

    #para cada entrada existe na consulta
    for item in data_db:

        #adicionamos entrada a lista de registos
        data.append(item)

        #verificar se as informaes relativas a produo so iguas entre registos
            seguidos
        if (item[0],item[1],item[2],item[3],item[4],item[5],item[6]) != last_item:

            #caso no seja criado um novo conjuntpo para adicionar a lista com
                dados relativos a produo
            last_item =
                tuple((item[0],item[1],item[2],item[3],item[4],item[5],item[6]))

            about_data.append(last_item)

    # a informao obtida +e adicionada variavel relativa aos registos encontrado
        para cada equipamento
    string_results=string_results+ "Items in " +machine+" with " +" " +
        clicked_select.get()+" = " + e_pn.get()+ " - Logs Found: "+
        str(len(data_db)) +"\n"
```

```
#e adicionada  variavel para aprensentar apos os calculos serem realizados a que
    keyword diz respeito
string_results=string_results+ "\nFor: "+ clicked_select.get()+" =
    "+e_pn.get()+"\n"*2


#Iterar sobre as diferentes varaiveis medidas
#da lista normal_items =
    ["Ipk","Upk","POT","R","s1","s3","F","p","Irms","Q","Urms","Ut","W","tw","twcyc","AY"]
for item in normal_items:

    #incrementar contagempara utilizar em barra de progresso
    count = count + 1

    #definir novo valor na barra de progresso
    progress_bar_update["value"]=((count/len(normal_items))*100)

    #atualizar pagina principal
    root.update()

    #declarar variavel na qual os dados da base de dados, transformados, vo ser
        guardados
    data_plot =[]

    #declarar opcoes possiveis - lista indentica a organizao da base de dados
    options=["DATE","TIME","COUNTER","PROGRAM","Ipk","Upk","POT","R","s1","s3",
    "F","p","Irms","Q","Urms","Ut","W","tw","twcyc","AY","STATUS","DETAILS",
    "PROG_NAME","EXT_INFO"]

    #guardar indice correspondete da variavel a analisar com base na lista de items
        normais
    indice=options.index(item)

    #criar lista com dados relativos apenas a essa variavel
    for log in data:

        data_plot.append(float(log[indice+7].replace(",",".")))

    #calculo da media dos valores
    media=np.mean(data_plot)

    #desvio padro da media
    std_dv= np.std(data_plot, ddof=1)

    #declarar variaveis usadas para registar quantidade de registos em diferentes
    #grupos de afastamento da media
    mean_maisdev=0
    mean_mais2dev=0
    mean_mais3dev=0
    mean_maisfora=0
```

```
mean_menosdev=0
mean_menos2dev=0
mean_menos3dev=0
mean_menosfora=0

# verificao  do grupo de cada valor
for val in data_plot:
    if val > (media ) and val < (media + std_dv) :
        mean_maisdev = mean_maisdev +1

    elif val > (media + std_dv) and val < (media + 2*std_dv):
        mean_mais2dev = mean_mais2dev +1

    elif val > (media + 2*std_dv) and val < (media + 3*std_dv):
        mean_mais3dev = mean_mais3dev +1

    elif val > (media + 3*std_dv):
        mean_maisfora = mean_maisfora+1

    elif val < (media ) and val > (media - std_dv) :
        mean_menosdev = mean_menosdev +1

    elif val < (media - std_dv) and val > (media - 2*std_dv):
        mean_menos2dev = mean_menos2dev +1

    elif val < (media - 2*std_dv) and val > (media - 3*std_dv):
        mean_menos3dev = mean_menos3dev +1

    elif val < (media - 3*std_dv):
        mean_menosfora = mean_menosfora+1

#caso a media e o desvio padro sejam diferentes de 0 os dados so submetidos a
    lista de dados a inserir na base de dados
if media != std_dv and std_dv !=0:

    #adicionar dados a lista de dados para gerar relatorio
    data_pdf.append((item,media,std_dv,mean_menosfora,mean_menos3dev,mean_menos2dev,mean_menosdev,me

    #executar  funo  de  construo  de graficos
    plots_generator(data_plot,item,False)#graficos
    plots_generator(data_plot,item,True)#histograma~

#print de debug a indiciar a media da variavel e o desvio padro
print("Media do",item,"=",round(media,4),"c/ desvio padro = ",round(std_dv,4))

#adicionar resultados  mensagem a apresentar na janela
string_results=string_results+ "Media "+item+" =
    "+str(round(media,4)).replace(".",",")+" " +
    units_items[normal_items.index(item)]+" " +" com um std. dev de
    "+str(round(std_dv,4)).replace(".",",")+"\n"*2
```

```
#definir etiquetas utilizadas para apresentar os resultados calculados ao
    utilizador e colocar na lista
Label(data_a_w, text="Results:").grid(row = 1, column=1)
l_results.config(text=string_results)
l_results.grid(row = 2, column=1,rowspan=2,padx=10,sticky="e")


#Caixa de aviso que pergunta se queremos mesmo gerar ficheiro pdf do relatorio
if messagebox.askokcancel("Aviso","Gerar ficheiro PDF ? ") == True:

    #print de debug a indicar que o ficheiro vai ser gerado
    print("gerar - about ",about_data)

    #Alterar etiqueta da pagina principal
    progress_bar_label['text']="A gerar relatorio..."
    progress_bar_update.destroy()

    #executar  funo  que gera ficheiro PDF
    report_generator(machine.upper(),data_pdf,about_data)

    #alterar etiqueta da janela principal
    progress_bar_label['text']="Relatorio gerado"
```

Appendix T: Função plots_generator

```
# funo  responsavel por criar os graficos utilizados no relatorio
#inputs: data - conjunto de dados a introduzir
#       specific - string - Indica sobre que variavel esta a ser construido o grafico
#        bar - boolean - se for true gerar grafico histograma, se for false, grafico de
    linha
def plots_generator(data,specific,bar):

    #calculo de media e devio padro de lista de dados utilizada como argumento
    mean=np.mean(data)
    std_dv= np.std(data, ddof=1)

    #declarar lista e variaveis onde so guardados conjuntos de dados
    data_dot=[]
    data_x=[]


    #contruir listas para servirem de barras nas no grafico
    data_mean=[mean]*len(data)
    data_mais3dev=[mean +3*std_dv]*len(data)
    data_menos3dev=[mean -3*std_dv]*len(data)
    data_mais2dev=[mean +2*std_dv]*len(data)
    data_menos2dev=[mean -2*std_dv]*len(data)
    data_mais1dev=[mean +1*std_dv]*len(data)
    data_menos1dev=[mean -1*std_dv]*len(data)



    #  opes  possiveis de varaiveis utilizadas para fazer graficos
    options=["Ipk","Upk","POT","R","s1","s3","F","p","Irms","Q","Urms","Ut","W","tw","twcyc","AY"]

    #etiqueta correspondentes
    options_y_label=["Ipk - Corrente de Pico [kA]","Upk - Tenso de Pico [V]","POT -
        Potencia [kW]","R - Resistencia []","s1","s3","F","p","Irms - Corrente RMS
        [kA]","Q","Urms - Tenso RMS [V]","Ut","W","tw","twcyc","AY"]
```

```
i=0

#ciclo que cria lista de valores a ser utilizado para o grafico
for item in data:
    #data para o eixo y, substituido , por .
    data_dot.append(float(str(item).replace(",","")))

    #data para o eixo x
    data_x.append(i)

    i=i+1

#caso no argumento bar esteja False
if bar == False:

    #definir nome do ficheiro
    file_name="G_"+specific+".png"

    #defninir tamanho da figura
    fig_1=plt.figure(figsize=(10,5))

    #adicionar eixos
    axes_1=fig_1.add_axes([0.1,0.1,0.9,0.9])

    #definir titulo de eixos e do grafico
    axes_1.set_xlabel("Data Points")
    axes_1.set_ylabel(options_y_label[options.index(specific)])
    axes_1.set_title("Grfico - "+ options_y_label[options.index(specific)])

    #introduzir os dados os registos no grafico
    axes_1.plot(data_x,data_dot,lw=0.8)

    #introduzir linhas ao grafico
    axes_1.plot(data_x,data_mean,color='black',label='mean',lw=0.5)

    axes_1.plot(data_x,data_mais3dev,color='red',lw=0.5,ls='dashed',label='mean3')
    axes_1.plot(data_x,data_menos3dev,color='red',lw=0.5,ls='dashed')

    axes_1.plot(data_x,data_mais2dev,color='orange',lw=0.5,ls='dashed')
    axes_1.plot(data_x,data_menos2dev,color='orange',lw=0.5,ls='dashed',label='mean2')

    axes_1.plot(data_x,data_mais1dev,color='green',lw=0.5,ls='dashed')
    axes_1.plot(data_x,data_menos1dev,color='green',lw=0.5,ls='dashed',label='mean')

    #atribuir legenda ao grafico
    axes_1.legend(loc='best')


#caso no argumento bar esteja True
if bar == True:
```

```
    #definir nome do grafico para ser gravado
    file_name="H_"+specific+".png"

    #guardar variavel com tamanho total dos dados
    total_size = len(data)

    #utiliar o metodo hist construir o grafico de histgrama
    # so  indicados nos argumentos quais os grupos a utilizar para dividir os dados
        atraves do paramero bin
    n,bins,patches=plt.hist(data_dot,bins=[mean-3*std_dv,mean-2*std_dv,mean-1*std_dv,mean,mean+1*std_dv,mea
        edgecolor='black',align='mid')

    #lista que vai guardar a % de registos em cada contentor
    list_relative_density=[]

    #calculo % dos dados para cada contentor
    for item in n:
        string = "\n"+str(round((item/total_size)*100,2))+"%"
        list_relative_density.append(string)

    x=3
    y=0.01

    #posicionamento das etiquetas por baixo do contentor
    for i in range(0, len(n)):

        x_pos = (bins[i + 1] - bins[i]) / x + bins[i]
        y_pos = n[i] + (n[i] * y)
        label = str((n[i]))+list_relative_density[i]
        plt.text(x_pos, y_pos, label)

    #lista com texto a mostrar no eixo do x
    list_string_plot=[str(round(bins[0],3))+"\nxbar-3",str(round(bins[1],3))+"\nxbar-2",str(round(bins[2],3

    #Associar posicionmento a texto a mostrar
    plt.xticks([mean-3*std_dv,mean-2*std_dv,mean-1*std_dv,mean,mean+1*std_dv,mean+2*std_dv,mean+3*std_dv],li

    #aumentar o tamanho do grafico para no ficar muito apertado
    plt.ylim(0, max(n)+300)

    #definir titulo dos eixos
    plt.ylabel("N de Peas ")
    plt.xlabel("Intervalos")

#extrair o grafico para o caminho com o nome do ficheiro
plt.savefig("graphs"+"\\"+file_name)

#limpar o objecto do que contem o grafico
plt.clf()
```

Appendix U: Função report_generator

```
# Funo  que gera pdf com base nos calculos feitos pela funcao plots generator
def report_generator(machine,data_pdf,pdf_info):

    #declarar variavel global em uso
    global path_app

    #mudar a diretoria de trabalho
    os.chdir(path_app)

    #definir variaveis de paginas possiveis assim como unidades
    options=["Ipk","Upk","POT","R","s1","s3","F","p","Irms","Q","Urms","Ut","W","tw","twcyc","AY"]
    options_y_label=["Ipk - Corrente de Pico [kA]","Upk - Tenso de Pico [V]","POT -
        Potencia [kW]","R - Resistencia ","Deslocamento s1","Deslocamento
        s3","Fora","Presso","Irms - Corrente RMS [kA]","Q - Ampere por segundo","Urms -
        Tenso RMS [V]","Ut - tenso por segundo","Ws - potencia por segundo","Durao
        soldadura","twcyc","AY"]
    units = ["[kA]", "[V]","[kW]",
        "[Ohm]","[m]","[m]","[N]","[bar]","[kA]","[As]","[V]","[Vs]","[Ws]","[ms]","[cyc]","[deg]"]

    #guardar hora e data em variavei
    str_time= datetime.now().strftime("%H:%M:%S") +" "+date.today().strftime('%d-%m-%Y')

    #guardar numero total de peas em variavel
    total_pecas= data_pdf[0][4] +data_pdf[0][5] +data_pdf[0][6] +data_pdf[0][7]
        +data_pdf[0][8] +data_pdf[0][9]

    #criar objecto da bibliteca FPDF onde vai ser introduzida a informacao
    #definimos o tamanho A4 e a descriacao de distancias em milmetros
    # e queremos o pdf em portrait 'P'
    pdf = FPDF('P', 'mm', 'A4')

    #Para cada variavel que existe dados vai ser criada uma nova pagina
    for item in data_pdf:
```

```
#adicionar pagina ao
pdf.add_page()

# Cabealho  do pdf

#definir fonte a usar no titulo
pdf.set_font("Arial",size = 15)

#adicionar imagem com logo da empresa e titulo
pdf.image("resources\Kemet.png", x=3, y=2, w=40, h=15)
pdf.cell(190, 0, txt = "Resistance Welding - Report", ln = 1, align = 'C')

#definir fonte para outros elementos do cabealho
pdf.set_font("Arial",size = 10)

#inserie elementos que mostram a data e hora a que foi gerado o registo
pdf.cell(350, -8, txt = "Generated: ", ln = 1, align = 'C')
pdf.cell(350, 20, txt = str_time, ln = 1, align = 'C')

#Conteudo igual para todas a paginas - informaes sobre  produo

#adiciona linha para separar informacao de lote
pdf.line(x1 = 0, y1 = 18, x2 = 600, y2 = 18)

pdf.cell(0, 2, txt = "Report based on: ", ln = 2, align = 'L')
line = "Machine: " + str(machine) +" "*6 +"N Lote: " +str(pdf_info[0][0])+ " "*6 +
    "Part Number: "+ str(pdf_info[0][2]) + " "*6 +"N Logs: " +str(total_pecas)

pdf.cell(0, 10, txt = line, ln = 3, align = 'C')
line = "Item Crown: " +str(pdf_info[0][3])+" "*6 +"N Lote Crown: "
    +str(pdf_info[0][4])+" "*6 + "Ref. Eletrode: "+str(pdf_info[0][3])
pdf.cell(0, 2, txt = line, ln = 4, align = 'C')

pdf.line(x1 = 0, y1 = 40, x2 = 600, y2 = 40)

#Adicionar a varaivel utilizada para os calculos para inserir no pdf
label_s=options_y_label[options.index(item[0])]

#alterar fonte
pdf.set_font("Times",size = 11)

#introduzir texto indicando a variavel no pdf
pdf.cell(0, 15, txt = label_s, ln = 5, align = 'C')

#defenir fonte para o resto do conteudo
pdf.set_font("Times",size = 10)

# introduo  de calculos na pagina
line ="Mean: " + str(round(item[1],4)).replace(".",",") +" "
    +units[options.index(item[0])] + " "*6 + "Std Dev: " +
```

```
    str(round(item[2],4)).replace(".",",") +" " +units[options.index(item[0])]+ "
    "*6 +"Coefficient of variation: "+
    str(round(item[2]/item[1],3)*100).replace(".",",")+" %"
pdf.cell(0, 0, txt =line, ln = 6, align = 'C')

#adicionar linha em branco
line=" "
pdf.cell(0, 10, txt =line, ln = 7, align = 'C')

# introduo  de calculos na pagina
line = "Mean + std.Dev: "+str(round(item[1]+item[2],3)).replace(".",",")+" "
    +units[options.index(item[0])]+ " "*6 +"Mean + 2std.Dev:
    "+str(round(item[1]+2*item[2],3)).replace(".",",")+" "
    +units[options.index(item[0])]+ " "*6 +"Mean + 3std.Dev: "+
    str(round(item[1]+3*item[2],3)).replace(".",",")+" "
    +units[options.index(item[0])]
pdf.cell(0, 0, txt =line, ln = 8, align = 'C')

# introduo  de calculos na pagina
line = "Mean - std.Dev: "+str(round(item[1]-item[2],3)).replace(".",",")+" "
    +units[options.index(item[0])]+ " "*6 +"Mean - 2std.Dev:
    "+str(round(item[1]-2*item[2],3)).replace(".",",")+" "
    +units[options.index(item[0])]+ " "*6 +"Mean - 3std.Dev: "+
    str(round(item[1]-3*item[2],3)).replace(".",",")+" "
    +units[options.index(item[0])]
pdf.cell(0, 10, txt = line, ln = 9, align = 'C')

#adicionar linha em branco
line=" "
pdf.cell(0, 5, txt =line, ln = 10, align = 'C')

# introduo  de calculos na pagina
line = "Mean + std.Dev: "+str(item[7])+" Pieces" +" "*6 +"Mean + 2std.Dev:
    "+str(item[8])+" Pieces" +" "*6 +"Mean + 3std.Dev: "+str(item[9])+" Pieces"+"
    "*6 # +">Mean + 3std.Dev: "+str(item[10])+" Pieces"
pdf.cell(0, 0, txt = line, ln = 11, align = 'C')

# introduo  de calculos na pagina
line = "Mean - std.Dev: "+str(item[6])+" Pieces" +" "*6 +"Mean - 2std.Dev:
    "+str(item[5])+" Pieces" +" "*6 +"Mean - 3std.Dev: "+str(item[4])+" Pieces"+"
    "*6 # +"<Mean - 3std.Dev: "+str(item[3])+" Pieces"
pdf.cell(0, 10, txt = line, ln = 12, align = 'C')

#adicionar linha em branco
line=" "
pdf.cell(0, 5, txt =line, ln = 13, align = 'C')

# introduo  de calculos na pagina
line = "Mean  std.Dev: "+str(item[6]+item[7])+" Pieces" +" "*6 +"Mean  2std.Dev:
    "+str(item[5]+item[8])+" Pieces" +" "*6 +"Mean  3std.Dev:
```

```
        "+str(item[4]+item[9] )+" Pieces"+" "*6 # +"<Mean - 3std.Dev: "+str(item[3])+"
        Pieces"
    pdf.cell(0, 0, txt = line, ln = 14, align = 'C')


     # introduo  de calculos na pagina
    line = "Mean  std.Dev: "+str(round(100*((item[6]+item[7])/total_pecas),3))+" %" +"
        "*6 +"Mean  2std.Dev: "+str(round(100*((item[5]+item[8])/total_pecas),3))+" %"
        +" "*6 +"Mean  3std.Dev: "+str(round(100*((item[4]+item[9])/total_pecas),3))+"
        %"+" "*6 # +"<Mean - 3std.Dev: "+str(item[3])+" Pieces"
    pdf.cell(0, 10, txt = line, ln = 15, align = 'C')

     # introduo  de calculos na pagina
    line = "Outside Mean  3std.Dev: " +str(item[4]+item[9])+" Pieces "
        +str(round(100*((item[4]+item[9])/total_pecas),3))+" %"
    pdf.cell(0, 0, txt = line, ln = 16, align = 'C')

    #Atribuir caminhos a variaveis com o caminho para os graficos relativos  variavel
    image_G="graphs\G_"+str(item[0])+".png"
    image_H="graphs\H_"+str(item[0])+".png"

    #adicionar as imagens ao pdf
    pdf.image(image_G, x=15, y=110, w=170, h=85)
    pdf.image(image_H, x=17, y=200, w=180, h=75)

#definir nome do ficheiro a extrair
#nome do ficheiro  o numero de lote caso contrario o partnumber
if str(pdf_info[0][0])=='' or len(pdf_info)>1:

    name_file="reports\\"+str(pdf_info[0][1])+".pdf"
else:
    name_file = "reports\\"+str(pdf_info[0][0])+".pdf"


#metodo output para extrarir o ficheiro depois de serem construidas todas as paginas
pdf.output(name_file)
```

Appendix V: Exemplo de relatorio extraido do programa

KEMET a YAGEO company

Resistance Welding - Report

Generated:
12:44:11 15-03-2023

Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

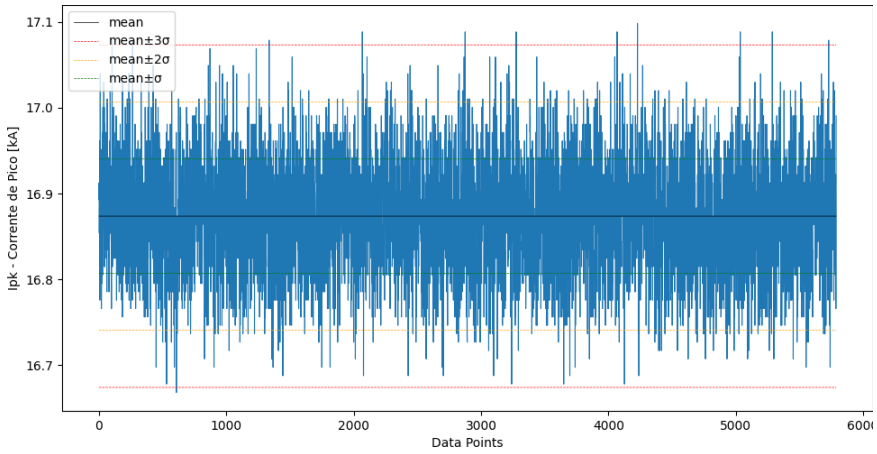
Ipk - Corrente de Pico [kA]

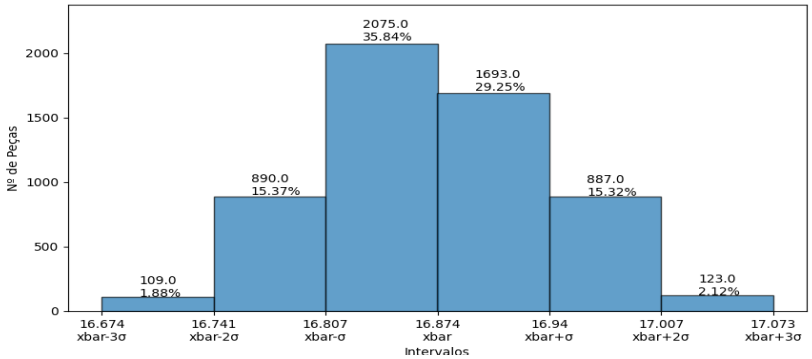
Mean: 16,8737 [kA] Std Dev: 0,0665 [kA] Coefficient of variation: 0,4 %

Mean + std.Dev: 16,94 [kA] Mean + 2std.Dev: 17,007 [kA] Mean + 3std.Dev: 17,073 [kA]
Mean - std.Dev: 16,807 [kA] Mean - 2std.Dev: 16,741 [kA] Mean - 3std.Dev: 16,674 [kA]

Mean + std.Dev: 1693 Pieces Mean + 2std.Dev: 887 Pieces Mean + 3std.Dev: 123 Pieces
Mean - std.Dev: 2075 Pieces Mean - 2std.Dev: 890 Pieces Mean - 3std.Dev: 109 Pieces

Mean ± std.Dev: 3768 Pieces Mean ± 2std.Dev: 1777 Pieces Mean ± 3std.Dev: 232 Pieces
Mean ± std.Dev: 65.224 % Mean ± 2std.Dev: 30.76 % Mean ± 3std.Dev: 4.016 %
Outside Mean ± 3std.Dev: 232 Pieces 4.016 %





Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

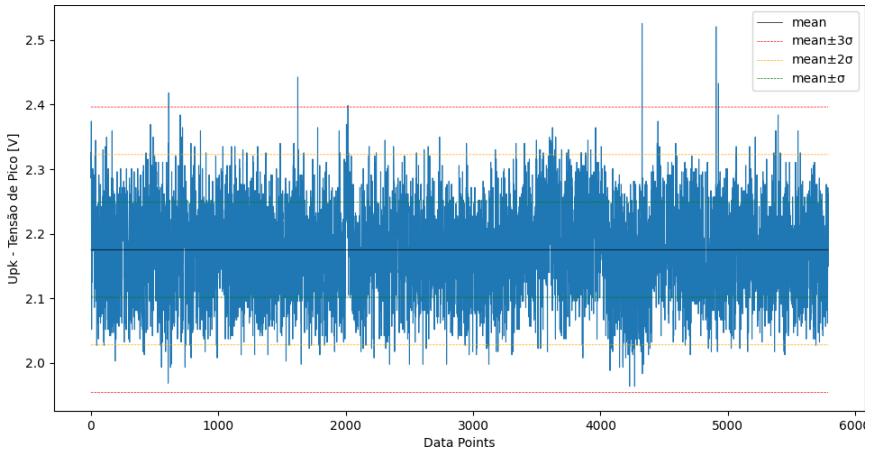
Upk - Tensão de Pico [V]

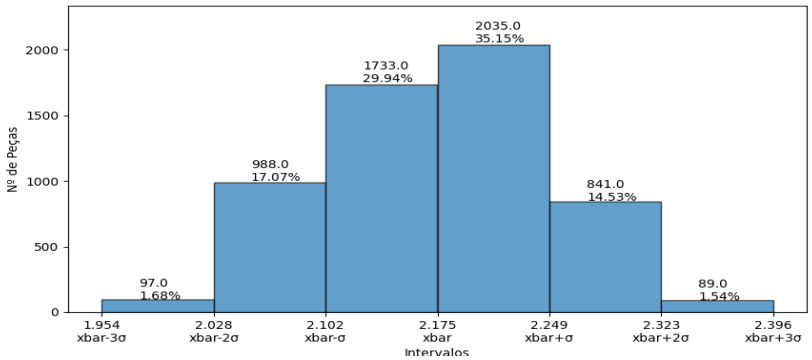
Mean: 2,1753 [V] Std Dev: 0,0737 [V] Coefficient of variation: 3,4000000000000004 %

Mean + std.Dev: 2,249 [V] Mean + 2std.Dev: 2,323 [V] Mean + 3std.Dev: 2,396 [V]
Mean - std.Dev: 2,102 [V] Mean - 2std.Dev: 2,028 [V] Mean - 3std.Dev: 1,954 [V]

Mean + std.Dev: 2035 Pieces Mean + 2std.Dev: 841 Pieces Mean + 3std.Dev: 89 Pieces
Mean - std.Dev: 1733 Pieces Mean - 2std.Dev: 988 Pieces Mean - 3std.Dev: 97 Pieces

Mean ± std.Dev: 3768 Pieces Mean ± 2std.Dev: 1829 Pieces Mean ± 3std.Dev: 186 Pieces
Mean ± std.Dev: 65.224 % Mean ± 2std.Dev: 31.66 % Mean ± 3std.Dev: 3.22 %
Outside Mean ± 3std.Dev: 186 Pieces 3.22 %





Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

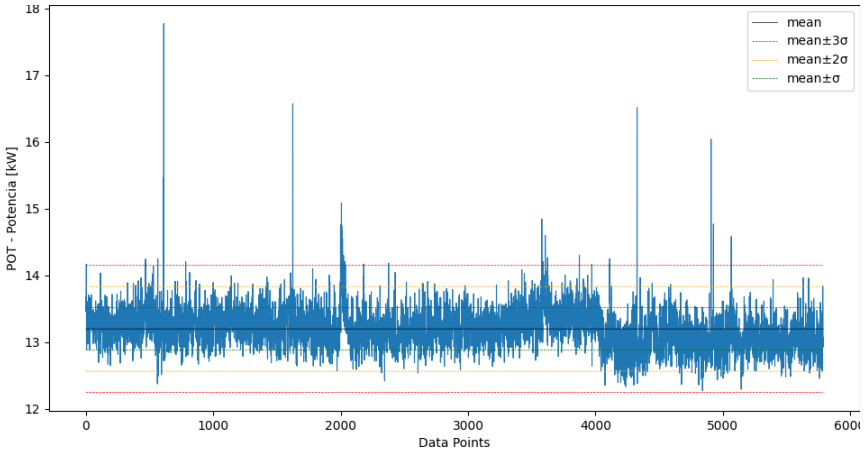
POT - Potencia [kW]

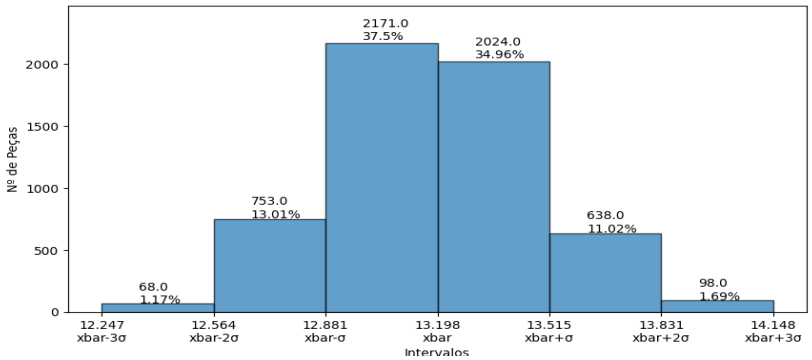
Mean: 13,1975 [kW] Std Dev: 0,317 [kW] Coefficient of variation: 2,4 %

Mean + std.Dev: 13,515 [kW] Mean + 2std.Dev: 13,831 [kW] Mean + 3std.Dev: 14,148 [kW]
Mean - std.Dev: 12,881 [kW] Mean - 2std.Dev: 12,564 [kW] Mean - 3std.Dev: 12,247 [kW]

Mean + std.Dev: 2024 Pieces Mean + 2std.Dev: 638 Pieces Mean + 3std.Dev: 98 Pieces
Mean - std.Dev: 2171 Pieces Mean - 2std.Dev: 753 Pieces Mean - 3std.Dev: 68 Pieces

Mean ± std.Dev: 4195 Pieces Mean ± 2std.Dev: 1391 Pieces Mean ± 3std.Dev: 166 Pieces
Mean ± std.Dev: 72.616 % Mean ± 2std.Dev: 24.078 % Mean ± 3std.Dev: 2.873 %
Outside Mean ± 3std.Dev: 166 Pieces 2.873 %





Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

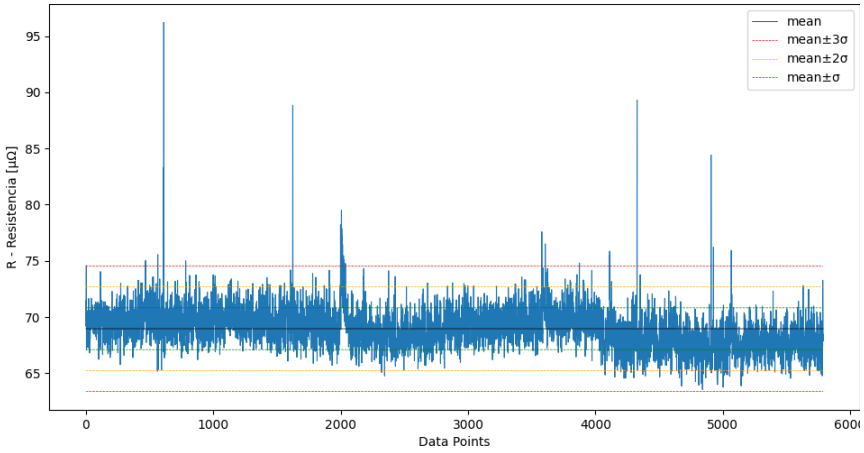
R - Resistencia

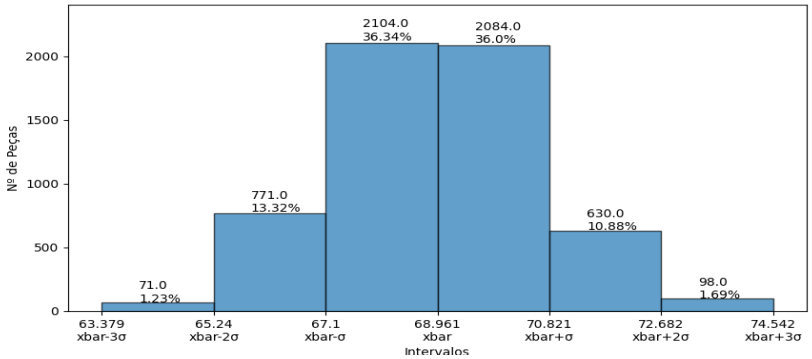
Mean: 68,9606 [µOhm] Std Dev: 1,8605 [µOhm] Coefficient of variation: 2,7 %

Mean + std.Dev: 70,821 [µOhm] Mean + 2std.Dev: 72,682 [µOhm] Mean + 3std.Dev: 74,542 [µOhm]
Mean - std.Dev: 67,1 [µOhm] Mean - 2std.Dev: 65,24 [µOhm] Mean - 3std.Dev: 63,379 [µOhm]

Mean + std.Dev: 2084 Pieces Mean + 2std.Dev: 630 Pieces Mean + 3std.Dev: 98 Pieces
Mean - std.Dev: 2104 Pieces Mean - 2std.Dev: 771 Pieces Mean - 3std.Dev: 71 Pieces

Mean ± std.Dev: 4188 Pieces Mean ± 2std.Dev: 1401 Pieces Mean ± 3std.Dev: 169 Pieces
Mean ± std.Dev: 72.494 % Mean ± 2std.Dev: 24.251 % Mean ± 3std.Dev: 2.925 %
Outside Mean ± 3std.Dev: 169 Pieces 2.925 %





Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

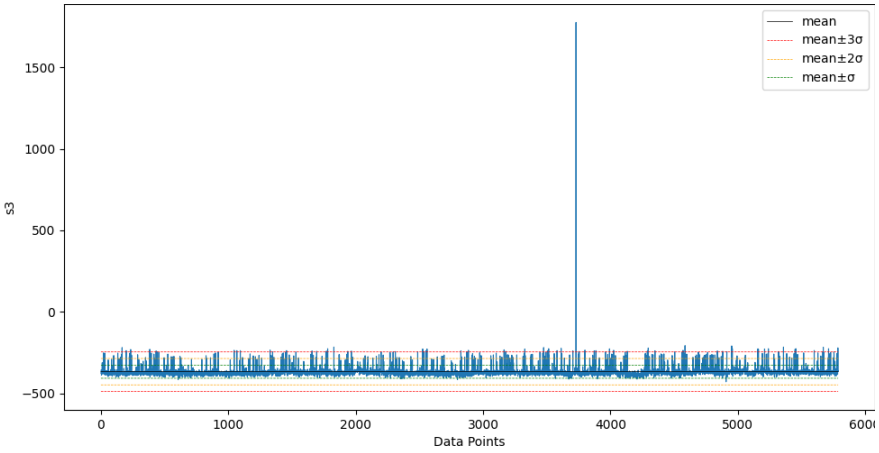
Deslocamento s3

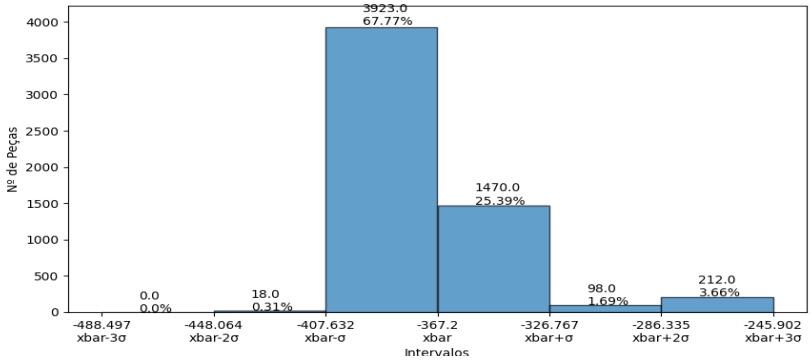
Mean: -367,1995 [µm] Std Dev: 40,4324 [µm] Coefficient of variation: -11,0 %

Mean + std.Dev: -326,767 [µm] Mean + 2std.Dev: -286,335 [µm] Mean + 3std.Dev: -245,902 [µm]
Mean - std.Dev: -407,632 [µm] Mean - 2std.Dev: -448,064 [µm] Mean - 3std.Dev: -488,497 [µm]

Mean + std.Dev: 1470 Pieces Mean + 2std.Dev: 98 Pieces Mean + 3std.Dev: 212 Pieces
Mean - std.Dev: 3923 Pieces Mean - 2std.Dev: 18 Pieces Mean - 3std.Dev: 0 Pieces

Mean ± std.Dev: 5393 Pieces Mean ± 2std.Dev: 116 Pieces Mean ± 3std.Dev: 212 Pieces
Mean ± std.Dev: 93.353 % Mean ± 2std.Dev: 2.008 % Mean ± 3std.Dev: 3.67 %
Outside Mean ± 3std.Dev: 212 Pieces 3.67 %





KEMET a YAGEO company

Resistance Welding - Report

Generated: 12:44:11 15-03-2023

Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

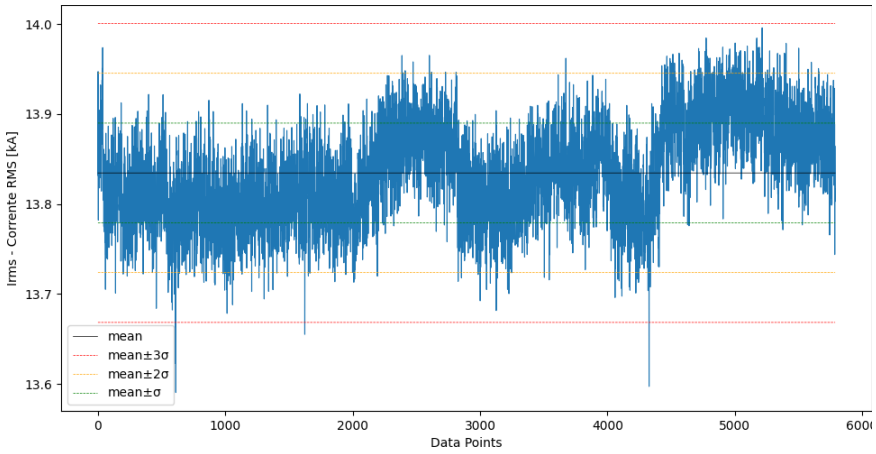
Irms - Corrente RMS [kA]

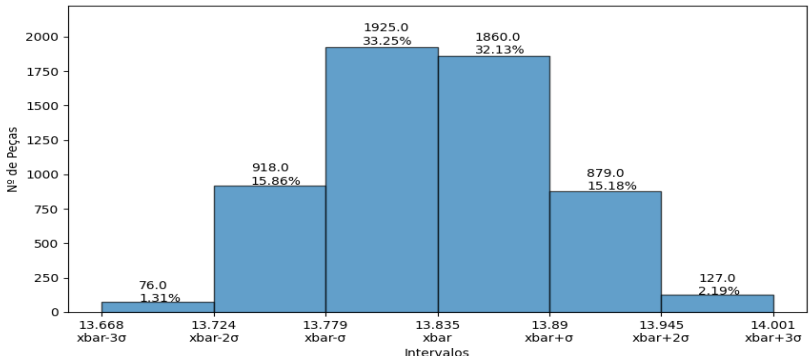
Mean: 13,8346 [kA] Std Dev: 0,0554 [kA] Coefficient of variation: 0,4 %

Mean + std.Dev: 13,89 [kA] Mean + 2std.Dev: 13,945 [kA] Mean + 3std.Dev: 14,001 [kA]
Mean - std.Dev: 13,779 [kA] Mean - 2std.Dev: 13,724 [kA] Mean - 3std.Dev: 13,668 [kA]

Mean + std.Dev: 1860 Pieces Mean + 2std.Dev: 879 Pieces Mean + 3std.Dev: 127 Pieces
Mean - std.Dev: 1925 Pieces Mean - 2std.Dev: 918 Pieces Mean - 3std.Dev: 76 Pieces

Mean ± std.Dev: 3785 Pieces Mean ± 2std.Dev: 1797 Pieces Mean ± 3std.Dev: 203 Pieces
Mean ± std.Dev: 65.518 % Mean ± 2std.Dev: 31.106 % Mean ± 3std.Dev: 3.514 %
Outside Mean ± 3std.Dev: 203 Pieces 3.514 %





KEMET a YAGEO company

Resistance Welding - Report

Generated: 12:44:11 15-03-2023

Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

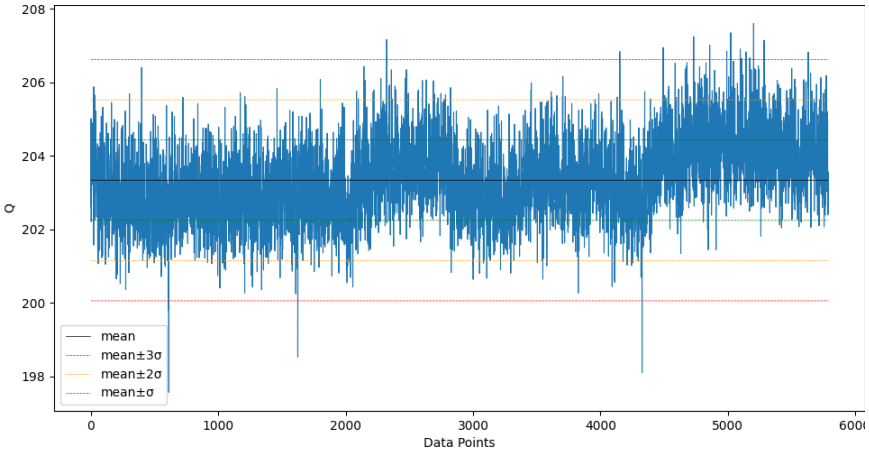
Q - Ampere por segundo

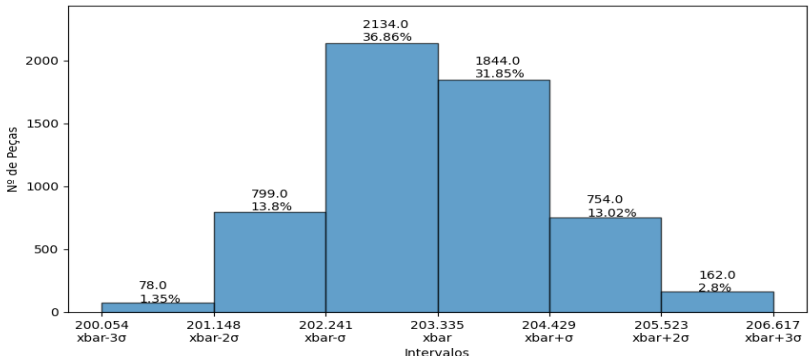
Mean: 203,3354 [As] Std Dev: 1,0939 [As] Coefficient of variation: 0,5 %

Mean + std.Dev: 204,429 [As] Mean + 2std.Dev: 205,523 [As] Mean + 3std.Dev: 206,617 [As]
Mean - std.Dev: 202,241 [As] Mean - 2std.Dev: 201,148 [As] Mean - 3std.Dev: 200,054 [As]

Mean + std.Dev: 1844 Pieces Mean + 2std.Dev: 754 Pieces Mean + 3std.Dev: 162 Pieces
Mean - std.Dev: 2134 Pieces Mean - 2std.Dev: 799 Pieces Mean - 3std.Dev: 78 Pieces

Mean ± std.Dev: 3978 Pieces Mean ± 2std.Dev: 1553 Pieces Mean ± 3std.Dev: 240 Pieces
Mean ± std.Dev: 68.859 % Mean ± 2std.Dev: 26.882 % Mean ± 3std.Dev: 4.154 %
Outside Mean ± 3std.Dev: 240 Pieces 4.154 %





Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

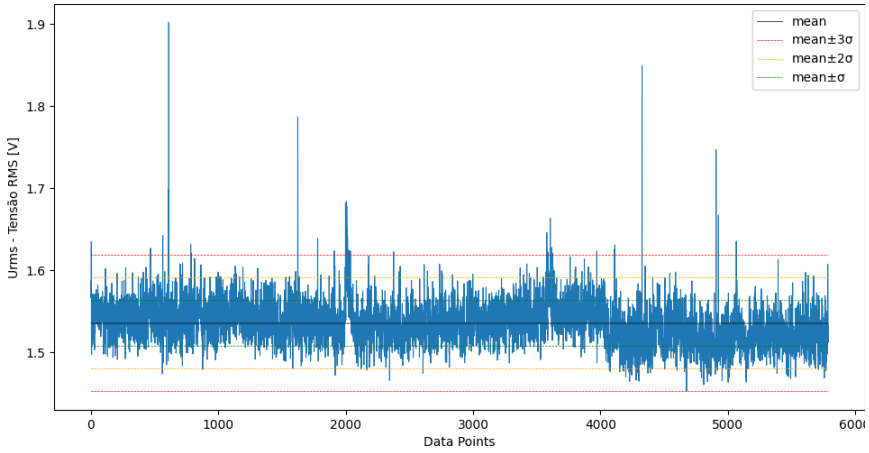
Urms - Tensão RMS [V]

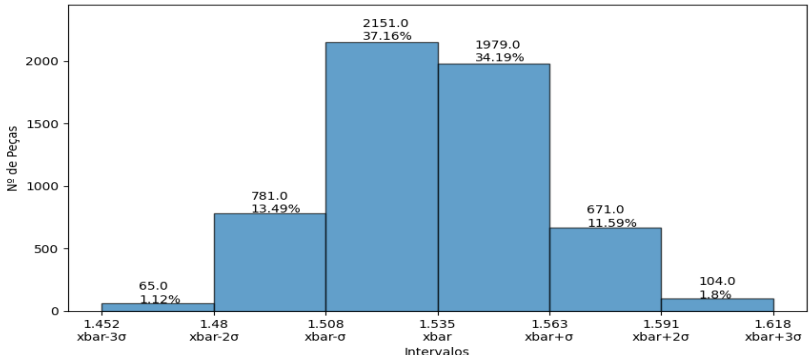
Mean: 1,5354 [V] Std Dev: 0,0277 [V] Coefficient of variation: 1,7999999999999998 %

Mean + std.Dev: 1,563 [V] Mean + 2std.Dev: 1,591 [V] Mean + 3std.Dev: 1,618 [V]
Mean - std.Dev: 1,508 [V] Mean - 2std.Dev: 1,48 [V] Mean - 3std.Dev: 1,452 [V]

Mean + std.Dev: 1979 Pieces Mean + 2std.Dev: 671 Pieces Mean + 3std.Dev: 104 Pieces
Mean - std.Dev: 2151 Pieces Mean - 2std.Dev: 781 Pieces Mean - 3std.Dev: 65 Pieces

Mean ± std.Dev: 4130 Pieces Mean ± 2std.Dev: 1452 Pieces Mean ± 3std.Dev: 169 Pieces
Mean ± std.Dev: 71.49 % Mean ± 2std.Dev: 25.134 % Mean ± 3std.Dev: 2.925 %
Outside Mean ± 3std.Dev: 169 Pieces 2.925 %





Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

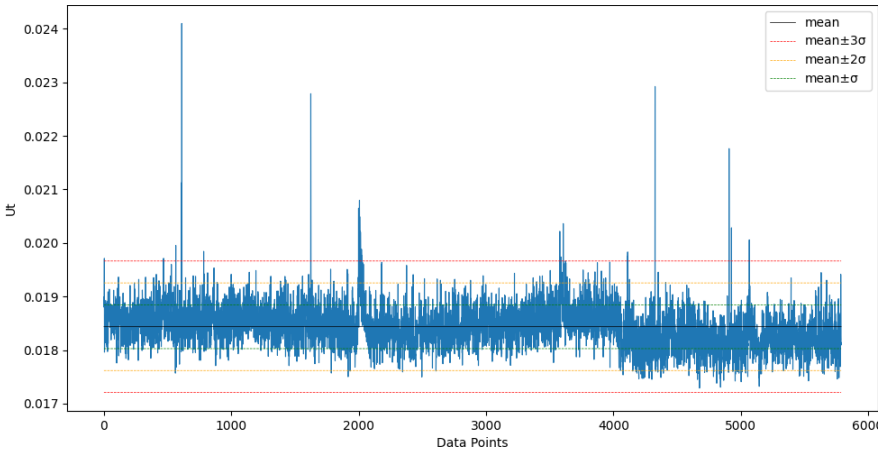
Ut - tensão por segundo

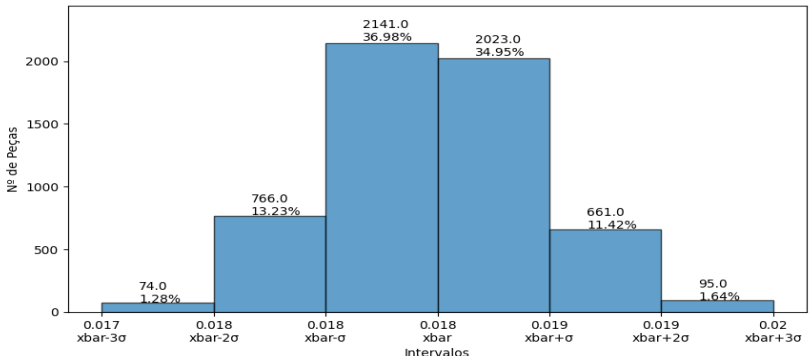
Mean: 0,0184 [Vs] Std Dev: 0,0004 [Vs] Coefficient of variation: 2,1999999999999997 %

Mean + std.Dev: 0,019 [Vs] Mean + 2std.Dev: 0,019 [Vs] Mean + 3std.Dev: 0,02 [Vs]
Mean - std.Dev: 0,018 [Vs] Mean - 2std.Dev: 0,018 [Vs] Mean - 3std.Dev: 0,017 [Vs]

Mean + std.Dev: 2023 Pieces Mean + 2std.Dev: 661 Pieces Mean + 3std.Dev: 95 Pieces
Mean - std.Dev: 2141 Pieces Mean - 2std.Dev: 766 Pieces Mean - 3std.Dev: 74 Pieces

Mean ± std.Dev: 4164 Pieces Mean ± 2std.Dev: 1427 Pieces Mean ± 3std.Dev: 169 Pieces
Mean ± std.Dev: 72.079 % Mean ± 2std.Dev: 24.701 % Mean ± 3std.Dev: 2.925 %
Outside Mean ± 3std.Dev: 169 Pieces 2.925 %





KEMET a YAGEO company

Resistance Welding - Report

Generated:
12:44:11 15-03-2023

Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777
Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

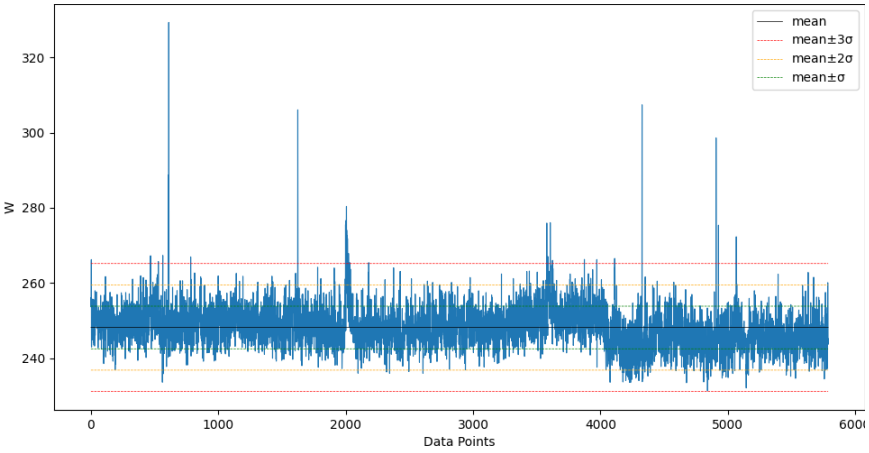
Ws - potencia por segundo

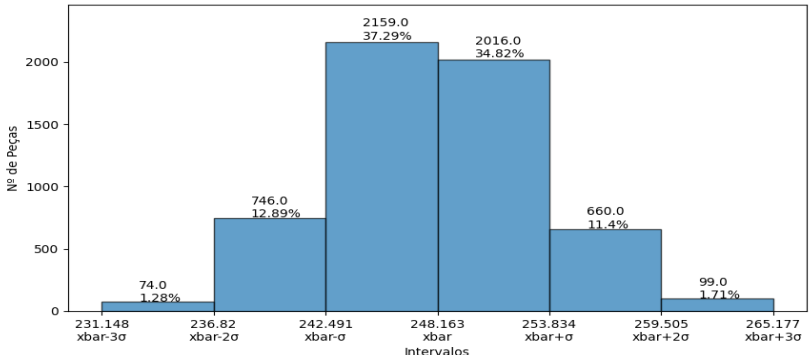
Mean: 248,1625 [Ws] Std Dev: 5,6714 [Ws] Coefficient of variation: 2,3 %

Mean + std.Dev: 253,834 [Ws] Mean + 2std.Dev: 259,505 [Ws] Mean + 3std.Dev: 265,177 [Ws]
Mean - std.Dev: 242,491 [Ws] Mean - 2std.Dev: 236,82 [Ws] Mean - 3std.Dev: 231,148 [Ws]

Mean + std.Dev: 2016 Pieces Mean + 2std.Dev: 660 Pieces Mean + 3std.Dev: 99 Pieces
Mean - std.Dev: 2159 Pieces Mean - 2std.Dev: 746 Pieces Mean - 3std.Dev: 74 Pieces

Mean ± std.Dev: 4175 Pieces Mean ± 2std.Dev: 1406 Pieces Mean ± 3std.Dev: 173 Pieces
Mean ± std.Dev: 72.269 % Mean ± 2std.Dev: 24.338 % Mean ± 3std.Dev: 2.995 %
Outside Mean ± 3std.Dev: 173 Pieces 2.995 %





Report based on:

Machine: CW9 Nº Lote: 22502092 Part Number: PEH226MH347KQE4 Nº Logs: 5777

Item Crown: PYC7131 Nº Lote Crown: PT930 Ref. Eletrode: PYC7131

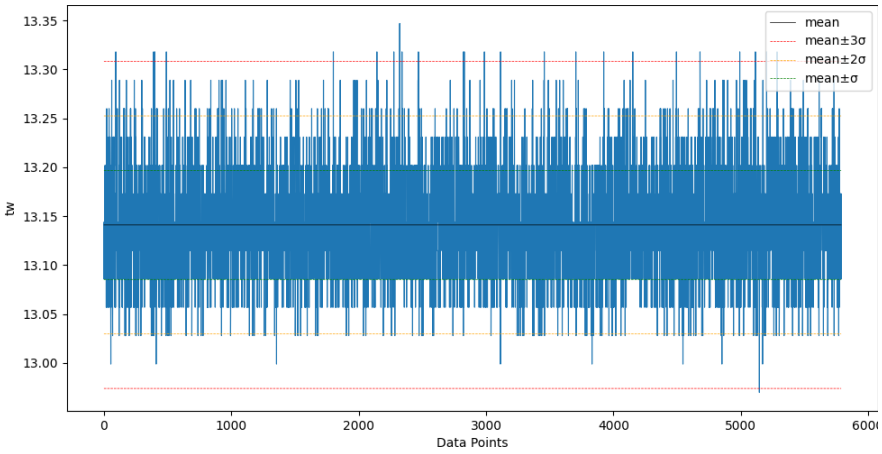
Duração soldadura

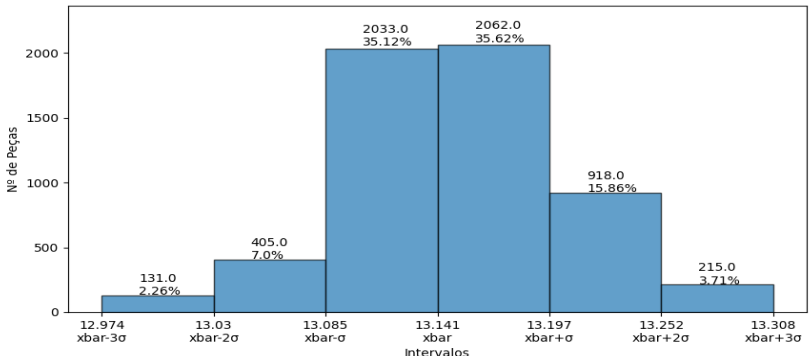
Mean: 13,1411 [ms] Std Dev: 0,0557 [ms] Coefficient of variation: 0,4 %

Mean + std.Dev: 13,197 [ms] Mean + 2std.Dev: 13,252 [ms] Mean + 3std.Dev: 13,308 [ms]
Mean - std.Dev: 13,085 [ms] Mean - 2std.Dev: 13,03 [ms] Mean - 3std.Dev: 12,974 [ms]

Mean + std.Dev: 2062 Pieces Mean + 2std.Dev: 918 Pieces Mean + 3std.Dev: 215 Pieces
Mean - std.Dev: 2033 Pieces Mean - 2std.Dev: 405 Pieces Mean - 3std.Dev: 131 Pieces

Mean ± std.Dev: 4095 Pieces Mean ± 2std.Dev: 1323 Pieces Mean ± 3std.Dev: 346 Pieces
Mean ± std.Dev: 70.885 % Mean ± 2std.Dev: 22.901 % Mean ± 3std.Dev: 5.989 %
Outside Mean ± 3std.Dev: 346 Pieces 5.989 %





Appendix W: Função machine_performance_w

Listing W.1: Codigo da função machine_performance_w

```
#funcao responsavel por pagina para calcular o desempenho de um equipamento num
    determinado intervalo de tempo
# uma data final e uma inicial
def machine_performance_w():

    #declarar variaveis globais em uso
    global machine_list
    global path_db

    #variavel para permitir introduzir equipamento
    lista_maquinas_d=[]

    #criar janela, definir titulo
    machineperfomance_wi= Toplevel()
    machineperfomance_wi.title("Machine performance")#alterar titulo

    #criar objecto de calendario na janela secundaria e poscionar no ecra
    calendario = Calendar(machineperfomance_wi,selectmode="day")
    calendario.grid(row = 0, column=3,pady=4, padx=5,rowspan=4)

    #contruir lista com equipamentos
    for items in machine_list:
        lista_maquinas_d.append(items[0].upper())

    #criar varaivel do tkinter para guardar o equipamento
    clicked_maq_d = StringVar()

    #introduzir etiqueta a intidicar que o menu de seleo  o equipamento
    Label(machineperfomance_wi,text="Maquina: ").grid(row = 0, column=0,pady=4, padx=5)

    #definir item padro do menu colapsavel
    clicked_maq_d.set(lista_maquinas_d[0])
```

```
#construir menu colaspavel
maq_drop=OptionMenu(machineperfomance_wi, clicked_maq_d,*lista_maquinas_d )
maq_drop.grid(row = 0, column=1,pady=4, padx=5)

#definir lista com intervalos de tempo possiveis
lista_deltat=[5,10,15,20,30,60]

#definir variavel para guardar resultado da caixa de selecao do intervalo de tempo
clicked_deltat= IntVar()

#definir intervalo de tempo padrao
clicked_deltat.set(lista_deltat[3])

#criar menu colapsavel com opoe possiveis de intervals de tempo
drop_deltat = OptionMenu(machineperfomance_wi,clicked_deltat, *lista_deltat)
drop_deltat.grid(row = 3, column=1,pady=4, padx=5)

#etiqueta a informar que o menu de intervalo de tempo  em minutos
Label(machineperfomance_wi,text="minutos").grid(row = 3, column=2,pady=4, padx=5)
Label(machineperfomance_wi,text="Intervalo em minutos:").grid(row = 3,
    column=0,pady=4, padx=5)

#etiquetas para a caixa de introduo da data inicial
Label(machineperfomance_wi,text="Data Inicial:").grid(row = 1 , column=0,pady=4,
    padx=5)

#implementar caixa de entrada da data inicial
data_inicial = Entry(machineperfomance_wi,borderwidth=2,width=15)
data_inicial.grid(row = 1, column=1,pady=4, padx=5)

#imlplementar botao que executa funcao de inserir dados do calendario na entrada da
    data inicial
b_data_i=Button(machineperfomance_wi, text="Selecionar data",command=b_data_i_click)
b_data_i.grid(row = 1, column=2,pady=4, padx=5)


#implementar caixa de entrada da data final
Label(machineperfomance_wi,text="Data Final:").grid(row = 2, column=0,pady=4, padx=5)
data_final = Entry(machineperfomance_wi,borderwidth=2,width=15)
data_final.grid(row = 2, column=1,pady=4, padx=5)

#imlplementar botao que executa funcao de inserir dados do calendario na entrada da
    data final
b_data_f=Button(machineperfomance_wi, text="Selecionar data",command=b_data_f_click)
b_data_f.grid(row = 2, column=2,pady=4, padx=5)

#imlplementar botao que executa funcao de inserir dados do calendario na entrada da
    data final
b_gerar=Button(machineperfomance_wi, text="Gerar",width=15,command=b_gerar_click)
```

```
    b_gerar.grid(row = 3, column=2,pady=4, padx=5)
```

Listing W.2: Codigo da função b_data_i_click

```
# funo  responsavel por inserir os dados do calenadario na campo da data inicial
    def b_data_i_click():

        #eliminar o que estiver presente na entrada
        data_inicial.delete(0, END)

        #obter data do calendario e por no formato de um objeto datetime
        var_date= datetime.strptime(calendario.get_date(), '%m/%d/%y')

        #inserir a data obtida a transformando o formato para o compativel com a base de
            dados
        data_inicial.insert(0, var_date.strftime('%d-%m-%Y'))
```

Listing W.3: Codigo da função b_data_f_click

```
    # funo  responsavel por inserir os dados do calenadario na campo da data final
    def b_data_f_click():

        #eliminar o que estiver presente na entrada
        data_final.delete(0, END)

        #obter data do calendario e por no formato de um objeto datetime
        var_date= datetime.strptime(calendario.get_date(), '%m/%d/%y')

        #inserir a data obtida a transformando o formato para o compativel com a base de
            dados
        data_final.insert(0, var_date.strftime('%d-%m-%Y'))
```

Listing W.4: Codigo da função b_gerar_click

```
# funo  executada quando o boto gerar  pressionado
    #Vai verificar as datas introduzidas e fazer as consultas necessarias  base de dados
        para obter informao para essas daras, em
    def b_gerar_click():

        #obtem a data inicial e final a gerar graficos atraves das datas presentes nas
            entradas
        data_inicial_val =data_inicial.get()
        data_final_val = data_final.get()

        #iniciar  ligao    base de dados e criar cursor para a ligao
        conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db")
        cursor_db = conn_db.cursor()

        #Se entradas forem validas e data final > data inicial
```

```
if (str(data_inicial_val) != str(data_final_val)) and str(data_inicial_val) != ''
    and str(data_final_val) != '' and datetime.strptime(data_final_val,'%d-%m-%Y')
    > datetime.strptime(data_inicial_val,'%d-%m-%Y'):

    # calcular intervalo de datas a analisar
    start_date= datetime.strptime(data_inicial_val,'%d-%m-%Y')
    end_date = datetime.strptime(data_final_val,'%d-%m-%Y')

    #diferena  entre as datas
    delta = end_date - start_date

    #criar lista para guardar todas as datas entre a data incial e final
    dates_list=[]

    #ciclo para obter todas as datas nesse periodo
    for i in range(delta.days + 1):
        #metodo timedelta com argumeto days para aumentar a data
        day = start_date + timedelta(days=i)

        #guardar em formato de texto na lista criada
        dates_list.append(day.strftime('%d-%m-%Y'))


    #ciclo para verificar cada data presente na lista das datas
    for item in dates_list:

        #criar consulta  base de dados para a data
        sql_query= "SELECT * FROM "+str(clicked_maq_d.get()).lower()+" WHERE DATE =
            '"+str(item) +"'"

        #executar consulta
        cursor_db.execute(sql_query)

        #guardar resultados em varivel
        to_add=cursor_db.fetchall()

        #print de debug a informar resultados encontrados
        print("Items resultantes da query: ",len(to_add))

        #se forem encontrados resultados incial-se a distribuidao por datas
        if len(to_add)> 0:

            #metodo ordenar lista atraves da data de registo
            data_sorted = sorted(to_add, key=lambda t: datetime.strptime(t[8],
                '%H:%M:%S'))

            #obter intervalo de tempo introduzido na caixa de selecao com metodo de
                get sobre objecto da caixa
            delta_t_get = clicked_deltat.get()
```

```
#criar variavel para guardar dados
data_y=[]
dates_x=[]

#Definir data inicial do intervalo a verificar
startdate = datetime(int(data_sorted[0][7][-4:]),
    int(data_sorted[0][7][3:5]), int(data_sorted[0][7][:2]), int(0),
    int(0), int(0))

#data final do intervalo, ou seja, data a cima mais 5 minutos por exemplo
enddate= startdate+ timedelta(minutes=delta_t_get)

#definir numero de ciclos necessarios para realizar as 24 horas do dia
range_t=int((60/delta_t_get)*24)

#assim, para cada ciclo necessario
for i in range(range_t):

    #lista temporaria onde registos do intervalo vao estar guardados
    temp_list=[]

    #verificar se cada registo  valido para o intervalo descrito
    for registo in data_sorted:

        #se for adicionamos a lista temporaria
        if startdate <= datetime(int(registo[7][-4:]),
            int(registo[7][3:5]), int(registo[7][:2]),
            int(registo[8][:2]), int(registo[8][3:5]),
            int(registo[8][-2:])) < enddate :

            temp_list.append(item)

        else:
            pass

    #   guardada data a que a lista temporaria corresponde
    dates_x.append(startdate.strftime("%H:%M:%S"))

    #incrementamos o intervalo de tempo, definindo a data inicial como a
        final e adicionando o intervalo de tempo  data final
    startdate = enddate
    enddate= startdate+ timedelta(minutes=delta_t_get)

    #adicionar lista temporaria com registos no intervalo de tempo em
        lista com todos os conjuntos
    data_y.append(temp_list)


#criar variaveis para guardar dados para o grafico
data_y_mean=[]
```

```
                soma_n_items= 0

                #para cada conjunto de registos de um intervalo de tempo
                for items in data_y:

                    #calculada a media de peas nesse intervalo de tempo, ou seja peas/m
                    data_y_mean.append(len(items)/delta_t_get)

                    #incrementar variavel para grafico
                    soma_n_items= soma_n_items+len(items)

                #executa  funo  que constroi os graficos
                plot_performace((dates_x,data_y_mean),(str(clicked_maq_d.get()),str(item),soma_n_items),delta


        #se a data tiver num formato errado mostrar mensagem de erro
        else:
               messagebox.showerror("Erro", "Erro na formatao data ")
```

Appendix X: Função configs_change

Listing X.1: Codigo da função configs_change

```
#Funcao responsavel por criar objectos na pagina de alterar caminhos para a base de dados
def config_change():

    #funcao que insere na entrada o caminho atual para a base de dados
    def change_entry():

        #ontem equipamento selecionado
        machine=eq_select.get()

        #verifica lista de todos os equipamentos
        for item in machine_list:

            #quando equipamento  encontrado, o caminho associado  inserido na entrada
            if item[0].lower() == machine.lower():

                e_caminho_logs.delete(0,END)
                e_caminho_logs.insert(0,item[1])

    #Funcao com o objectivo de abrir um explorador de ficherios do windows para seja
        possivel selecionar a localizao desejada
    #para o ficheiros de registos de soldadura
    def file_dialog():
        folder = filedialog.askdirectory(title = "Selecionar Pasta")
        e_caminho_logs.delete(0,END)
        e_caminho_logs.insert(0,folder)

    #Declarar varaveis globais em uso
    global path_app
    global machine_list
    global machine_list
    global e_linklogs
    global eq_select
```

```
    #mudar ambiente de trabalho para diretoria do programa
    os.chdir(path_app)

    #declarar varaivel para guardar equipamento selecionado
    eq_select=StringVar()

    #criar a janela secundaria e alterar o titulo
    config_change_wi= Toplevel()
    config_change_wi.title("Editar Equipamentos")#alterar titul

    #criar e lista de equipamentos para utilizar em caixa de seleo multipla
    lista_drop=[]

    for eq in machine_list:
        lista_drop.append(eq[0].upper())

    #definir valor padrao da caixa de selecao multipla
    eq_select.set(lista_drop[0])
    select_drop=OptionMenu(config_change_wi, eq_select,*lista_drop )
    select_drop.grid(row = 0, column=1,padx=10,pady=10)

    #criar etiqueta a indicar campo onde deve ser introduzido o equipamento e o caminho
        para os ficheiros de registo
    Label(config_change_wi,text="Equipamento: ").grid(row=0,column=0,padx=5,pady=2)
    Label(config_change_wi,text="Caminho Logs").grid(row=1,column=0,padx=5,pady=2)

    #criar e posciioanr entrada de texto onde deve ser introduzido o caminho
    e_caminho_logs = Entry(config_change_wi,borderwidth=2,width=50)
    e_caminho_logs.grid(row=1,column=1,padx=5,pady=2)

    #criar e posciionar boto que introduzir na entrada de texto o caminho atual para o
        equipamento que estiver selecionado
    b_atual= Button(config_change_wi,text="Ver Caminho\nAtual",command=change_entry)
    b_atual.grid(row=3,column=0,padx=5,pady=10,columnspan=1)

    #criar e poscionar boto para guardar  alteraes  efetuadas
    b_save= Button(config_change_wi,text="Guardar",command=save_config_change)
    b_save.grid(row=3,column=2,padx=5,pady=10,columnspan=1)

    #criar e poscionar boto para abrir explorador de ficheiros do windows
    b_file_dialog = Button(config_change_wi,text="Procurar\nPasta",command=file_dialog)
    b_file_dialog.grid(row=3,column=1,padx=5,pady=10,columnspan=1)
```

Listing X.2: Codigo da função machine_list_stats

```
# Funo  que lista os equipamentos assim como o caminho para o ficheir de logs
#Verifica estado do acesso ao caminho
def machine_list_stats():

    #Declarar variaveis globais em usp
```

```
global machine_list
global path_db
global path_app


# construo  de janela secundaria, mudanca de nome da janela
machine_list_wi= Toplevel()
machine_list_wi.title("Listar Equipamentos")#alterar titulo

#Criar etiquetas a indicar equipamento e caminho para ficheiro de registos de soldadura
Label(machine_list_wi,text="Equipamento").grid(row=0,column=0,padx=5,pady=2)
Label(machine_list_wi,text="Caminho LOGS").grid(row=0,column=1,padx=5,pady=2)

#varaivel para usar no posciionamento das coisas na janela
i = 1

#definir variavel que vai verificar se consegue acedeer a base de dados
status="ok"
try:
    os.chdir(path_db)
    status="OK"
except:
    status="NOK"

#Etiqueta a indicar que o caminho para a base de dados
Label(machine_list_wi,text="Caminho Base de Dados:
    ").grid(row=i,column=0,padx=5,pady=2)

#criar etiquetas para resultado de tentativa de acesso  localizacao da base de dados
Label(machine_list_wi,text=str(path_db)).grid(row=i,column=1,padx=5,pady=2)

# Construo  de mensagem indicadora de ok/nok ao acesso, if usado para determinar cor
    aplicavel
if status =="NOK":
    Label(machine_list_wi,text=status,fg='red').grid(row=i,column=2,padx=5,pady=2)
else:
    Label(machine_list_wi,text=status,fg='green').grid(row=i,column=2,padx=5,pady=2)
i = i +1

#Para cada equipamento presente da lista deve ser feita esta verificao
for item in machine_list:

    #tentar aceder a  localizao  do ficheiro
    try:
        os.chdir(item[1])
        status="OK"
    #Caso  no  seja possivel aceder   localizao  do ficheiro
    except:
        status="NOK"
```

```
        #Faz print do nome da maquina em caps
        Label(machine_list_wi,text=item[0].upper()).grid(row=i,column=0,padx=5,pady=2)

        #diz o endereo da BD associada
        Label(machine_list_wi,text=item[1]).grid(row=i,column=1,padx=5,pady=2)

        #cria etiqueta vermelha ou verde conforme o resultado
        if status =="NOK":
            Label(machine_list_wi,text=status,fg='red').grid(row=i,column=2,padx=5,pady=2)
        else:
            Label(machine_list_wi,text=status,fg='green').grid(row=i,column=2,padx=5,pady=2)
        i = i +1

    #voltar para ambiente de trabalho normal da app
    os.chdir(path_app)
```

Listing X.3: Codigo da função adicionar_equipamento

```
#Extrai os dados do ficheiro config.csv e atribui as variaveis globais
#Guarda variavel com caminho para o programa
def adicionar_equipamento():

    #declarar variaveis globais em uso
    global machine_list
    global e_nomemaquina
    global e_linklogs
    global configs_list

    #guardar equipamento na entrada em varaivel
    machine_add=e_nomemaquina.get().lower()

    #guardar caminho introduzido na entrada em variavel
    link_logs_add= e_linklogs.get()

    #ciclo para ir buscar os nomes de todas as maquinas
    list_compare_machine=[]

    #cria lista para comparar nome
    for item in machine_list:
        list_compare_machine.append(item[0])

    #Verifica se equipamento ja se esta na base de dados
    if machine_add in list_compare_machine:

        msg_erro= "O equipamento: "+machine_add+" j se encontra na base de dados"
        messagebox.showerror("Erro", msg_erro)
```

```
    #caso nao esteja, perguntar se utilizador pretende mesmo adicionar equipamento
    else:

        #se for pressionado sim
        if (messagebox.askquestion("askquestion", "Are you sure?")) == "yes":

            #adiciona equipamento a lista global de equipamentos
            machine_list.append((machine_add,link_logs_add))

            # Alterao  do ficheiro de  configuraes  configs.txt
            with open("configs.txt", "w", newline="") as file:

                csvwrite = csv.writer(file,delimiter=",")

                csvwrite.writerows(configs_list)

            #executa  funo  que cria nova tabela na base de ados
            create_table(machine_add)
```

Listing X.4: Codigo da função create_table

```
#funcao responsavel por criar novo equipamento na base de dados
def create_table(machine):

    #variaveis globais em uso
    global path_db
    global path_app


    #iniciar  ligao     base de dados e cursor
    conn_db = sqlite3.connect(path_db+"\\"+"RW_DB.db")
    cursor_db = conn_db.cursor()

    #criar consulta a executar com criao de nova tabela para o equipamento
    query_add_table="CREATE TABLE "+str(machine).lower()+""" (

                N_lote text,
                DESCRIPTION text,
                PART_NUM text,
                ITEM_CROWN text,
                LOT_CROWN text,
                I_SET text,
                ELECTRODE_REF text,
                DATE text,
                TIME text,
                COUNTER INTEGER,
                PROGRAM text,
                Ipk text,
```

```
                Upk text,
                POT text,
                R text,
                s1 text,
                s3 text,
                F text,
                p text,
                Irms text,
                Q text,
                Urms text,
                Ut text,
                W text,
                tw text,
                twcyc text,
                AY text,
                STATUS text,
                DETAILS text,
                PROG_NAME text,
                EXT_INFO text,
                LogId INTEGER PRIMARY KEY AUTOINCREMENT

    )"""

    #Executar a consulta
    cursor_db.execute(query_add_table)

    #confirmar  alteraes
    conn_db.commit()
```

Listing X.5: Codigo da função save_config_change

```
#Funcao que guarda  alteraes  de  configuraes
def save_config_change():

    #declarar varaiveis globais em uso
    global machine_list

    #guardar caminho novo em varaivel
    new_path=str(e_caminho_logs.get()).replace("/","\\")

    #guardar equipamento selecionado em variavel
    machine=eq_select.get()

    i = 0

    #procurar equipamento selecionado na lista global de equipamentos
    for item in machine_list:

        #caso encontre, remove entrada da lista e adiciona nova entrada com novo caminho
```

```
    #como so tuplos so imutaveis
    if item[0].lower()==str(machine).lower():

        del machine_list[i]
        machine_list.append((item[0],new_path))

        break
    i=i+1


#guardar alteraes no ficheiro de configs.txt
#incluindo a informao da base de dados
configs_list = machine_list
machine_list.append(("db_RW",path_db))
with open("configs.txt", "w", newline="") as file:

    csvwrite = csv.writer(file,delimiter=",")

    csvwrite.writerows(configs_list)
```