NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE
ENGENHARIA MECÂNICA E INDUSTRIAL

TOMÁS DOS SANTOS NOVÁLIO

Licenciado em Ciências de Engenharia Mecânica

DESENVOLVIMENTO DE CÉLULA DE INSPEÇÃO ROBOTIZADA UTILIZANDO TECNOLOGIA DE CORRENTES INDUZIDAS

MESTRADO INTEGRADO EM ENGENHARIA MECÂNICA

Universidade NOVA de Lisboa
Setembro, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTAMENTO DE
ENGENHARIA MECÂNICA E INDUSTRIAL

DESENVOLVIMENTO DE CÉLULA DE INSPEÇÃO ROBOTIZADA UTILIZANDO TECNOLOGIA DE CORRENTES INDUZIDAS

TOMÁS DOS SANTOS NOVÁLIO

Licenciado em Ciências de Engenharia Mecânica

Orientador: Nuno Alberto Marques Mendes
Professor Auxiliar, FCT NOVA

Coorientador: Miguel Araújo Machado
Professor Auxiliar, FCT NOVA

Júri

Presidente: Catarina Isabel Silva Vidal
Professora Auxiliar, FCT NOVA

Arguente: Pedro Mariano Simões Neto
Professor Associado com Agregação, FCT UC

Orientador: Nuno Alberto Marques Mendes
Professor Auxiliar, FCT NOVA

MESTRADO INTEGRADO EM ENGENHARIA MECÂNICA

Universidade NOVA de Lisboa
Setembro, 2023

Desenvolvimento de célula de inspeção robotizada utilizando tecnologia de correntes induzidas

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeX e o modelo NOVAthesis (v6.10.17) [1].

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, Prof. Nuno Mendes, pelos conselhos fornecidos ao longo do desenvolvimento da dissertação. Além disso, também gostaria de agradecer ao meu co-orientador, Prof. Miguel Machado, pelos seus esclarecimentos relativamente ao tema das inspeções por Correntes Induzidas.

Devo também uma agradecimento à FCT-NOVA e a todos os professores que marcaram a minha caminhada.

Agradeço também à minha família, sem a qual o meu percurso nos últimos cinco anos nunca poderia ter tido o mesmo significado. Aos meus avós, aos meus tios e primos, ao meu pai e à minha "boadrasta", à minha mãe e ao meu irmão, agradeço todo o apoio, carinho e motivação que me ofereceram em todas as etapas, além do orgulho que demonstraram durante as minhas conquistas dentro e fora da faculdade.

No entanto, como a vida não é só feita de bons momentos, deixo um agradecimento especial ao meu irmão e à minha mãe por me terem apoiado (e sobretudo aturado) todos os dias, durante os meus períodos mais difíceis.

Além disso, gostaria de também de agradecer ao meu tio Miguel, por ter sido uma das principais razões pelas quais escolhi este curso e pela disponibilidade em partilhar comigo o seu conhecimento e a sua experiência.

Por fim, como a vida académica não é só feita de estudos, gostaria de agradecer a todos os amigos que me acompanharam.

Das amizades mais antigas, gostaria de deixar um abraço especial aos membros da Trindade, que me acompanharam nos picos mais altos e nos vales mais fundos.

Das amizades que a faculdade me proporcionou, queria, por um lado, agradecer ao Bernardo Raposo, ao José Caçador e ao João Franco, por serem uma constante motivação para ser melhor, tanto nos momentos mais tensos da faculdade, como nos mais descontraídos fora dela.

Por outro lado, não podia deixar de destacar todo pessoal do Alumina, por serem a melhor coisa que a faculdade me deu e por terem tornado inesquecível esta caminhada.

”

«Zwei Dinge sollen Kinder von ihren Eltern bekommen: Wurzeln und Flügel.»

— Johann Wolfgang von Goethe

Resumo

Nesta dissertação, apresenta-se uma metodologia para realizar uma inspeção robotizada numa superfície de geometria irregular sujeita a desalinhamentos. Utilizando controlo de força e posicional e partindo de uma posição genérica, procura-se atingir uma posição alvo pré-definida, e, seguidamente, percorrer um caminho ao longo da superfície.

Para testar a metodologia, utilizou-se um tejadilho de um automóvel e uma sonda de correntes induzidas movimentada por um robô colaborativo.

Analisaram-se o posicionamento da sonda ao longo do processo e os dados do sensor de força para avaliar a capacidade de adaptação da sonda à geometria do tejadilho e a desalinhamentos.

Os resultados revelam uma capacidade de correção de desvios na orientação da sonda segundo RY de 8° e de 5° segundo RZ (*Tool Frame*), além de desvios posicionais até 100 mm, com um tempo entre 18 s e 26 s. Também confirmam a possibilidade de percorrer o caminho utilizando controlo de força segundo Y e Z (*Tool Frame*).

Além disso, analisaram-se os dados da inspeção por correntes induzidas para estudar a viabilidade do sistema proposto na deteção de defeitos no cordão brasado. Foram detetados os defeitos superficiais produzidos, de diâmetro de $0,9$ mm e profundidades entre $0,5$ mm e $0,8$ mm.

Palavras-chave: Inspeção robotizada, controlo de força, correntes induzidas, cordão brasado

Abstract

This dissertation presents a methodology for robotized inspection of irregular surfaces prone to misalignment. Employing force and position control from an initial position, the objective is to attain a predetermined target position and subsequently navigate a surface path.

Testing involved a collaborative robot managing an eddy current probe on a car roof. Analysis encompassed probe positioning and force sensor data to evaluate its adaptability to the car roof's geometry and misalignment.

Results demonstrate the probe's capability to rectify deviations of 8° and 5° along the RY and RZ axes (Tool Frame), respectively, and correct positional deviations of up to 100 mm within 18 s to 26 s. Additionally, the study confirms the feasibility of path tracing using force control along the Y and Z axes (Tool Frame).

Furthermore, an analysis of eddy current data was conducted to assess the system's efficacy in detecting defects in laser welds. Surface defects with a diameter of $0,9$ mm and depths ranging from $0,5$ mm to $0,8$ mm were successfully identified.

Keywords: Robotized inspection, force control, eddy currents, laser weld

Índice

Índice de Figuras

Índice de Tabelas

Índice de Algoritmos

SIGLAS

END Ensaios Não Destrutivos *(pp. 1, 4, 5)*

OLP *offline programming (p. 1)*

RTDE Real-Time Data Exchange *(p. 25)*

SDK Software Development Kit *(p. 26)*

TCP Tool Center Point *(p. 20)*

1

Introdução

1.1 Motivação

A automação de processos de Ensaios Não Destrutivos (END) é algo cada vez mais prevalente, na medida em que a inspeção manual está associada a menor rapidez, repetibilidade e necessidade de operadores especializados [2], além de poder estar associada a erro humano.

Geralmente, as operações de inspeção que utilizam robôs industriais para movimentar um sensor de inspeção envolvem a geração de uma trajetória para o robô, recorrendo para isso a *offline programming* (OLP) *software* e utilizando um modelo virtual da peça a inspecionar e do ambiente em que o robô se encontra. Contudo, tal é dependente da relação entre o modelo virtual e a realidade, além de exigir o rigoroso alinhamento da peça a inspecionar [3], condições que nem sempre se verificam.

O caso investigado neste trabalho é a inspeção por correntes induzidas de uma junta brasada entre o painel lateral e o tejadilho de um automóvel [4]. O posicionamento grosseiro dos componentes do automóvel sobre uma estrutura de apoio leva a que eventuais desalinhamentos do componente no seu posicionamento possam conduzir a grandes desvios, algo que impede a utilização da mesma trajetória para todas as inspeções. Além disso, eventuais irregularidades na soldadura podem acentuar este efeito.

Este é um exemplo de um ambiente não estruturado, em que os objetos presentes no ambiente de trabalho do robô industrial não se encontram sempre na mesma localização nem posicionados com o mesmo sentido e direção. De facto, no caso referido, não existe o cuidado de garantir que cada componente é rigorosamente posicionado, algo que facilitaria a conclusão da tarefa utilizando mero controlo posicional do robô.

Desse modo, torna-se relevante a aplicação de técnicas de controlo de posição e trajetória para garantir o contacto adequado entre a sonda e o cordão, algo especialmente relevante no caso de ensaios por correntes induzidas, na medida em que o *lift-off*, a distância entre a sonda e a peça, e a orientação relativa entre ambas têm impacto nos resultados obtidos.

Tendo isso em conta, planeia-se explorar as potencialidades de controlo de força

do robô colaborativo Universal Robots UR10e, de forma a procurar garantir o contacto constante entre a sonda e a peça.

1.2 Objetivos

O objetivo proposto é o de melhorar um processo robotizado de inspeção por correntes induzidas de uma junta brasada, numa situação em que o ambiente é parcialmente desconhecido, devido a potenciais desvios no posicionamento da mesma.

Para tal, será proposta uma abordagem baseada em controlo de força, de forma a garantir o contacto entre a sonda e a junta e diminuir eventuais desvios na trajetória.

1.3 Estrutura do Documento

Segue-se uma descrição da estrutura deste documento.

No capítulo 2 é apresentado o Estado da Arte, sendo focados conceitos relativos ao controlo de força de robôs industriais e colaborativos.

Já no capítulo 3 é apresentada uma versão genérica do problema, propondo-se uma metodologia para o resolver.

De seguida, no capítulo 4, aborda-se uma estratégia para estabelecer comunicação entre os diferentes equipamentos utilizados.

Além disso, no capítulo 5, apresentam-se os testes realizados, bem como os resultados subjacentes, e discutem-se os mesmos, terminando com uma breve análise da viabilidade da solução encontrada.

Por fim, no capítulo 6 apresentam-se as conclusões que se podem retirar do trabalho efetuado, além de eventuais trabalhos futuros que podem ser levados a cabo para procurar melhorar a solução encontrada.

De notar também que existe o Anexo I, em que são fornecidos detalhes adicionais sobre os programas .urp (para um robô Universal Robots UR10e) e .py desenvolvidos, além de instruções para a sua utilização. Já no Anexo II encontra-se o programa .py desenvolvido para criação de uma interface gráfica para facilitar a comunicação entre o robô e o computador, durante a execução do programa .urp criado para realizar o ajustamento ao cordão e a inspeção ao mesmo. Finalmente, tem-se o Anexo III, em que o programa .urp referido é apresentado.

2

Estado da arte

2.1 Processo geral de inspeção

Atualmente, o método das correntes induzidas, baseado no princípio da indução eletromagnética, é amplamente estudado e utilizado na deteção de defeitos superficiais e subsuperficiais em peças cujo material seja condutor elétrico. Outras aplicações deste método são a medição de espessuras de revestimento e da condutividade elétrica de um material, que está associada à dimensão do grão e porosidade do material, ambos parâmetros relevantes para a resistência mecânica [5–7].

De forma geral, utiliza-se uma sonda constituída por uma bobina espiral helicoidal cilíndrica, que cria um campo magnético primário alternado quando percorrida por corrente elétrica alternada. Caso o material da peça a inspecionar seja condutor elétrico, o campo magnético primário vai induzir uma corrente alternada na mesma, que, por sua vez, cria um campo magnético secundário, que se opõe ao primário. Por outro lado, este campo secundário induz uma corrente na bobina que constitui a sonda [8].

Ora, um defeito na peça a inspecionar age como obstáculo à circulação da corrente alternada induzida na mesma, pelo que o campo magnético secundário criado por esta corrente terá menor intensidade do que numa situação sem defeito. Estas variações de intensidade do campo magnético secundário provocam uma mudança na impedância elétrica medida na sonda, permitindo a deteção de defeitos [8].

Como referido, este método não permite apenas detetar defeitos superficiais, na medida em que existe uma certa profundidade de penetração. Contudo, as correntes induzidas tendem a concentrar-se junto da superfície, perdendo intensidade até se anularem a uma dada profundidade. Este fenómeno é conhecido como efeito de pele. De forma a ser possível ter uma ideia de qual a profundidade a que se pode controlar uma peça, definiu-se o conceito de profundidade de penetração, δ, correspondente à profundidade em que a densidade de corrente é e^{-1}, cerca de 37 %, do valor da densidade à superfície. A profundidade de penetração, δ, pode ser calculada através da Equação 2.1, em que f é a frequência de excitação em Hz, μ é a permeabilidade magnética do material em H.m^{-1} e σ é a condutividade do material em %IACS [8].

$$\delta = \frac{1}{\sqrt{\pi f \mu \sigma}} \tag{2.1}$$

Outro aspeto a ter em conta é o *lift-off*, ou seja a distância que separa a sonda da peça. Este parâmetro tem um impacto negativo na penetração das correntes induzidas na peça. Desse modo, flutuações no *lift-off* provocam oscilações na leitura dos dados dos ensaios, que podem mascarar defeitos, ou, inclusivamente, reduzir de tal forma a intensidade das correntes induzidas que deixa de ser possível a deteção de defeitos.

Assim, é bastante útil a combinação deste método com automação, na medida em que permite aumentar a rapidez de inspeção e reduzir flutuações no *lift-off* ou velocidade de passagem da sonda, algo que pode conduzir a erros de análise [4, 9], na medida em que um robô é capaz de movimentos mais precisos e com maior repetibilidade do que um operador humano.

Um exemplo onde estas vantagens são muito interessantes é na inspeção de componentes sensíveis, como os da indústria nuclear, como abordado em [6, 10], ou da indústria aeronáutica [11, 12].

Apesar disso, em inspeções a peças de elevada complexidade ou dimensão, a automação poderá não ser possível, sendo necessário recorrer a um operador humano [9]. Um exemplo é a inspeção de *pipeline* [13], como é visível na Figura 2.1.

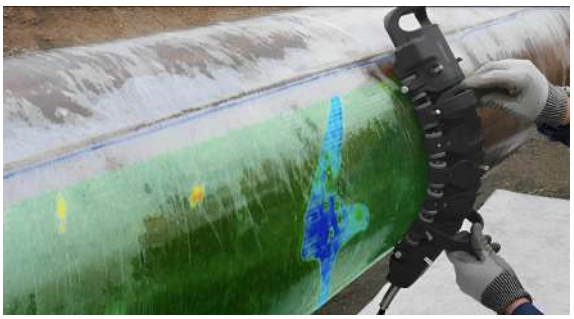


Figura 2.1: Inspeção de *pipeline* com sonda *Eddy Current Array* [14]

No que concerne especificamente à inspeção de juntas brasadas na indústria automóvel, em [4] foi analisada a viabilidade de diferentes métodos de END numa perspetiva de automatização do processo. Foram estudados, além das correntes induzidas, os métodos de radiologia (raios-x, mais especificamente), ultrassons e líquidos penetrantes. Estes outros métodos apresentam limitações sérias no que concerne à automação do processo de inspeção, entre as quais se contam a necessidade de ter acesso a ambos os lados da junta para a radiologia, a dificuldade de acoplamento de uma sonda de ultrassons a uma geometria complexa, e o facto de os líquidos penetrantes apenas permitirem detetar defeitos superficiais.

Contudo, as soluções comerciais existentes de sondas de correntes induzidas não permitem assegurar o cumprimento dos requisitos exigentes da indústria automóvel, como distinção entre defeitos superficiais e em profundidade, deteção de defeitos de

dimensão inferior a 0, 25 mm, elevada resistência ao desgaste, bem como boa relação sinal-ruído. Como tal, existem alguns exemplos de desenvolvimento de sondas customizadas, como as apresentadas nas Figuras 2.2 e 2.3.

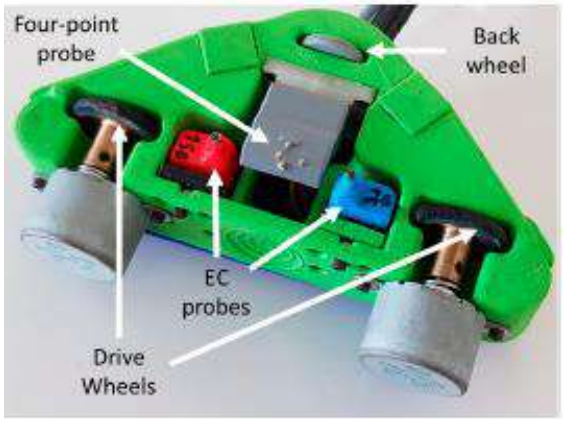


Figura 2.2: Modelo de sonda de correntes induzidas (movimento autónomo) [5]

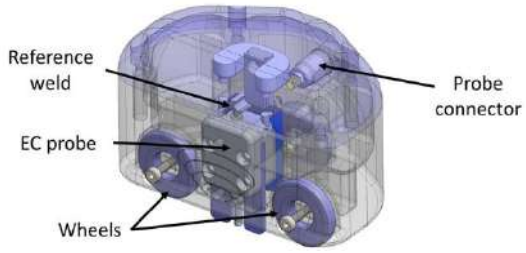


Figura 2.3: Modelo de sonda de correntes induzidas (movimento por braço robotizado) [4]

Ambas estas soluções permitem a identificação da posição do defeito ao longo da direção longitudinal do cordão, e apresentaram melhores resultados do que as sondas comerciais utilizadas como controlo na deteção de defeitos superficiais e não superficiais.

2.2 Técnicas de controlo de posição e trajetória

Nos casos em que as tarefas desempenhadas por um robô envolvem uma interação limitada e/ou simples com o ambiente em seu redor, a definição da sua operação através de mero controlo posicional é adequada. Contudo, esse nem sempre é o caso, nomeadamente para operações de acabamento em que a força de contacto é um parâmetro relevante [15] ou quando não é possível obter um modelo detalhado e preciso do ambiente, como seja em casos de END a peças de geometria variável (por efeitos de *spring back*[16, 17], por exemplo) ou quando não é possível garantir o posicionamento da peça [3]. Esse tipo de controlo oferece menor flexibilidade, por necessitar de um modelo preciso da realidade, algo que nem sempre está disponível.

É neste âmbito que surgem técnicas como o controlo de força-momento, que visa adicionar ao robô uma melhor capacidade de adaptação ao ambiente que o rodeia.

Outra alternativa para tornar a operação do robô mais flexível é a utilização de sistemas de visão, de forma a procurar gerar uma trajetória fiel à realidade [17], sobretudo nos casos em que o robô necessita de realizar trajetórias ligeiramente diferentes em cada utilização, como é o caso em ambientes não estruturados.

2.2.1 Controlo de força-momento

O controlo de força-momento é utilizado em situações nas quais as forças decorrentes do contacto entre a ferramenta do robô e o ambiente externo devem ser controladas [15].

Ora, no caso de ensaios não destrutivos como os realizados por correntes induzidas, o contacto entre a sonda e a peça é extremamente importante. Se a força de contacto for nula, a sonda não está a tocar na peça, pelo que, a não ser que o controlo posicional seja perfeito, é praticamente impossível garantir que a distância entre a sonda e a peça seja constante, algo que implica perturbações nos resultados. Por outro lado, se essa força for demasiado elevada, pode causar danos à sonda, peça, ou até ao robô, dependendo da resistência relativa destes três componentes.

Existem duas categorias de controlo de força-momento, direto e indireto, cada uma direcionada para aplicações distintas [15, 18].

2.2.1.1 Tipos de controlo de força-momento indireto

Relativamente ao controlo de força-momento indireto, este dispensa a incorporação de medições de força numa malha de controlo, centrando-se na aplicação de controlo do movimento do robô para manter a força dentro de certos parâmetros [18]. Esta categoria engloba *compliance control* e *impedance control* [18, 19].

Ora, *impedance control* consiste em tratar o *end-effector* do robô como um sistema massa-mola-amortecedor. Uma medida do desvio da trajetória prevista é convertida, através do modelo, numa estimativa de força, procurando-se agir a nível do controlo posicional do robô para manter esta força tão constante quanto possível [18].

No que concerne ao *compliance control*, este consiste em utilizar apenas a relação entre a rigidez do *end-effector* do robô e a do ambiente que o rodeia para realizar o controlo da força. Tal conduz a que apenas seja possível ter em conta desvios em termos da posição/orientação do robô, perdendo-se controlo a nível da velocidade e da aceleração ao longo da trajetória. Nas direções em que se pretende que o *end-effector* se adapte às condições do ambiente, a rigidez do segundo deve suplantar a do primeiro e vice-versa. Esta estratégia pode ser concretizada de duas formas distintas: *passive* e *active compliance* [18, 20].

No caso da *passive compliance*, este controlo é feito à base de um dispositivo mecânico entre o *end-effector* e o ambiente externo que possui uma matriz de rigidez adequada à função desejada, algo que possui como desvantagem a perda de versatilidade [18].

Já a *active compliance* envolve a utilização de um esquema de controlo estático, em que os momentos atribuídos a cada junta do robô são controlados através do erro posicional/orientacional do mesmo e de matrizes de ganho que influenciam quão abrupta é a correção [18].

2.2.1.2 Tipos de controlo de força-momento direto

O controlo de força-momento direto, em oposição, utiliza um *feedback loop*, de forma a garantir que a força de contacto é igual a um determinado valor desejado [18].

Apesar disso, limitar o controlo do movimento do robô à definição de um perfil de força não é uma solução flexível. Um exemplo desta situação é quando o robô inicia o seu movimento numa posição em que o mesmo não se encontra restringido, i.e. se o *end-effector* não se encontrar em contacto com qualquer superfície (robô em movimento livre).

Em [21] aborda-se a possibilidade de utilizar uma *Finite State-Machine*, que permite que o controlo de força não seja ativado enquanto o robô estiver em movimento livre, sendo neste caso apenas posicionalmente controlado.

Uma outra solução, ainda no âmbito do controlo de força direto, é o *Hybrid Motion-Force Control*, que combina o controlo de força com controlo posicional. Esta abordagem envolve controlar a força nas direções em que o movimento está restringido e vice-versa [22]. Existe um controlador para força e outro, separado, para o controlo da posição/orientação, sendo os *outputs* dos mesmos fornecidos ao robô.

Ora, as direções controladas por cada um dos controladores são definidas por matrizes de seleção, pelo que, caso as restrições sejam variáveis ao longo da trajetória, estas matrizes têm de ser continuamente atualizadas [23]. Desse modo, uma particularidade desta abordagem prende-se com o facto de ser necessário um modelo da superfície ou do ambiente com que o robô contacta, para existir informação acerca das restrições a que o robô está sujeito. Como qualquer modelo apresenta incerteza, uma eventual solução é recorrer a um algoritmo de estimação das direções restringidas a partir de *feedback* proveniente da força de contacto [19].

Uma forma de contornar problemas deste género é a utilização de *Parallel Force-Position Control*, que, apesar de ser representado por um diagrama de blocos semelhante ao *Hybrid Motion-Force Control*, como se vê na Figura 2.4, emprega controladores projetados de forma a que o controlo da força possua prioridade sobre o controlo de trajetória, não se recorrendo a matrizes de seleção. Desse modo, ligeiros erros de planeamento ou colisões indesejadas entre o *end-effector* do robô e o seu ambiente de trabalho podem ser acomodados [23]. Na Figura 2.4, P e q representam a posição do robô no *task space* e *joint soace*, respetivamente, F corresponde a força, F_d e P_d são a força e a posição pretendidas, F_s é a composição de F e F_d, sendo P_s a de P e P_d. Por fim, na Figura 2.4b, F_f corresponde à correção proveniente do ramo da força e F_p corresponde à proveniente do ramo posicional.

Uma questão a ter em atenção é que a maior parte dos robôs são controlados posicionalmente, no sentido em que funcionam como uma caixa negra, não estando a sua estrutura

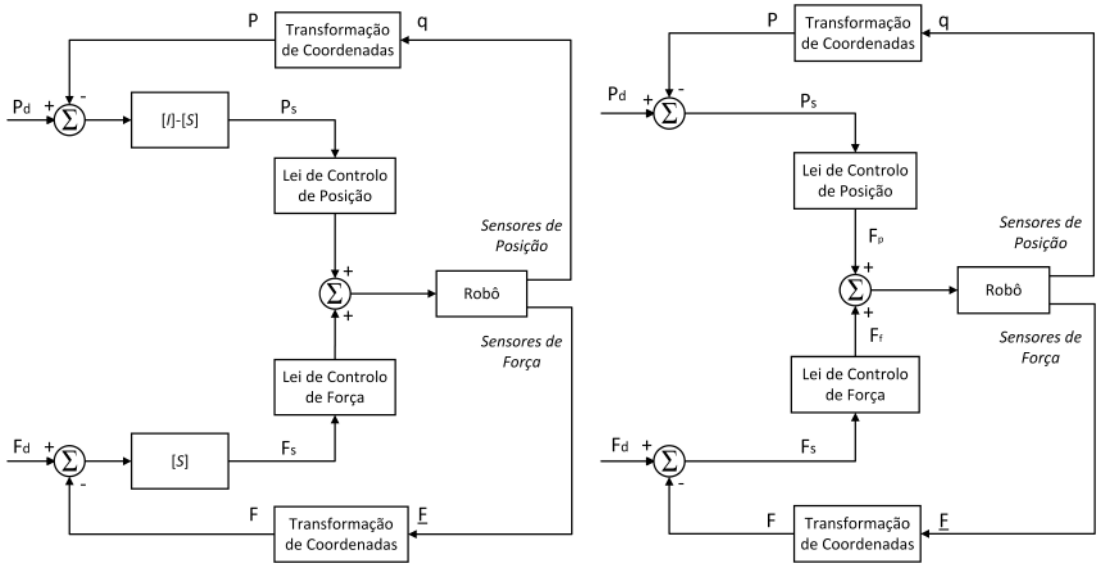


Figura 2.4: Diagrama de blocos: (a) *Hybrid Motion-Force Control* [24]; (b) *Parallel Force-Position Control* [23]

interna de controlo acessível e aceitando esta apenas inputs posicionais. Esta estrutura de controlo geralmente opera com valores no de ângulo/rotação de cada uma das juntas do robô, pelo que, para poder trabalhar com valores em coordenadas cartesianas, é necessário utilizar o modelo de cinemática inversa do robô [22].

Uma implicação deste pressuposto tem a ver com o facto de, na maior parte dos casos, não ser possível ter acesso aos *joint torques*. Desse modo, ao projetar controladores externos, deve ter-se em atenção se este é um fator limitante [22]. Contudo, ressalva-se que existe a possibilidade de o fabricante implementar uma estratégia de controlo de força baseada neste tipo de informações.

Assim, o *Position Based Force Control* pode ser favorecido. Este tipo de controlo consiste em utilizar uma lei de controlo de força que transforma desvios entre as forças desejada e medida numa correção de trajetória que, por sua vez, é fornecida ao robô [25]. Desta forma, é o próprio controlador interno do robô que se encarrega de acomodar estas correções, ao invés de tal ser feito num ramo separado (como no caso de *Parallel Force-Position Control* ou *Hybrid Motion-Force Control*), o que permite boa precisão e repetibilidade, já que esse controlador possui modelos de dinâmica e cinemática otimizados do robô, não sendo necessário recorrer a aproximações [21]. Na Figura 2.5 encontra-se um exemplo de uma ramo de *Position Based Force Control*, que pode ser inserido no ramo relativo à força de uma estrutura como a apresentada na Figura 2.4a.

2.2.1.3 Sensores de força

Tendo em conta o que foi referido previamente, é natural que sejam necessárias medições das forças e dos momentos gerados pelo contacto para fornecer às estruturas de

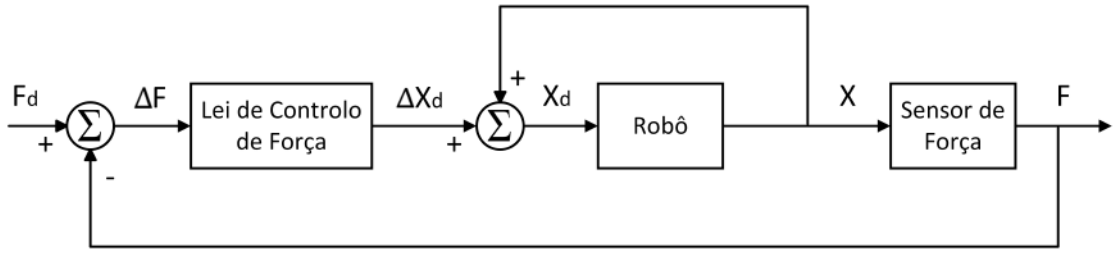


Figura 2.5: Exemplo de *Position Based Force Control* [25]

controlo. Tal é feito através de sensores, que são geralmente colocados entre o *end-effector* e a flange do robô [22].

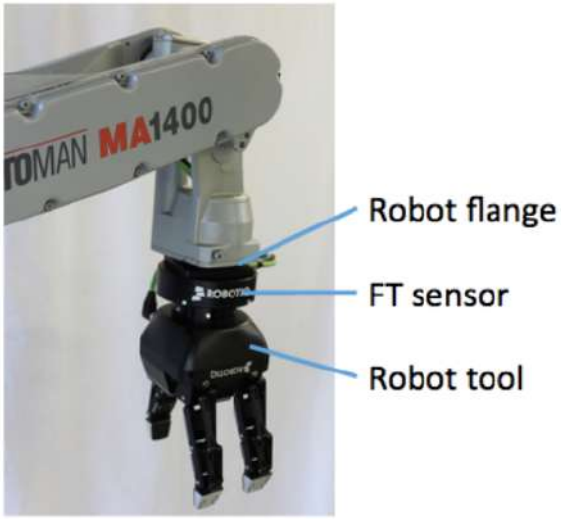


Figura 2.6: Incorporação de um sensor de força num braço robótico [26]

Existem sensores de vários tipos, como *Force Sensitive Resistors*, *Strain Gauges*, e Sensores Piezoelétricos. Os *Force Sensitive Resistors* apenas conseguem detetar força segundo uma direção. Já os *Strain Gauges* baseiam-se na utilização de um fio metálico que, ao ser deformado quando o sensor sofre um carregamento, altera a sua resistência, sendo possível converter essa variação numa leitura de força. Por fim, os Sensores Piezoelétricos baseiam-se, como o nome indica, no efeito piezoelétrico, em que o material, ao ser deformado, gera um sinal elétrico que pode ser convertido numa leitura de força. Enquanto que estes últimos não permitem ler cargas estáticas e estão mais direcionados para carregamentos de baixa intensidade, os *Strain Gauges* são mais adequados para forças mais elevadas [22]. Exemplos de cada um dos tipos de sensores referidos encontram-se presentes na Figura 2.7.

Atualmente, várias empresas já disponibilizam sensores de força e *software* para implementar controlo de força nos robôs [30], existindo inclusive alguns modelos em que estes sensores vêm incorporados nos mesmos [31], algo que demonstra como cada vez é mais evidente a utilidade deste tipo de estratégias em ambiente industrial.

(a) (b) (c)

Figura 2.7: Sensores de força: (a) *Force Sensitive Resistors* [27]; (b) *Strain Gauges* [28]; (c) Sensores Piezoelétricos [29]

2.2.1.4 Aplicações de controlo de força

Em [22], encontra-se um exemplo de aplicação *Hybrid Motion-Force Control* para permitir a um robô percorrer uma trajetória sobre um modelo de geometria desconhecida ao mesmo, apresentado na Figura 2.8. Os resultados obtidos são promissores, na medida em que conduzem a uma força de contacto estável, não ocorrendo *lift-off* da ferramenta. Contudo, existem algumas limitações do método utilizado, na medida em que apenas envolve alterações da trajetória segundo a direção vertical (não existem desalinhamentos segundo a direção perpendicular ao papel, na Figura 2.8). Desse modo, esta situação acaba por ser mais simples do que pode ocorrer em algumas aplicações industriais.

Figura 2.8: Modelo de geometria desconhecida [22]

De facto, é precisamente o facto de ser uma aplicação relativamente simples que permite a utilização de *Hybrid Motion-Force Control*, pois, como referido em 2.2.1.2, ao não existir um modelo bem definido da superfície, a aplicação deste tipo de controlo torna-se mais complexa.

Outra abordagem, apresentada em [32], procura solucionar este problema, utilizando *fuzzy logic* num esquema de *Hybrid Force-Motion Control* para controlar a trajetória do robô em situações em que a superície de contacto não é totalmente conhecida. Este tipo de lógica visa incorporar no controlador uma linguagem mais próxima da utilizada pelos humanos. O controlo é feito a partir de uma série de regras linguísticas, cuja influência no sistema é ponderada em função dos *outputs* do mesmo. Note-se que o facto de ser utilizada esta linguagem mais humana facilita a definição da estratégia de controlo [33].

Por outro lado, outra questão bastante relevante levantada em [22, 34] é a possibilidade de controlar a orientação do *end-effector*, algo que pode ser relevante. Para tal, é possível utilizar as componentes de força e momento medidas de forma a perceber a diferença entre a orientação do *end-effector* e a normal à superfície. Um exemplo simples, e apenas bidimensional, é ilustrado na Figura 2.9, na qual o movimento da ferramenta é feito da esquerda para a direita.

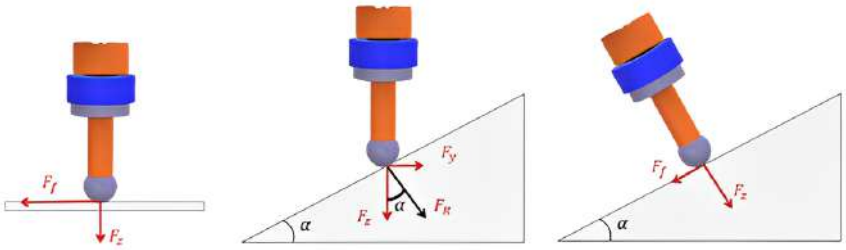


Figura 2.9: Controlo da orientação do *end-effector* [22]
F_f: Força de atrito F_z: Força exercida segundo o eixo da ferramenta
F_R: Força resultante F_y: Força exercida perpendicularmente ao eixo da ferramenta

Uma abordagem deste género é analisada em maior detalhe em [35]. Neste artigo, é apresentada uma metodologia para identificar em cada ponto um referencial em que dois dos vetores formam um plano tangente à superfície e o terceiro é normal à mesma. Por outro lado, a partir de trigonometria e de medição das componentes de força é possível definir a matriz de rotação necessária para que a força de contacto seja exclusivamente segundo a normal à superfície.

Assim, é possível estabelecer um sistema de controlo integrador (integrador de forma a procurar eliminar *steady state errors*), para ajustar a orientação [22].

Uma outra alternativa apresentada em [22, 34], mas não testada em ambiente real, é procurar controlar os momentos adequados de forma a que sejam iguais a zero.

Contudo, em nenhum dos artigos referidos é analisada a problemática de eventualmente existir conflito entre o controlo destinado à reorientação do *end-effector* e o designado para controlo da trajetória, por exemplo em peças com um perfil semelhante ao apresentado na Figura 2.10.

Figura 2.10: Perfil possivelmente propenso a conflitos entre controlo de trajetória e orientação

Esta questão poderá eventualmente, e para determinadas geometrias, ser solucionada recorrendo também a uma comparação com a *base frame* do robô.

3

Metodologia proposta

3.1 Delineamento da estratégia geral

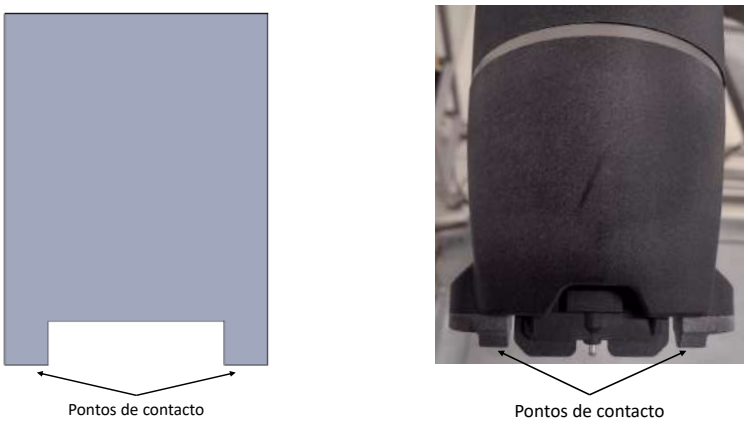
Os processos de inspeção por correntes induzidas de juntas brasadas em carroçarias de automóveis ocorrem em ambientes parcialmente estruturados, nos quais o ponto de início da inspeção e o caminho a ser percorrido por uma sonda de inspeção variam de carroçaria para carroçaria. Para realizar uma inspeção desse tipo, é necessário ser capaz de levar a sonda até ao início do cordão brasado, a partir de uma posição sobre o tejadilho que é variável de caso para caso. De seguida, é necessário estimar o caminho que deve ser percorrido pela sonda para que a mesma percorra o cordão, sem perdas de contacto.

Esse tipo de inspeção enfrenta duas fontes de dificuldade: a irregularidade superficial das juntas brasadas e a necessidade de manter um distanciamento adequado entre a sonda de correntes induzidas e o cordão (*lift-off*). Considerando esses dois fatores principais, têm sido propostos vários formatos de estrutura de sonda, destacando-se a estrutura com dois pontos de apoio equidistantes da sonda por apresentar os melhores resultados. Um desenho esquemático desse tipo de estrutura encontra-se na Figura 3.1a, encontrando-se a sonda no seu interior, na região central. A sonda posteriormente utilizada na validação da metodologia encontra-se na Figura 3.1b.

Para definir a superfície, tenha-se em consideração uma peça genérica com irregularidades e um ressalto que ajude a definir qual a posição alvo que a ferramenta deve atingir antes de ser iniciada a inspeção. O ressalto também ajuda a definir qual o caminho que a ferramenta terá de percorrer após a fase de ajustamento. Um exemplo deste tipo de superfície está presente na Figura 3.2. De notar que, na posição alvo referida, a sonda se encontra encostada ao ressalto e com um dos apoios no início da superfície, junto ao referencial apresentado na Figura 3.2. Tal é ilustrado na Figura 3.3.

Ora, a preocupação inicial foi o facto de a posição inicial da sonda poder ser em três regiões distintas: uma à esquerda do ressalto e outra à direita do mesmo (ver regiões A e B, respetivamente, na Figura 3.4), e outra região mais genérica, C, fora da superfície.

Assim, numa primeira abordagem, ponderou-se a elaboração de três programas distintos, além de um sistema de decisão, para o robô conseguir identificar a região em que



(a) (b)

Figura 3.1: Estrutura de sonda de correntes induzidas: (a) modelo genérico; (b) modelo real

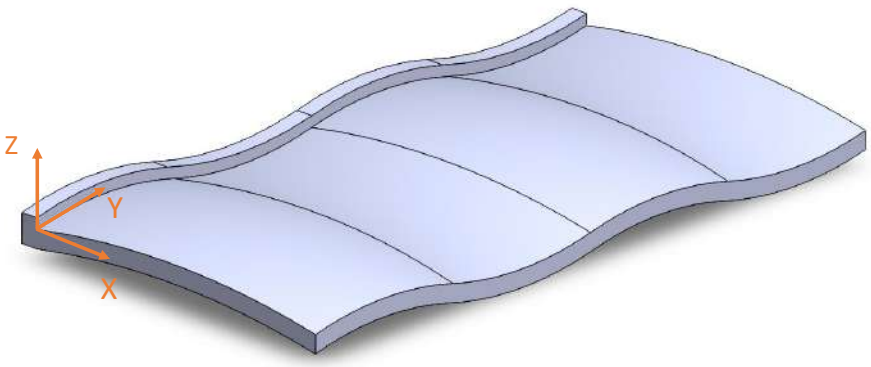


Figura 3.2: Superfície genérica

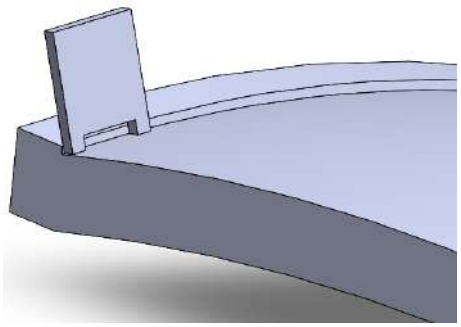


Figura 3.3: Posição alvo

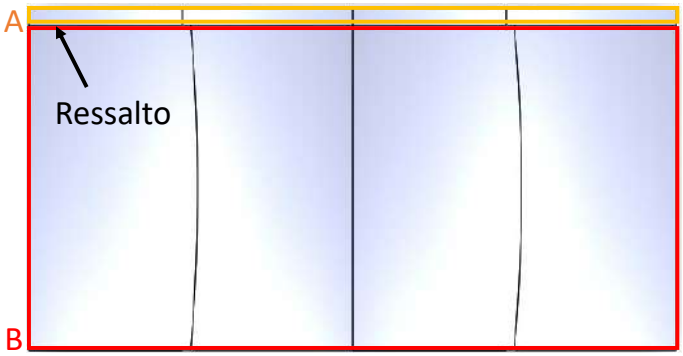


Figura 3.4: Vista de cima da superfície

se encontra e optar pelo programa correto.

Por outro lado, uma vez que a posição alvo da sonda é na região B (ferramenta encostada ao ressalto), propôs-se que a posição inicial da sonda fosse sempre na região B. Para tal ocorrer, a posição inicial do robô no programa deve ser definida com margem de erro suficiente. Além de mais simples, esta abordagem também aparenta ter vantagens a nível do tempo de execução da tarefa, uma vez que se saltam as etapas de avaliação da posição inicial (e decisão acerca da região em que a sonda se encontra) e de movimentação para a região B. Sendo esta aplicação direcionada para a indústria automóvel, em que os tempos de ciclo são tão apertados, considerou-se então que a metodologia proposta seria a mais adequada.

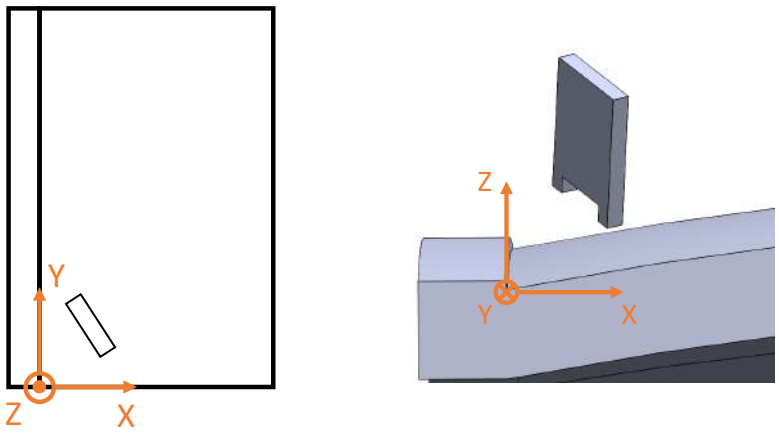
Ora, pelas razões apontadas no parágrafo anterior, a sonda acaba por movimentar-se sempre para a região B, independentemente da alternativa escolhida. Desse modo, para chegar a uma solução ao problema em questão, deve-se considerar então o que pode ser realizado a partir do momento em que a sonda se encontra numa posição genérica acima da superfície na região B, ou seja com desvios posicionais positivos em X, Y e Z.

No que concerne aos desvios de orientação, por motivos de simplicidade considerou-se que não teriam de ser realizadas correções segundo RY. Contudo, a solução deve ser capaz de corrigir desvios segundo RX e RZ, pelo que a posição genérica referida deve apresentar uma composição de rotações segundo os eixos X e Z.

No diagrama apresentado na Figura 3.5a é possível perceber os desvios referidos em X e Y, bem como a componente da rotação em torno de Z. Já na Figura 3.5b são visíveis o desvio posicional segundo Z e a rotação total em relação à posição alvo (ver Figura 3.3).

3.2 Primeiro ajustamento

Numa primeira etapa, aquilo que se pretende é levar a sonda até ao ressalto e conseguir um alinhamento da sonda com o mesmo. Contudo, em primeiro lugar, é relevante clarificar os referenciais que serão doravante mencionados. O referencial *Base Frame* é o que tem sido apresentado, nomeadamente na Figura 3.2 e na Figura 3.5. Já o referencial *Tool Frame* é apresentado na Figura 3.6.



(a) (b)

Figura 3.5: Desvios da posição alvo: (a) segundo X, Y e RZ; (b) segundo Z, RX e RZ

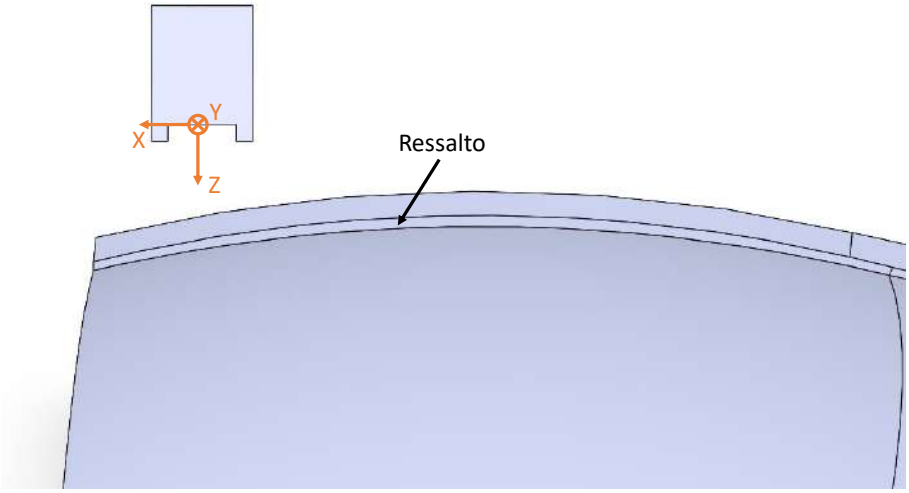


Figura 3.6: *Tool Frame*

Algo extremamente relevante é que, uma vez que a superfície possui uma geometria irregular e a posição inicial da sonda é desconhecida, o mero controlo posicional não parece ser suficiente para cumprir os objetivos propostos.

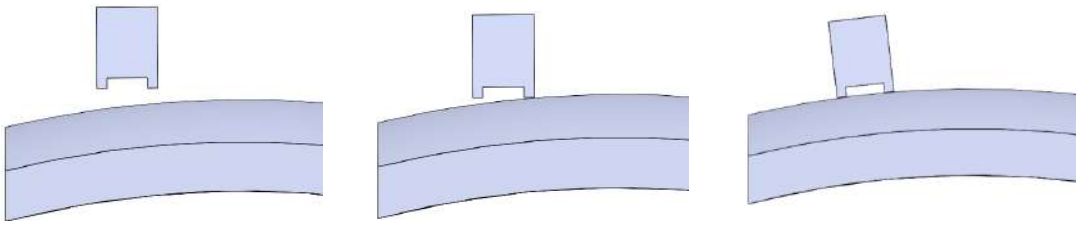
Assim, a abordagem que será apresentada de seguida sugere a movimentação da sonda através de um robô, com utilização de controlo de força aliado ao controlo posicional, de forma a existir uma maior capacidade de adaptação ao meio desconhecido com que a sonda interage.

3.2.1 Fase 0 - Aproximação ao tejadilho

Em primeiro lugar, a sonda realiza um movimento vertical descendente, relativamente à *Base Frame*, utilizando-se controlo de força para detetar o instante em que existe um contacto com a superfície, como ilustrado nas Figuras 3.10a e 3.10b.

3.2.2 Fase 1 - Estabelecimento de contacto com os dois apoios

Já a Fase 1 consiste na reorientação da sonda segundo RY (*Tool Frame*). Para tal, aplica-se uma força segundo o sentido positivo do eixo Z e define-se *compliance* segundo RY (*Tool Frame*). A sequência definida pelas Fases 0 e 1 está ilustrada na Figura 3.7.



(a) (b) (c)

Figura 3.7: Sequência de movimentos: (a) Início da Fase 0; (b) Fim da Fase 0/Início da Fase 1; (c) Fim da Fase 1

De forma otimizar esta correção, sugere-se que o movimento associado ao controlo de força seja também realizado segundo o sentido positivo do eixo Z da sonda, numa tentativa de fazer com que a sonda tenda a alinhar-se com o tejadilho, em vez de realizar uma rotação no sentido oposto ao desejado.

Para determinar se a Fase 1 já pode terminar, ou seja, se já houve uma reorientação da sonda de tal forma que ambos os apoios se encontrem em contacto com a superfície, propõe-se a definição de um intervalo de tempo destinado à correção, que poderá ser determinado empiricamente.

3.2.3 Fase 2 - Movimento de aproximação ao cordão

Esta fase consiste em levar a sonda até ao cordão. Para tal, realiza-se um movimento para um ponto genérico no lado oposto do ressalto na superfície. O objetivo é levar a sonda até esse ressalto e, quando ocorre o contacto, existe um aumento na componente da força segundo o sentido negativo do eixo Y da ferramenta. Esse aumento pode ser detetado e utilizado para parar este movimento, passando-se para a etapa seguinte.

Durante o movimento, controla-se a força de forma a que seja aplicada uma carga segundo o sentido positivo do eixo Z da sonda (*Tool Frame*). Esta força destina-se a garantir que não é perdido o contacto entre a sonda e o tejadilho, o que poderia fazer com a sonda passasse por cima do ressalto, não sendo o contacto detetado, significando uma falha na operação.

Por outro lado, durante este movimento, mantém-se *compliance* segundo RY (*Tool Frame*), pois caso contrário, devido à geometria irregular da superfície, mesmo que inicialmente a sonda esteja perfeitamente alinhada, tal pode sofrer alterações. Uma vez que este movimento pode envolver alguma trepidação, por ser basicamente um movimento de arrasto da sonda ao longo do tejadilho, considerou-se a possibilidade de a velocidade de correção segundo RY dever ser reduzida, para reduzir instabilidades.

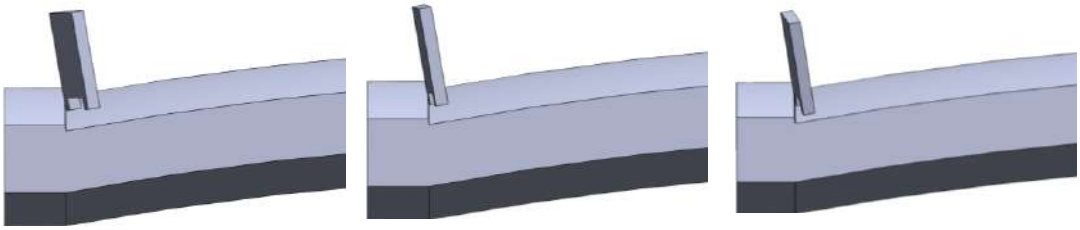
3.2.4 Fase 3 - Ajustamento ao cordão

Chegando a sonda à zona do cordão, é então necessário proceder ao ajustamento inicial da sonda ao cordão, algo que é realizado com base numa sequência de ajustamentos segundo RY e RZ (*Tool Frame*).

Os ajustamentos segundo RZ destinam-se a alinhar a sonda com a direção do cordão, sendo proposto que tal seja feito à base de um movimento no sentido negativo de X (*Base Frame*) em conjunto com força segundo o sentido positivo de Z (*Tool Frame*), para garantir a manutenção do contacto com o cordão, e *compliance* em RZ (*Tool Frame*), para permitir a rotação.

Já os ajustamentos segundo RY procuram que, no final desta etapa, ambos os apoios da sonda se encontram em contacto com o cordão, algo que poderá ser realizado a partir de movimento no sentido negativo de Z (*Base Frame*), em conjunto com força segundo o sentido positivo de Z e *compliance* em RY (*Tool Frame*), à semelhança do que foi descrito na Fase 1 (3.2.2).

Na Figura 5.11, apresenta-se uma sequência exemplificativa.



(a) (b) (c)

Figura 3.8: Ajustamento ao cordão: (a) Primeiro contacto com o ressalto; (b) Etapa de reorientação segundo RZ (*Tool Frame*); (c) Etapa de reorientação segundo RY (*Tool Frame*)

Para que as correções sejam feitas aos poucos e a sonda lentamente se aproxime da posição desejada, é necessário definir uma condição que deve ser cumprida para que se alterne entre as etapas de correção segundo RY e RZ (*Tool Frame*). À semelhança do que foi sugerido para a Fase 1 (ver 3.2.2), propõe-se novamente que seja definido um intervalo de tempo para cada correção, que poderá ser determinado empiricamente.

Esta sequência de movimentos de correção dos desvios segundo RY e RZ (*Tool Frame*) deve ser repetida até se considerar que o ajustamento já é adequado o suficiente para se poder passar à fase seguinte. Uma forma de determinar isso poderá ser por exemplo medir o deslocamento efetuado segundo RZ em cada iteração. Quando esse deslocamento for bastante reduzido, significa que a sonda praticamente não se moveu durante o último movimento corretivo, ou seja, que a sua posição está a convergir para a pretendida.

No final desta fase, o objetivo é que a sonda se encontre numa posição genérica, alinhada com o ressalto e em contacto com o mesmo, algo ilustrado na Figura 3.9.

A sequência de ajustamento ao ressalto descrita neste subcapítulo 3.2 encontra-se resumida no Algoritmo 1. Este algoritmo foi posteriormente implementado num robô

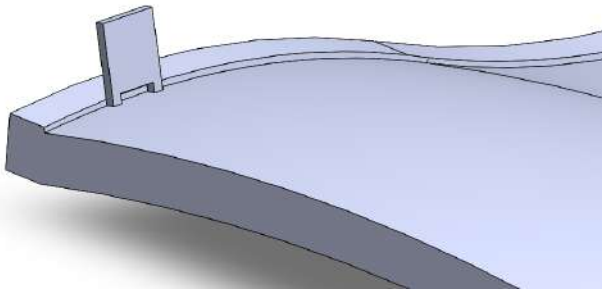


Figura 3.9: Posição genérica após o primeiro ajustamento ao ressalto

colaborativo (Anexo III).

Algoritmo 1 Alinhar a sonda com o ressalto

```
1: Movimento para um ponto na região B
2: Movimento no sentido negativo de Z (Base Frame) até contactar com o tejadilho
3: Iniciar Timer
4: Iniciar controlo de força (Tool Frame, Fz: sentido positivo, TRy: compliance)
5: Aguardar pela reorientação da sonda
6: Parar o controlo de força
7: Iniciar controlo de força (Tool Frame, Fz: sentido positivo, TRy: compliance)
8: while Colisão com o ressalto não detetada do
9:     Movimento no sentido negativo de X (Base Frame)
10: end while
11: Parar o controlo de força
12: while Rotação RZ (Tool Frame) > valor do
13:     Iniciar controlo de força (Tool Frame, Fz: sentido positivo, TRz: compliance)
14:     while Limite de tempo para a correção não for atingido do
15:         Movimento no sentido negativo de X (Base Frame)
16:     end while
17:     Parar o controlo de força
18:     Iniciar controlo de força (Tool Frame, Fz: sentido positivo, TRy: compliance)
19:     while Limite de tempo para a correção não for atingido do
20:         Movimento no sentido positivo de Z (Base Frame)
21:     end while
22:     Parar o controlo de força
23: end while
```

3.3 Encontro da posição inicial ideal

As fases descritas previamente constituem uma metodologia que poderá ser capaz de encostar a sonda ao ressalto e de alinhá-la com o mesmo. Contudo, não chegam para levar a sonda para a posição alvo (ver Figura 3.3). Desse modo, de seguida será proposta uma abordagem para resolver esse problema.

3.3.1 Fase 4 - Pausa e aquisição de dados

Em primeiro lugar, poderá ser conveniente realizar uma pausa no movimento da sonda, para garantir que a mesma se encontra estável, após os movimentos de correção realizados na fase anterior. Esta pausa deve ser o mais reduzida possível, de forma a que todo o processo de ajustamento seja célere.

Durante esta pausa, deve ser guardada a posição da sonda no início deste processo de correção. Esta posição será denominada de ponto inicial durante o resto deste subcapítulo.

3.3.2 Fases 5, 6, 7 e 8 - Encontro do início da superfície

Após a fase anterior, o que se propõe é movimentar a sonda no sentido positivo de X (*Tool Frame*), até a mesma sair totalmente da superfície. A partir disso, é possível calcular a distância entre o ponto inicial e a posição alvo, e, consequentemente, movimentar a sonda de forma a que a mesma se passe a situar na posição pretendida (alvo).

Para tal, no início do movimento no sentido positivo de X (*Tool Frame*), inicia-se um temporizador, que apenas será parado quando a sonda sair totalmente da superfície. Este movimento engloba as Fases 5 e 6, sendo que a Fase 5 termina quando existe perda total do contacto do apoio anterior e a Fase 6 termina quando existe perda total de contacto do apoio posterior. A razão pela qual se distinguem estas duas fases é por ser possível que, devido às características diferentes de apoio em ambas, o perfil de forças e momentos sentido pela sonda poder ser distinto. Na Figura 3.10 encontra-se ilustrado o movimento referido.

(a) (b) (c)

Figura 3.10: Sequência de movimentos: (a) Início da Fase 5; (b) Fase 6 em andamento; (c) Fim da Fase 6

Um possível modo de detetar a perda total de contacto é através da constante monitorização da velocidade do Tool Center Point (TCP). Caso seja aplicado um controlo de força segundo o sentido positivo de Z (*Tool Frame*) durante o movimento da sonda ao longo da superfície, então a tendência da sonda quando perde contacto com a superfície é de repentinamente aumentar a sua velocidade neste sentido.

Quando efetivamente é detetada essa perda total, inicia-se a Fase 7, em que a sonda deve parar o seu movimento e deve ser guardado o valor do temporizador numa variável.

Nesta altura, sabendo a velocidade da sonda (v) durante o movimento segundo o sentido positivo de X (*Tool Frame*) e sabendo o tempo que a mesma demorou a perder o contacto com a superfície (t), pode ser utilizada a fórmula da Equação 3.1 para calcular a

distância entre o ponto inicial e a posição alvo (d), e, então definir o movimento necessário para que a sonda se movimente para esta última (Fase 8). De notar apenas que, para tal, é necessário ter em conta o comprimento da base da sonda (L), que deve ser descontado, uma vez que o Timer apenas é parado quando a totalidade da sonda saiu do cordão.

$$d = vt - L \tag{3.1}$$

Assim, para finalizar, sugere-se que a sonda se movimente novamente para o ponto inicial, e depois realize o movimento calculado para chegar à posição alvo, estando este movimento, correspondente à Fase 8, representado na Figura 3.11, em que também é visível a Fase 9, que será abordada de seguida.

No Algoritmo 2, encontra-se resumida a parte do programa responsável pela execução das tarefas referidas nas Fases 5 a 8. Este algoritmo foi posteriormente implementado num robô colaborativo (Anexo III).

Algoritmo 2 Encontrar a posição inicial e ajustamento final

1: Guardar numa variável a posição atual (ponto inicial)
2: Realizar uma pausa
3: Iniciar Timer
4: Iniciar controlo de força (Tool Frame, Sentido positivo de Z)
5: Movimento no sentido positivo de X (*Tool Frame*) até ser verificado um aumento repentino na velocidade do TCP segundo Z (*Tool Frame*)
6: Parar Timer e guardar numa variável o seu valor
7: Reset Timer
8: Calcular a partir do valor do Timer e da velocidade segundo X (*Tool Frame*) a distância percorrida
9: Calcular a distância segundo X (*Tool Frame*) entre o ponto inicial e a posição alvo
10: Movimentar a sonda para o ponto inicial
11: Movimentar a sonda para a posição alvo

3.3.3 Fase 9 - Ajustamento final

Após a sonda chegar à posição alvo, poderá eventualmente ser necessário um ajuste final segundo RY (*Tool Frame*), sobretudo quando o ponto inicial e a posição alvo estão algo distantes. Para tal, à semelhança do que acontece na Fase 1, propõe-se a utilização de força no sentido positivo de Z, em conjunto com *compliance* segundo RY (*Tool Frame*).

Isto deve-se ao facto de o movimento referido na Fase 8, tal como nas Fases 5 e 6, ser um movimento segundo uma direção fixa (X do *Tool Frame*) e a superfície apresentar alguma ondulação. Apesar de, em teoria, o controlo de força aplicado significar que não será perdido o contacto entre a sonda e a superfície durante este movimento, podem ser criados desalinhamentos segundo RY, que no final devem ser corrigidos. A Figura 3.11 ilustra o que pode acontecer com esta abordagem.

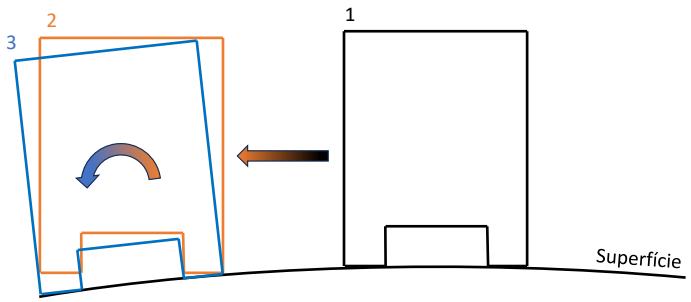
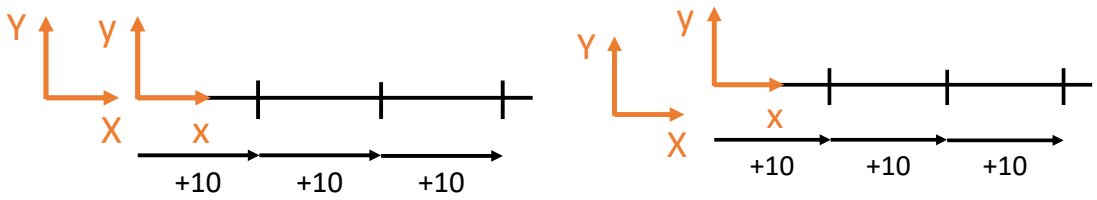


Figura 3.11: Fases 8 e 9

3.4 Definição do caminho a percorrer

Após o ajustamento da sonda ao cordão, segue-se a segunda tarefa, de percorrer um caminho ao longo da superfície, mantendo a sonda encostada ao ressalto.

De uma forma geral, pensou-se numa abordagem que envolvesse mapear um cordão, de forma a gerar um caminho de referência. Assim, após a fase de ajustamento, comparar-se-ia a posição da sonda com a posição inicial da sonda no caminho de referência, e consoante a diferença verificada, obter-se-ia o caminho de inspeção pela transformação do caminho de referência, como representado na Figura 3.12.



(a) (b)

Figura 3.12: Fase de inspeção genérica: (a) Percurso de referência; (b) Percurso de inspeção calculado

Uma metodologia proposta para cumprir esse objetivo encontra-se resumida no Algoritmo 3.

Contudo, essa metodologia pressupõe o cumprimento de uma etapa preliminar: deve definir-se a posição e orientação de um conjunto de pontos que constituem o percurso de referência da sonda (percurso ideal). De seguida, deve ser calculada a diferença entre cada um dos vetores posição/orientação dos pontos que compõem o percurso de referência e o do ponto inicial ideal (ou seja, o primeiro do conjunto dos pontos do percurso de referência). Este processo apenas necessita de ser realizado uma vez, e este percurso ideal servirá de referência para todas as inspeções subsequentes.

Cumprida esta etapa preliminar, a solução proposta envolve, em primeiro lugar, guardar a posição e orientação do ponto inicial da inspeção, que é a posição atingida no final de todas as etapas prévias de ajustamento. Este ponto, teoricamente, corresponde à posição alvo da fase de ajustamento. Diz-se teoricamente porque é expectável que durante

o processo de ajustamento referido surjam pequenos erros, que acabem por fazer com que a posição atingida não seja totalmente igual à posição alvo.

De seguida, devem ser guardadas (em variáveis) as diferenças calculadas entre as posições dos pontos ideais do caminho e a do ponto inicial ideal. Estas diferenças também correspondem a vetores posição/orientação.

Por fim, estes vetores contendo as diferenças entre os pontos do percurso ideal e o ponto inicial ideal devem ser somados à posição inicial de inspeção.

Assim, é possível criar uma espécie de um mapeamento relativo do cordão. Como referido no Algoritmo 3, a partir da posição e da orientação do ponto inicial de inspeção (alcançado através da metodologia descrita em 3.2), é possível estimar então o trajeto necessário para percorrer o cordão. Este algoritmo foi posteriormente implementado num robô colaborativo (Anexo III).

Contudo, uma vez que o ponto inicial e o percurso ideais foram definidos manualmente, é natural que o mapeamento relativo não seja perfeito. A esta dificuldade, acresce o facto de também não ser possível garantir que o ponto inicial de inspeção (ponto final da sequência de aproximação ao cordão) está perfeitamente alinhado com o cordão, sobretudo em termos de orientação. Tal poderá ser especialmente gravoso para caminhos de inspeção mais longos, em que erros mínimos de orientação inicial se podem traduzir em desvios significativos nas zonas finais do caminho. Este fenómeno é ilustrado pela Figura 3.13.

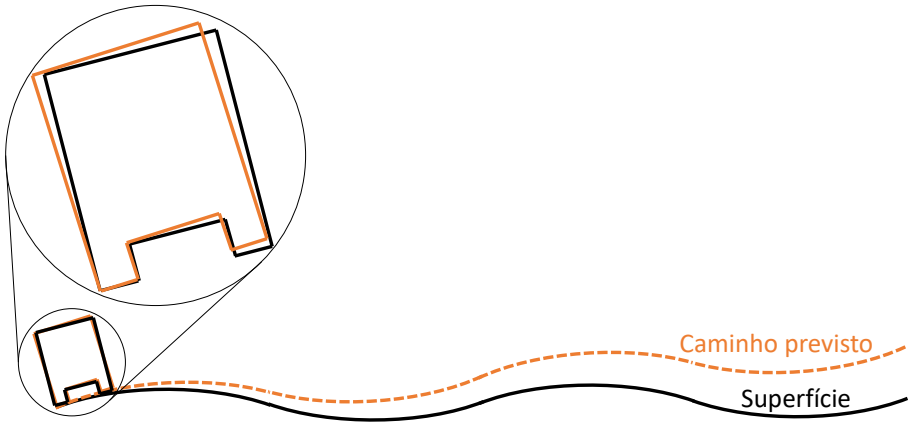


Figura 3.13: Erro na previsão da trajetória devido a desvio na posição inicial

Além disso, por ser pretendido que o programa desenvolvido seja capaz de se adaptar a superfícies teoricamente idênticas, mas na realidade com diferenças (por exemplo devido às incertezas habitualmente inerentes aos processos produtivos), é natural que o caminho a percorrer nas diversas superfícies não seja totalmente igual.

Estas questões levam a que, além do mero mapeamento do caminho previsto, se sugira uma abordagem alternativa, que também integre controlo de força.

No que concerne ao controlo de força, existem duas direções que parecem ser especialmente úteis: Y e Z (*Tool Frame*). O controlo de força segundo o sentido positivo de Y (*Tool Frame*) poderá ajudar a manter a sonda encostada ao ressalto, mesmo que o mapeamento relativo defina um caminho que faça com que a sonda tenha tendência a afastar-se do

ressalto. Por sua vez, o controlo de força segundo o sentido positivo de Z poderá ajudar a que a sonda se mantenha constantemente em contacto com a superfície.

Um último pormenor a referir é que esta abordagem, de utilizar um mapeamento relativo, adiciona importância à etapa detalhada no subcapítulo 3.3, de atingir consistentemente sempre a mesma posição alvo.

Por exemplo, devido à ondulação da superfície, se o ponto inicial para percorrer o caminho for um pouco mais longe do início da superfície do que o previsto no mapeamento ideal, o programa assumiria que os picos se encontram mais afastados do que na realidade se encontram. Desse modo, a sonda possivelmente iniciaria uma trajetória descendente mais tarde do que o pretendido, ou seja, o caminho não seria percorrido como previsto.

Algoritmo 3 Percorrer o caminho

1: Guardar a posição atual, correspondente à posição inicial da inspeção
2: Guardar as diferenças entre as posições de cada um dos pontos que define o percurso ideal e a posição inicial ideal
3: Calcular os pontos do trajeto, somando as diferenças calculadas ao ponto inicial de inspeção
4: Iniciar controlo de força
5: Movimento da sonda ao longo dos pontos definidos
6: Parar o controlo de força
7: Afastamento da sonda da superfície

4

Plataforma de aquisição de dados

Para testar a metodologia proposta, constituiu-se uma célula robótica de inspeção. Utilizou-se um robô UR10e, da Universal Robots, um analisador de falha Olympus Nortec 600, um conversor de sinal analógico-digital Digilent Analog Discovery 2, e, para poder conectar o analisador de falha com o conversor foi utilizado um Digilent BNC Adapter. Além disso, o computador utilizado foi um ASUS VivoBook X513UA, com o sistema operativo Windows 11. As relações entre estes dispositivos serão posteriormente esclarecidas.

O objetivo, como será elaborado no capítulo 5, é o de proceder ao ajustamento ao cordão brasado presente no tejadilho de um automóvel, e de percorrer um caminho ao longo da junta brasada do mesmo, realizando uma inspeção por correntes induzidas.

Em primeiro lugar, o desenvolvimento de uma *interface* que permitisse a comunicação entre o robô e um computador externo (através de um cabo *ethernet*) tornou-se essencial para a coordenação do processo de inspeção. A comunicação referida foi feita à base da utilização do protocolo Real-Time Data Exchange (RTDE), em conjunção com uma conexão ao servidor *Dashboard* (disponível nos robôs da *e-series* da Universal Robots).

Um dos objetivos referidos é a operação remota do robô, por exemplo a possibilidade de fornecer energia aos motores ou de desbloquear os travões, até carregar programas e colocá-los em execução ou em pausa. Para tal, é possível estabelecer uma comunicação através do servidor *Dashboard* [36, 37], encontrando-se disponível em [37] um ficheiro com a lista de comandos que é possível transmitir ao robô por esta via.

Por outro lado, é extremamente útil poder ter acesso aos *outputs* do robô, como o estado dos seus I/O's (a partir dos quais é possível por exemplo ter acesso a valores de variáveis definidas no programa .urp do robô) ou dados relativos à posição, velocidade, força, corrente nas *joints*, entre outros, para efeitos de monitorização do processo. Além disso, também existe valor na possibilidade de fornecer *inputs* ao robô, por exemplo para controlar o progresso da execução do programa. Para os fins referidos, é possível recorrer ao protocolo RTDE, encontrando-se a lista completa do tipo de comunicações que podem ser estabelecidas em [38].

Na Figura 4.1 é esquematizada a conexão realizada por cabo *ethernet* entre o computador e o robô, através da qual é possível estabelecer as comunicações previamente referidas

(protocolo RTDE e servidor *Dashboard*).

Além disso, a célula robótica foi planeada para ser possível recolher num computador externo dados provenientes da sonda de correntes induzidas, para posterior tratamento e/ou análise. Como ilustrado na Figura 4.1, a sonda encontra-se conectada a um analisador de falha (Olympus Nortec 600C).

De forma a receber os dados pretendidos num computador (Asus VivoBook X513UA), é necessário converter o sinal analógico fornecido pelo analisador de falha num sinal digital, caso contrário a frequência de aquisição não é adequada. Para tal, optou-se pelo Digilent Analog Discovery 2 como conversor de sinal, em conjunto com o adaptador Digilent BNC Adapter, também presentes na Figura 4.1.

Além disso, é necessário estabelecer comunicação entre o Digilent Analog Discovery 2 e o computador, podendo utilizar-se uma biblioteca presente no WaveForms Software Development Kit (SDK) para esse fim [39].

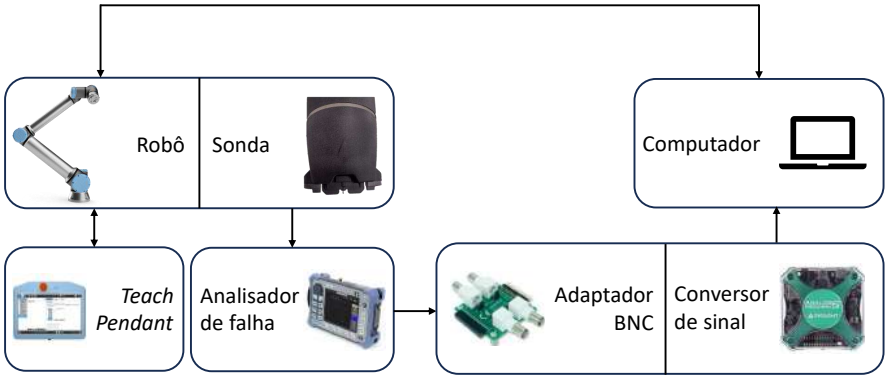


Figura 4.1: Esquema da comunicação entre os elementos da montagem experimental [40–44]

4.1 Lógica geral de programação

A estrutura do programa desenvolvido apresenta duas partes principais: por um lado, tem-se a secção do código destinada à criação da interface e, por outro, a secção do código em que são definidas as funções que se destinam a estabelecer comunicação com o robô, e que podem ser chamadas de forma intuitiva através da interface.

O código relativo à criação da interface foi programado tendo como base a biblioteca tkinter disponível em python, encontrando-se o resultado presente na Figura 4.2.

Ora, é fácil perceber que o código desenvolvido deve ser capaz de realizar várias tarefas simultaneamente. Por exemplo, deve ser possível executar o comando *Stop* enquanto o programa de inspeção está em execução, o código deve ser capaz de atualizar a janela de *feedback* enquanto outra instrução está em funcionamento, entre outros. Como tal, não é possível que, ao carregar em determinado botão, a função correspondente seja chamada de forma convencional. De facto, tal resulta em que a janela da interface fique irresponsiva durante a duração da execução do comnado, o que impossibilita a execução de uma

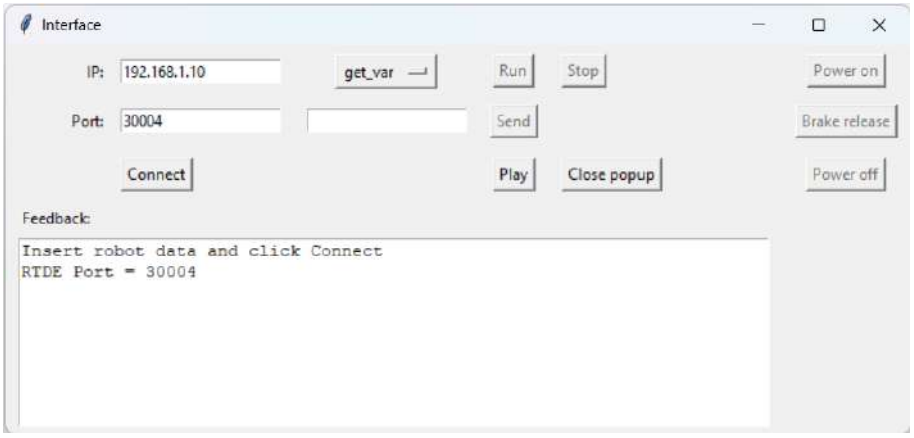


Figura 4.2: Interface para comunicação entre um computador e o robô

nova instrução e a atualização da janela de *feedback* em tempo real. Tal é especialmente problemático no caso da execução do programa de inspeção (get_var), por ter uma duração nunca inferior a cerca de trinta segundos, e por ser necessário ter acesso ao botão de *Stop* e ao *feedback* durante o decorrer do mesmo.

O modo como este problema foi contornado foi através da utilização do módulo threading, que permite que o corpo principal do programa, correspondente ao *loop* em que funciona a interface, não seja interrompido durante a execução de outros comandos. Assim, ao carregar num determinado botão, é chamada uma função que, por sua vez, abre um *thread* que contém a função que realmente define o comportamento desejado quando o botão é premido. Tal é ilustrado pelo Algoritmo 4, em que o botão é criado de forma a que, quando premido, chame a função pwr_on_button_clicked, que por sua vez cria uma *thread* em que é chamada a função pwr_on.

Algoritmo 4 Botão *Power on*

1: Criar o botão *Power on* na interface e definir que o mesmo chama a função pwr_on_button_clicked
2: **function** PWR_ON_BUTTON_CLICKED()
3: Iniciar uma *thread* com a função pwr_on
4: **end function**
5: **function** PWR_ON()
6: Aceder à variável global que contém o IP do robô
7: Estabelecer a conexão com o servidor Dashboard
8: Envio do comando Dashboard para fornecer potência aos motores do robô
9: $robot\_mode \leftarrow$ estado atual do robô
10: **while** $robot\_mode \neq$ 'IDLE' **do**
11: $robot\_mode \leftarrow$ estado atual do robô
12: **end while**
13: Atualizar a *Feedback box* com o estado atual do robô
14: **end function**

De notar que no caso de funções cuja execução seja quase instantânea, este aspeto não é problemático, podendo a função ser chamada diretamente pelo premir do botão. Tal pode

ser observado no Algoritmo 5, em que é definido o botão *Stop* de forma a que, quando premido, chame a função stop_button_clicked, que por sua vez define o comportamento desejado após tal acontecer, sem necessidade de recorrer a *threads*.

Algoritmo 5 Botão *Stop*

```
1: Criar o botão Stop na interface e definir que o mesmo chama a função stop_button_clicked
2: function STOP_BUTTON_CLICKED()
3:     Aceder à variável global que contém o IP do robô
4:     Estabelecer a conexão com o servidor Dashboard
5:     program_state ← Estado de execução do programa
6:     if program_state = 'STOPPED' then
7:         Informar o utilizador de que o programa já se encontra parado
8:     else
9:         Enviar o comando Dashboard para parar o programa
10:    end if
11: end function
```

4.2 Programa de inspeção

4.2.1 *Outputs*

No que concerne à função get_var(), que contém as instruções relativas ao programa de ajustamento e inspeção do cordão brasado, a mesma possui dois *outputs* distintos. Por um lado, através do módulo openpyxl, é criada um documento .xlsx com quatro folhas de cálculo distintas. Na primeira, apresentam-se os dados relativos à posição do robô (*Base Frame*) durante a fase de aproximação do robô, em função do tempo decorrido. A segunda folha de cálculo apenas difere no facto de apresentar os dados provenientes do sensor de força-momento segundo a *Tool Frame*. Já a terceira e a quarta folhas referem-se à fase de inspeção em si, contendo dados relativos à posição (*Base Frame*) e força (*Tool Frame*), respetivamente. Estes dados são apresentados em função não só do tempo, como também da distância percorrida. Além disso, a folha relativa à posição também encerra em si os dados recolhidos pelo Digilent, de forma a ser possível associar anomalias no sinal a uma determinada posição, algo fulcral para ser possível determinar a localização dos defeitos. Esta folha pode ser observada na Figura 4.3.

Por outro lado, esta função também pode apresentar um gráfico do sinal obtido pelo Digilent em função da distância percorrida ao longo da fase de inspeção, recorrendo à biblioteca matplotlib.pyplot.

4.2.2 *Inputs*

Em termos de *inputs*, é necessário que a função seja capaz de receber informação acerca do IP do robô e da *communication port* a utilizar.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Tempo [s]	Dist [m]	X [m]	Y [m]	Z [m]	RX [rad]	RY [rad]	RZ [rad]	CH1 [V]	CH2 [V]	BASE FRAME	
2	0	0	0.652063	-0.69802	0.15178	-2.1936	2.201284	0.064387	-0.01111	0.022631		
3	0.002	4.6E-05	0.652029	-0.69805	0.151783	-2.19341	2.201451	0.064429	-0.01279	0.03743		
4	0.004	2.98E-05	0.652062	-0.69804	0.151807	-2.19357	2.201374	0.064374	-0.01178	0.052229		
5	0.006	6.17E-05	0.652041	-0.69804	0.151837	-2.19347	2.20138	0.06429	-0.01313	0.03373		
6	0.008	4.07E-05	0.652079	-0.69799	0.151792	-2.19356	2.201388	0.064348	-0.01548	0.007832		
7	0.01	6.4E-05	0.652058	-0.69799	0.151833	-2.19358	2.201346	0.064304	-0.01816	0.026331		
8	0.012	5.68E-05	0.65206	-0.69799	0.151827	-2.19354	2.201381	0.064256	-0.01615	0.04853		
9	0.014	0.000151	0.652096	-0.6979	0.151857	-2.19357	2.201378	0.064226	-0.01145	0.070728		
10	0.016	0.000153	0.652075	-0.69791	0.151882	-2.19351	2.20138	0.064151	-0.0091	0.059629		
11	0.018	0.000201	0.652146	-0.69787	0.151885	-2.19359	2.201312	0.063986	-0.01447	0.04113		
12	0.02	0.000201	0.652091	-0.69783	0.151839	-2.19363	2.201396	0.064052	-0.01514	0.03743		
13	0.022	0.000232	0.652091	-0.6978	0.151819	-2.1936	2.201446	0.064111	-0.02084	0.04483		
14	0.024	0.00031	0.652064	-0.69772	0.15182	-2.19367	2.201379	0.06419	-0.02453	0.04853		
15	0.026	0.000326	0.652064	-0.6977	0.151805	-2.19358	2.201445	0.064069	-0.02319	0.04853		
16	0.028	0.000383	0.652036	-0.69764	0.151764	-2.19364	2.201443	0.064186	-0.02051	0.04853		
17	0.03	0.00044	0.652124	-0.6976	0.151872	-2.19371	2.201337	0.063879	-0.01548	0.04113		
18	0.032	0.000528	0.652098	-0.6975	0.151854	-2.19359	2.201372	0.06389	-0.01212	-0.00327		
19	0.034	0.000588	0.652146	-0.69745	0.15189	-2.19369	2.201408	0.063707	-0.00944	-0.03287		
20	0.036	0.000674	0.652106	-0.69735	0.151848	-2.19363	2.201433	0.063864	-0.00507	-0.05506		
21	0.038	0.000703	0.652076	-0.69732	0.151859	-2.19366	2.201499	0.063701	-0.00709	-0.01807		
22	0.04	0.000773	0.65206	-0.69725	0.151853	-2.19358	2.201513	0.063673	-0.00574	0.018931		
23	0.042	0.000866	0.652116	-0.69717	0.151885	-2.19359	2.201465	0.063432	-0.00709	-0.00327		
24	0.044	0.001012	0.652135	-0.69702	0.151858	-2.19365	2.201479	0.063537	-0.00373	-0.02177		
25	0.046	0.001113	0.652139	-0.69692	0.151871	-2.19373	2.201428	0.063531	-0.00574	-0.02177		
26	0.048	0.001179	0.652081	-0.69685	0.151894	-2.19355	2.201574	0.06344	-0.0091	-0.02547		
27	0.05	0.001331	0.652179	-0.6967	0.151911	-2.19371	2.201527	0.063314	-0.0091	0.018931		

Aprox_TCP_pose | Aprox_TCP_force | Inspec_TCP_pose | Inspec_TCP_force

Figura 4.3: Dados de posição do robô e do sinal proveniente da sonda

Além destes *inputs*, existem algumas outras opções de customização, mas que já requerem acesso ao próprio código, não sendo possível alterá-las diretamente a partir da interface. Entre as mesmas contam-se a frequência de comunicação entre robô e computador, a frequência de aquisição de dados pelo Digilent, o nome do ficheiro .xlsx e a diretoria em que deve ser gravado, e o aspeto que as folhas de cálculo do mesmo devem assumir.

4.2.3 Aquisição de dados

Relativamente à aquisição de dados do robô, que, como referido, é realizada através do protocolo RTDE, existem algumas questões a assinalar.

Em primeiro lugar, importa referir que o controlador em tempo real do robô possui prioridade sobre a interface RTDE [38], o que significa que, em casos de escassez de poder computacional, o robô diminui a frequência de envio de dados para o computador. Desse modo, em situações nas quais o programa .urp a correr no robô seja demasiado exigente, poderá ser necessário reduzir a frequência de comunicação do protocolo RTDE, de forma a obter um conjunto de dados com uma distribuição temporal estável. Para evitar ter de recorrer a esta solução, procurou-se tornar o programa de inspeção que corre no robô tão simples quanto possível.

Passando para os aspetos relacionados com o próprio computador, o pseudocódigo apresentado no Algoritmo 6 corresponde a uma versão simplificada do ciclo que permite adquirir dados durante a fase de aproximação e permite tirar algumas conclusões acerca de fatores cruciais à programação. De notar que este pseudocódigo apenas difere do que seria apresentado para a fase de inspeção por faltar a parte da aquisição de dados

provenientes do Digilent.

Algoritmo 6 Adquirir dados durante a fase de ajustamento

```
1: Iniciar vetores para receber todos os outputs do robô
2: while True do
3:     Receber output package do robô
4:     while Sonda em contacto com o tejadilho and Programa .urp em execução and
   Conexão estabelecida do
5:         Receber output package do robô
6:         Gravar os dados
7:         if Sinal de fim da fase de aproximação then
8:             Sair do ciclo
9:         end if
10:    end while
11:    if Sinal de fim da fase de aproximação then
12:        Sair do ciclo
13:    end if
14: end while
```

O ciclo externo serve apenas para ser possível, chegando-se a este ponto do programa, atualizar continuamente os dados provenientes do robô até que estejam reunidas as condições para fazer sentido começar a escrever os mesmos (mais especificamente, já existir contacto entre a sonda e o tejadilho, pois caso contrário as medições de força não têm significado físico de relevo), altura em que se entra no ciclo interno.

Já o ciclo interno é onde surgem as maiores restrições. Em cada iteração, o computador aguarda a receção de um pacote de dados proveniente do robô e escreve imediatamente esses dados em vetores, uma vez que, na iteração seguinte, a variável que recebe o pacote de dados será atualizada, perdendo-se os valores anteriores. Desse modo, como a frequência de comunicação é de 500 Hz, o computador tem até 2 ms para realizar a escrita dos dados, caso contrário a iteração seguinte do ciclo não começará a tempo de apanhar o pacote de dados seguinte.

Na fase de inspeção existe a dificuldade adicional de ser necessário, além do que foi referido para a fase de aproximação, recolher e escrever os dados provenientes do Digilent. Como tal, foi necessário tornar ainda mais eficiente o processo de leitura e escrita de dados. Assim, utilizaram-se vetores numpy, por tornarem a escrita de dados mais eficiente [45].

No final de tudo, após ter terminado a inspeção, os valores presentes nos vetores são então escritos nas folhas de cálculo adequadas e o gráfico do sinal obtido pelo Digilent é apresentado.

5

Resultados e Discussão

Neste capítulo, serão apresentados os resultados provenientes dos testes realizados. Em primeiro lugar, será introduzida a plataforma de testes utilizada. Em segundo lugar, abordar-se-á a capacidade de o robô se conseguir ajustar à posição alvo pretendida. De seguida, demonstrar-se-ão evidências da utilidade do controlo de força durante a fase de percorrer o caminho. Por fim, será discutida a capacidade de a montagem experimental e o procedimento desenvolvido permitirem a deteção de defeitos no cordão brasado do tejadilho utilizado como superfície, fazendo face a desalinhamentos no posicionamento do mesmo.

Estão disponíveis vídeos relativos ao funcionamento da metodologia e da interface na seguinte hiperligação: vídeos.

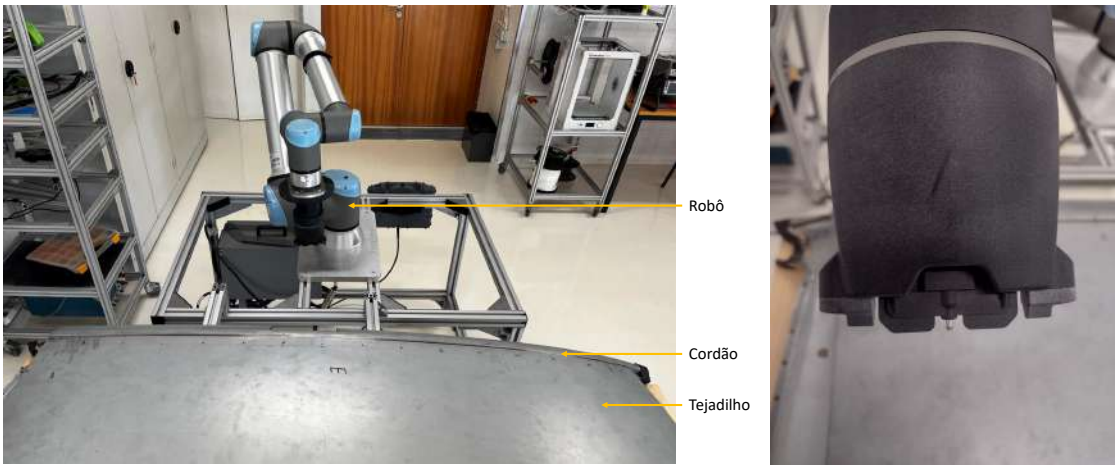
5.1 Plataforma de testes

Como previamente mencionado, a superfície utilizada para os testes foi o tejadilho de um automóvel, existindo então o objetivo de a sonda se acomodar ao cordão brasado presente no mesmo. De seguida, a sonda deve percorrer o cordão, realizando uma inspeção por correntes induzidas.

Na Figura 5.1a apresenta-se o tejadilho, bem como o posicionamento do robô em relação ao mesmo, e na Figura 5.1b a sonda utilizada.

Além disso, a montagem experimental previamente esquematizada na Figura 4.1 encontra-se concretizada na Figura 5.2.

Por fim, é bastante importante clarificar os referenciais que serão mencionados neste capítulo, algo que pode ser observado na Figura 5.3, em que estão ilustrados os referenciais *Base Frame* e *Tool Frame*.



(a) (b)

Figura 5.1: (a) Plataforma de testes; (b) Sonda de correntes induzidas

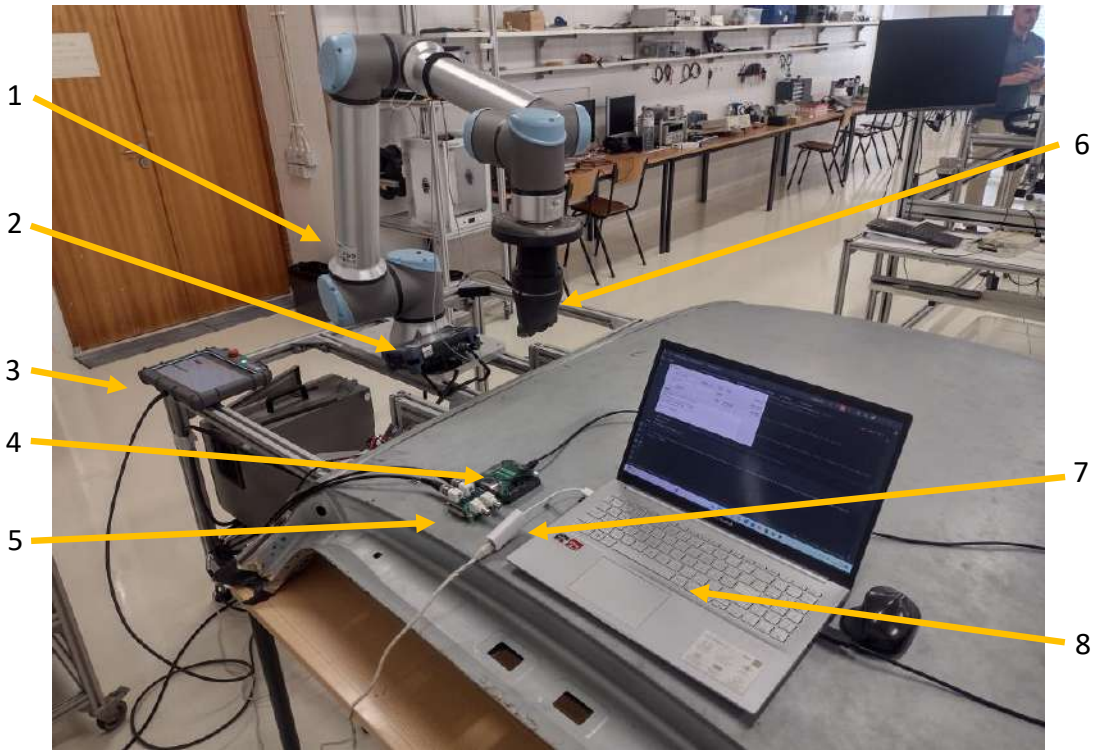
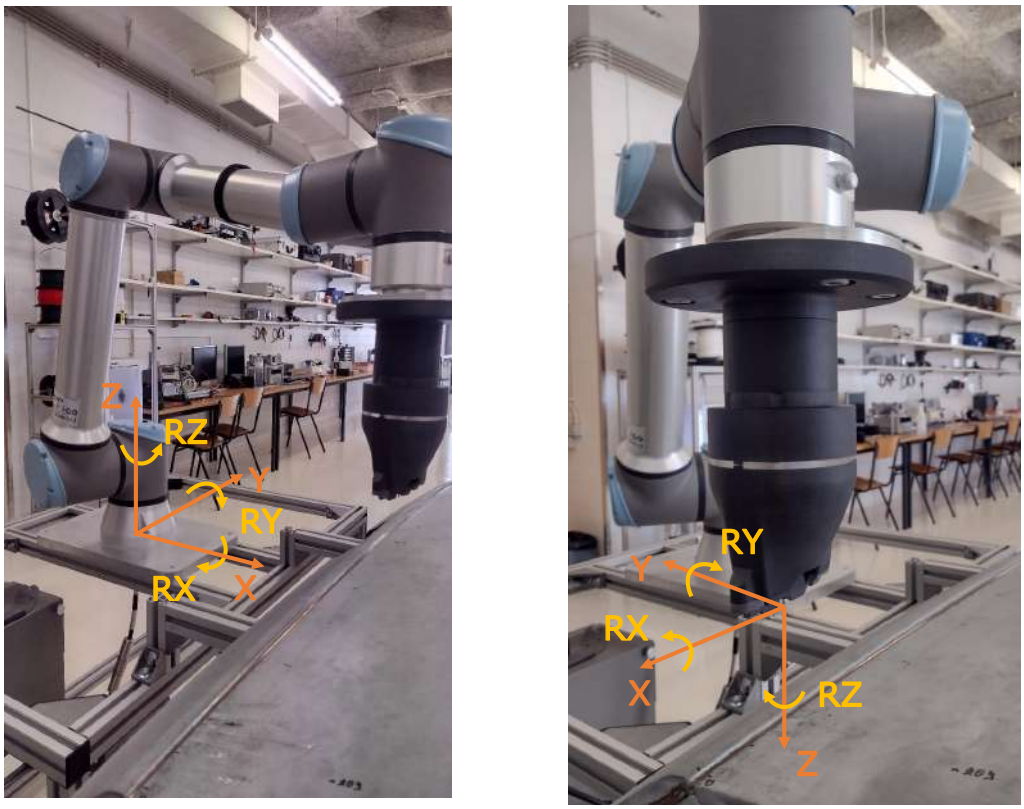


Figura 5.2: Montagem Experimental

1: Robô 2: Analisador de falha 3: *Teach Pendant*
4: Digilent Analog Discovery 2 5: Digilent BNC Adapter 6: Corpo da Sonda
7: Computador externo



(a) (b)

Figura 5.3: Referencial: (a) *Base Frame*; (b) *Tool Frame*

5.2 Fase de ajustamento ao cordão

5.2.1 Definição de parâmetros

Definida a montagem que será utilizada para realizar os testes, é necessário abordar algumas questões relativas ao controlo de força presente no robô UR10e, já que esta funcionalidade será extensamente utilizada, como foi proposto no capítulo 3.

Na linguagem URScript, a linguagem de programação da Universal Robots [46], estão disponíveis três funções distintas que permitem influenciar o modo como o controlo de força se desenrola [46].

Em primeiro lugar, o controlo de força é implementado através da função *force_mode*. Os seus argumentos permitem definir a que *frame* (referencial) se refere o controlo de força, os eixos segundo os quais esse controlo é exercido, as forças e momentos que devem ser aplicados segundo esses eixos, e, por fim, os limites de velocidade (nos eixos controlados por força) e de desvio de posição/orientação (nos eixos controlados posicionalmente). Um aspeto a referir é que, no sistema desenvolvido, o controlo de força foi realizado em todas as situações relativamente à *Tool Frame*.

Além disso, existe a função *force_mode_set_gain_scaling*, cujo argumento é um número entre 0 e 2, sendo 1 o valor *default*, e que permite influenciar o ganho do controlo de força [46]. Por fim, tem-se a função *force_mode_set_damping*, cujo argumento é um número de 0

a 1 que influencia o amortecimento [46].

De notar também que no robô UR10e o controlo de força deve ser sempre associado a um movimento, pelo que as correções realizadas envolveram encontrar uma combinação de controlo de força e de movimento que permitisse atingir os objetivos pretendidos.

Apesar de a lei de controlo de força utilizada pelo robô não ser disponibilizada pelo fabricante, a existência da possibilidade de influenciar o ganho e o amortecimento (*damping*) do controlo de força abre a possibilidade de o mesmo seguir um modelo semelhante a um dos sugeridos em [19], apresentado na Equação 5.1.

$$\tau = \tilde{g}(\theta) + J^T(\theta)\left(\mathcal{F}_d + K_{fp}\mathcal{F}_e + K_{fi}\int \mathcal{F}_e(t)dt - K_{damp}\mathcal{V}\right) \tag{5.1}$$

Nesta equação, τ corresponde ao vetor que contém os *joint torques* . Por outro lado, $\tilde{g}(\theta)$ consiste num modelo dos torques gravitacionais, que depende de θ, um vetor com os *joint angles*. Já $J^T(\theta)$ representa a transposta da jacobiana do vetor com os *joint angles*. Por sua vez, $\mathcal{F}_d$ é um vetor com os valores de forças e momentos pretendidos pelo utilizador. Já os termos K_{fp} e K_{fi} correspondem às matrizes de ganho proporcional e integral respetivamente. Tem-se também $\mathcal{F}_e$, que é dado por $\mathcal{F}_e = \mathcal{F}_d - \mathcal{F}_{tip}$, em que $\mathcal{F}_{tip}$ corresponde à leitura efetuada por um sensor de força na junta em que o *end-effector* está conectado ao robô. Por fim, K_{damp} corresponde a uma matriz de amortecimento (*damping*) e $\mathcal{V}$ a um vetor que contém as velocidades linear e rotacional do *end-effector*, ou seja, $\mathcal{V} = (\omega, v)$ [19].

Contudo, uma vez que o robô também é controlado posicionalmente nas direções segundo as quais não se define controlo de força, devem ser acrescentados alguns termos à Equação 5.1, obtendo-se a Equação 5.2. Esta equação de controlo constitui um exemplo de *Hybrid Motion-Force Control* [19], e corresponde a uma hipotética versão da que é realmente aplicada no robô utilizado.

$$\begin{aligned}\tau =& J_b^T(\theta)\Big((P(\theta)L_{cp}(\theta,t)) \\ &+ (I - P(\theta))\left(\mathcal{F}_d + K_{fp}\mathcal{F}_e + K_{fi}\int \mathcal{F}_e(t)dt - K_{damp}\mathcal{V}\right) \\ &+ \tilde{\eta}(\theta, \mathcal{V}_b)\Big)\end{aligned} \tag{5.2}$$

Nesta equação, o índice b denota o facto de os vetores de força/momento e de velocidade linear/rotacional estarem definidos no referencial do *end-effector* (*Tool Frame*). Por sua vez, $P(\theta)$ e $I - P(\theta)$ correspondem a matrizes de seleção que permitem que o controlo de força seja apenas efetuado segundo as direções definidas pelo utilizador, sendo as restantes direções controladas apenas posicionalmente. Já $L_{cp}(\theta, t)$ simboliza uma lei de controlo posicional genérica. Por fim, $\tilde{\eta}(\theta, \mathcal{V}_b)$ é um termo que se refere a compensações devido a efeitos como a gravidade ou a aceleração de Coriolis. Os restantes termos são iguais à Equação 5.1.

Dito isto, serão agora então apresentados alguns dos resultados obtidos, que permitiram chegar a valores para os parâmetros relativos à fase de ajustamento, que foram abordados no capítulo 3.

5.2.1.1 Fase 1

Na Fase 1 (ver 3.2.2), de forma a estabelecer um contacto adequado dos dois apoios da sonda com o tejadilho, existem algumas questões a ter em conta, que se tornaram evidentes nos testes que foram realizados para esta etapa.

Estes testes, que serão referenciados ao longo de 5.2.1.1, corresponderam a realizar a sequência de movimentos da Fase 1, cuja sequência é visível na Figura 5.4, mas variando parâmetros como a força aplicada e o período de estabilização, ou seja o intervalo de tempo atribuído a esta correção. Além disso, fez-se variar a posição inicial, testando vários desalinhamentos iniciais segundo RY (*Tool Frame*) para perceber quais os desvios máximos passíveis de ser corrigidos pelas diversas combinações de parâmetros.

(a) (b) (c)

Figura 5.4: Sequência inicial: (a) Posição inicial; (b) Contacto com o tejadilho; (c) Reorientação segundo RY (*Tool Frame*)

Em primeiro lugar, é relevante salientar que existe um limite ao desvio inicial permitido segundo RY (*Tool Frame*), pois, a partir de um certo valor, a sonda torna-se instável e realiza uma rotação para o lado oposto ao necessário para corrigir a orientação. Um exemplo disso está presente na sequência da Figura 5.5.

Numa tentativa de mitigar este problema, realizaram-se testes com aplicação de forças de diferentes magnitudes segundo o sentido positivo de Z (*Tool Frame*). Na sequência destes testes, observou-se que uma força mais elevada tendia a produzir melhores resultados, na medida em que permitia corrigir desalinhamentos mais elevados. Contudo, apresentando o tejadilho uma rigidez relativamente reduzida em algumas zonas, acabou por ter de ser encontrado um equilíbrio que combinasse uma boa capacidade de reorientação da ferramenta sem correr o risco de danificar o tejadilho

(a) (b) (c)

Figura 5.5: Falha na correção inicial: (a) Posição inicial; (b) Contacto com o tejadilho; (c) Reorientação no sentido errado

Assim, o limite máximo de correção e a força aplicada segundo o sentido positivo de Z (*Tool Frame*) estão relacionados, pelo que se procederam a testes para perceber qual a correção máxima possível para cada força, sendo também registada a deformação provocada no tejadilho para cada uma. Os resultados estão presentes na Tabela 5.1.

Tabela 5.1: Testes de força aplicada durante a Fase 1

Força [N]	Desvio RY máximo [°]	Deformação provocada
30	5	Bastante reduzida
45	8	Reduzida
60	9	Elevada

Face a estes resultados, optou-se pelo valor de 45 N, por permitir a correção de desvios mais elevados do que os 30 N, sem uma deformação significativa do tejadilho. Descartou-se a utilização de 60 N por não permitir uma melhoria significativa relativamente aos 45 N, e por provocar uma deformação mais significativa no tejadilho.

Outro parâmetro de extrema importância é o limite da velocidade de rotação segundo RY (*Tool Frame*): uma velocidade demasiado elevada provocava sobrecorreções que por vezes originavam erros de orientação dos quais era impossível recuperar. Um exemplo deste tipo de erros está presente na sequência da Figura 5.6. Contudo, uma velocidade demasiado reduzida significa que este período de estabilização seria muito demorado, prejudicando os tempos de ciclo.

Desse modo, após definir a força de contacto como sendo de 45 N, realizaram-se alguns testes de forma a tentar perceber qual o valor máximo de velocidade de correção que poderia ser utilizado sem que ocorressem sobrecorreções irrecuperáveis. Foram testadas as velocidades de $0,05$ rad/s, $0,06$ rad/s, $0,07$ rad/s e $0,08$ rad/s e concluiu-se que $0,07$ rad/s era o limite máximo, pelo que foi este o valor incorporado no sistema proposto.

Após um determinado período de estabilização, considera-se a sonda suficientemente

(a) (b) (c)

Figura 5.6: Sobrecorreção na etapa inicial: (a) Posição inicial; (b) Contacto com o tejadilho; (c) Sobrecorreção

Tabela 5.2: Testes ao período de estabilização durante a Fase 1

Período de estabilização [s]	Desvio RY [°]		
	2	5	8
2	Corrigido	Corrigido	Não corrigido
3,5	Corrigido	Corrigido	Corrigido
5	Inadequado	Corrigido	Corrigido

bem alinhada e avança-se para a próxima etapa. Para definir este intervalo, procurou-se um equilíbrio: um desvio inicial mais elevado necessita de maior tempo de correção, mas como a correção é de maior amplitude propicia-se o acontecimento de sobrecorreção, pelo que o intervalo de tempo também não pode ser excessivamente elevado.

Para poder perceber qual a duração necessária para este período de estabilização, realizaram-se testes para avaliar a capacidade de correção de desvios de 2°, 5° e 8°, segundo RY (*Tool Frame*).

Para cada um dos desvios referidos, os períodos de estabilização avaliados foram de 2 s, 3, 5 s e 5 s. Os resultados encontram-se resumidos na Tabela 5.2 e serão detalhados de seguida.

Ora, para o período de estabilização de 2 s, percebeu-se que para os desvios mais elevados (8°) existiam várias ocasiões em que não chegava a verificar-se uma reorientação suficiente para que ambos os apoios da sonda se encontrassem em contacto com o tejadilho, algo ilustrado na Figura 5.7.

Já para o período de estabilização de 5 s, os problemas surgiam durante a tentativa de correção de desvios menos elevados (2°). Por um lado, na maior parte das ocasiões a correção era feita com relativa rapidez, ficando a sonda praticamente imóvel, meramente à espera do fim do período de estabilização.

Noutras ocasiões, contudo, em vez de a sonda ficar imóvel, começava a tornar-se instável, iniciando uma oscilação cada vez maior, que acabava por fazer com que, no

Figura 5.7: Reorientação insuficiente

final do período de estabilização, já não estivesse com os dois apoios em contacto com o tejadilho.

As razões para o aparecimento desta instabilidade são algo incertas, mas uma conjetura possível tem a ver com a incerteza relativamente elevada nas medições de força, que sofriam alguma flutuação. Caso subitamente fosse detetado um valor um pouco mais anómalo, o controlo de força poderia considerar que a sonda não se encontrava apoiada nos dois pontos de contacto, iniciando uma tentativa de correção que, na verdade retirava a sonda da posição desejada.

Assim, acabou por considerar-se que o valor intermédio de $3,5$ s para o período de estabilização seria o mais adequado, por ser de alguma forma o melhor de dois mundos, fornecendo uma correção adequada para os desvios mais elevados, mas não causando tempos de inatividade excessivos na correção de desvios mais reduzidos.

Desta conjunção de parâmetros resulta que o desvio máximo que pode ser corrigido é de 8°, segundo RY (*Tool Frame*).

5.2.1.2 Fase 2

Na Fase 2 (ver 3.2.3), em que existe uma aproximação ao cordão brasado, a preocupação é manter o contacto dos dois apoios com o tejadilho, ao longo do movimento, até que o contacto com o ressalto (ver Figura 5.8) seja detetado.

Como previamente referido, tal é atingido através da combinação de *compliance* segundo RY e de uma força segundo o sentido positivo de Z (*Tool Frame*). Desse modo, devem ser avaliados os seguintes parâmetros: velocidade de correção segundo RY e magnitude da força segundo Z.

O modo como se procedeu aos testes para determinar os parâmetros ótimos foi simplesmente executar a sequência de movimentos desta fase, que consiste na aproximação ao cordão até ser detetado o cordão, variar os parâmetros, e retirar algumas observações qualitativas.

Figura 5.8: Ressalto no tejadilho

Relativamente à magnitude da força, foram considerados os valores 20 N, 30 N, 40 N. Os testes efetuados revelaram que com uma força de 20 N, existia uma capacidade insuficiente de manter ou repor o contacto de ambos os apoios com o tejadilho.

No entanto, o valor de 40 N também revelou alguns problemas. Por um lado, por este ser um movimento de arrasto, notou-se que uma força de intensidade excessiva causava algum desgaste no tejadilho. Por outro, também se constatou que provocava um aumento da trepidação durante o movimento, causando pequenas oscilações segundo RY por causa da *compliance* definida, e, consequentemente, mais perdas de contacto de um dos apoios com o tejadilho.

Desse modo, escolheu-se o valor intermédio de 30 N, um valor elevado o suficiente para permitir as correções necessárias, mas não tão alto que provocasse desgaste no tejadilho ou trepidação excessivos.

No que concerne à velocidade de correção segundo RY, testaram-se os valores de $0,01$ rad/s, $0,02$ rad/s, e $0,03$ rad/s.

Os testes confirmaram a hipótese avançada em 3.2.3, de que a trepidação inerente ao movimento de aproximação ao cordão faz com que velocidades de correção demasiado elevadas provoquem instabilidade na sonda. De facto, isso verificou-se para a velocidade de $0,03$ rad/s.

Isto deve-se ao facto de a trepidação provocar flutuações na leitura do momento MY, que são agravadas quando se procura uma correção demasiado rápida, por exemplo por existir um reestabelecimento demasiado brusco do contacto de um dos apoios com o tejadilho.

Assim, a velocidade deve ser tão reduzida quanto possível, desde que essa velocidade reduzida ainda permita realizar as correções necessárias.

Contudo, também se verificou que a velocidade de $0,01$ rad/s nem sempre tinha a capacidade de corrigir adequadamente os desalinhamentos segundo RY (*Tool Frame*).

Desse modo, optou-se pela velocidade de correção de 0, 02 rad/s, por não provocar a instabilidade referida e por ser capaz de corrigir os eventuais desalinhamentos decorrentes de execuções menos adequadas da Fase 1.

5.2.1.3 Fase 3

Relativamente à Fase 3, em que ocorre pela primeira vez um ajustamento ao cordão, existem várias questões relevantes, no que concerne ao controlo de força realizado.

Em primeiro lugar, realça-se que o modo como se realizaram os testes para definir os parâmetros desta etapa consistiu em realizar a sequência de movimentos correspondente às Fases 0 a 3, partindo de diversas posições iniciais, definidas na Tabela 5.3 em relação ao referencial apresentado na Figura 5.9. Este referencial tem origem no início do cordão, encontrando-se o eixo Y alinhado com o mesmo, e sendo o eixo Z perpendicular ao plano do cordão. Para testar cada parâmetro, realizaram-se três ensaios por cada posição inicial.

Tabela 5.3: Posições iniciais (início da Fase 0) dos testes para a Fase 3

Posição	X [mm]	Y [mm]	Z [mm]	RX [°]	RY [°]	RZ [°]
1	25	55	25	-8	0	-1
2	50	55	25	0	0	-5
3	75	55	25	8	0	5

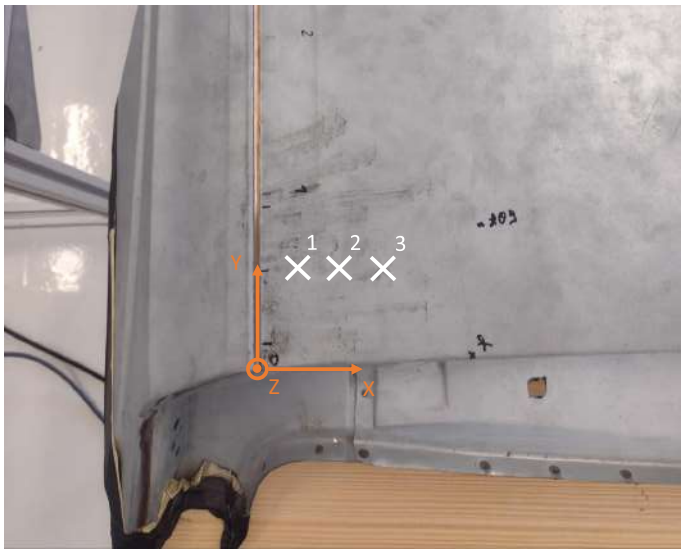


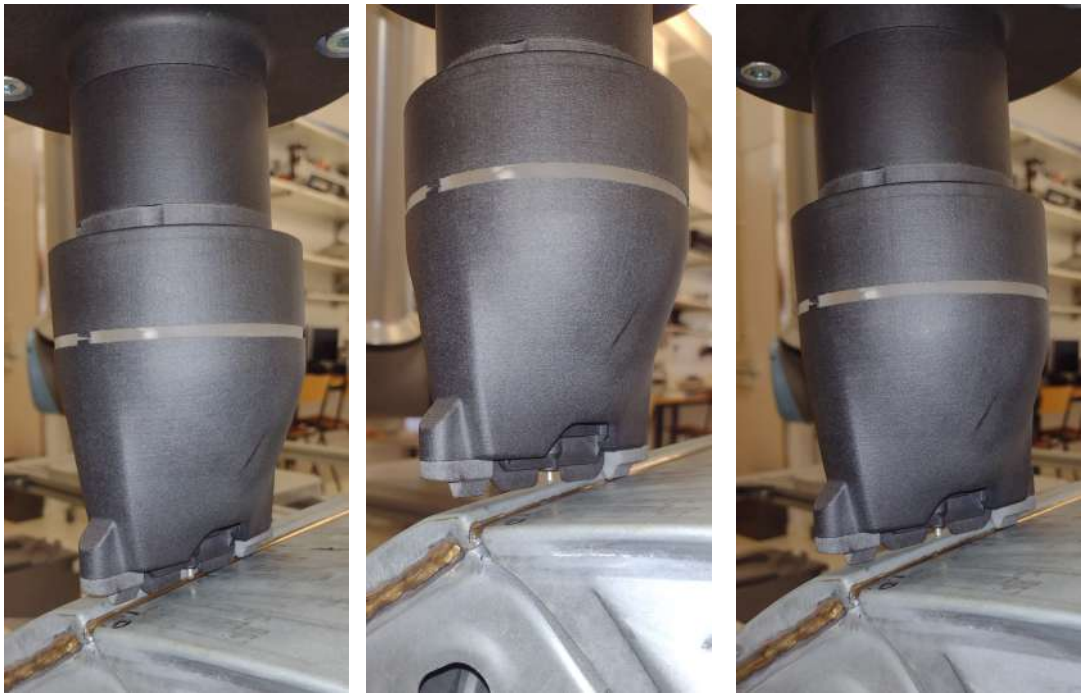
Figura 5.9: Diagrama representativo das posições iniciais (início da Fase 0) dos testes para a Fase 3

Os testes revelaram que a posição inicial 1 era a menos favorável, pela combinação entre reduzida distância segundo X ao cordão, elevado desvio RX e reduzido desvio RZ. Isto porque, nesta situação, nem sempre era possível corrigir na totalidade o desvio RX durante as Fases 1 e 2.

Note-se que, em termos da programação do robô, uma vez que o controlo de força foi definido com base no referencial *Tool Frame*, as correções referidas em termos de RX serão

doravante referidas como correções em termos de RY. Isto porque RX no referencial da Figura 5.9 corresponde (com sentidos opostos) a RY no *Tool Frame*.

Os testes para esta posição inicial 1 demonstraram que, durante a Fase 3, não podem existir simplesmente correções alternadas de RY e RZ (*Tool Frame*), como inicialmente proposto. De facto, se no início desta fase for feita meramente uma correção segundo RZ, existe a possibilidade de ocorrer um erro como o ilustrado na Figura 5.10, em que a sonda ultrapassa o ressalto, não sendo possível recuperar desta situação.



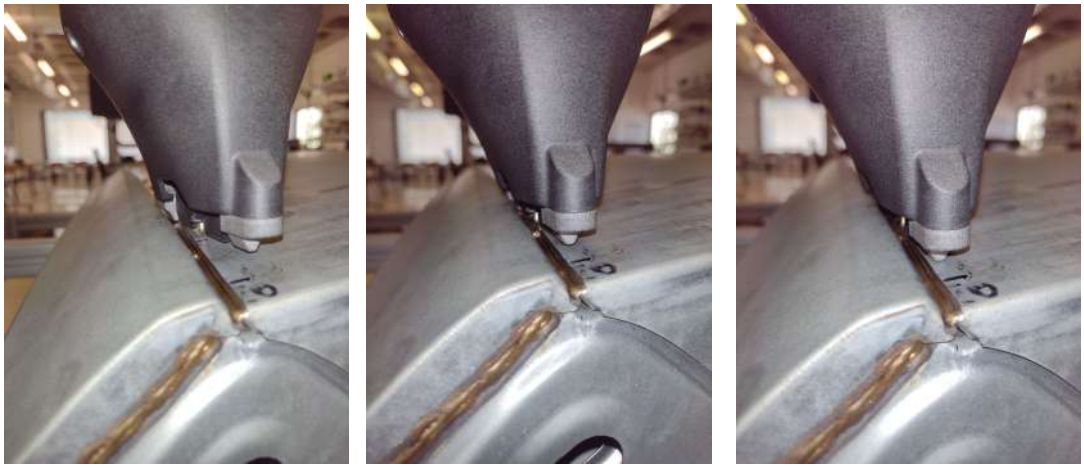
(a) (b) (c)

Figura 5.10: Erro no ajustamento: (a) Primeiro contacto com o ressalto; (b) Etapa de reorientação segundo RY e RZ (*Tool Frame*); (c) Etapa de reorientação apenas segundo RY (*Tool Frame*)

Para evitar a ocorrência destes erros, verificou-se que era preferível fazer pequenas correções segundo RZ de cada vez, mas simultaneamente mantendo *compliance* segundo RY. Seguidamente a cada pequena etapa de correção em simultâneo de RY e RZ, deve também existir uma etapa de reorientação apenas segundo RY, para evitar que o desvio segundo esta última orientação se torne tão elevado que deixe de ser possível através de mero controlo de força corrigi-lo. A sequência é ilustrada na Figura 5.11.

Para garantir que as correções segundo RZ são feitas aos poucos optou-se por um ciclo While. Se a correção durar mais de $0,4$ s ou envolver uma rotação RZ (*Tool Frame*) de mais de $1,5°$, o ciclo é interrompido

Utiliza-se esta disjunção de critérios porque caso apenas se considerasse a duração do movimento não seria possível confirmar que a sonda está a convergir para a posição desejada, através de movimentos corretivos cada vez menores segundo RZ (*Tool Frame*). Por outro lado, se apenas se utilizasse como limite a rotação RZ, em cada iteração de



(a) (b) (c)

Figura 5.11: Ajustamento inicial ao cordão: (a) Primeiro contacto com o ressalto; (b) Etapa de reorientação segundo RY e RZ (*Tool Frame*); (c) Etapa de reorientação segundo RY (*Tool Frame*)

correção o robô tentaria sempre continuar a mover-se até atingir a rotação pré-definida, mesmo que já tivesse atingido a posição pretendida.

Como referido, seguidamente a esta etapa de reorientação simultânea segundo RY e RZ, passa-se a uma etapa exclusiva de reorientação segundo RY durante 1 s.

Relativamente aos valores arbitrados, estes acabaram por ser aqueles que, durante os testes, permitiram assegurar tanto um ajustamento rápido, como um ajustamento com repetibilidade. Neste contexto, repetibilidade significa que nos vários testes efetuados estes parâmetros não conduziam à situação da Figura 5.10, além de garantirem um bom ajustamento ao cordão, algo determinado por inspeção visual.

Por fim, de referir que esta fase de reorientação se considera como terminada quando a última correção segundo RZ for inferior a um determinado valor, que teve de ser determinado a partir dos testes realizados. A lógica por detrás deste procedimento é considerar-se que se a última correção não foi significativa (ou seja, inferior a um determinado valor), então o programa já convergiu para a solução e a sonda já está tão bem acomodada como possível, pelo menos utilizando a metodologia descrita nesta etapa.

Os valores testados para o limiar abaixo do qual se considera terminada a reorientação em RZ foram $0,2°$, $0,5°$ e $0,8°$.

Em algumas ocasiões, os testes com o valor de $0,2°$ implicavam fases de ajustamento extremamente demoradas, por vezes superiores a um minuto. Isto porque a natureza oscilatória desta fase de ajustamento fazia com que fosse bastante difícil que durante a fase de rotação segundo RZ o valor da rotação fosse assim tão reduzido.

Já os testes com o valor de $0,8°$ conduziram por vezes a ajustamentos um pouco mais grosseiros, após inspeção visual, na medida em que este não é um critério tão apertado para definir o fim da fase de ajustamento.

Assim, optou-se pelo valor de $0,5°$ como o limiar abaixo do qual se considera o

ajustamento terminado.

5.2.1.4 Fase 4

Como referido em 3.3.1, nesta fase poderá ser conveniente realizar uma pausa no movimento da sonda e esta pausa deverá ser tão reduzida quanto possível.

Para determinar a duração adequada desta pausa, executaram-se as Fases 0 a 4, partindo-se da posição inicial 2 (ver Tabela 5.3 e Figura 5.9), e testaram-se as durações 0, 2 s, 0, 5 s, 1 s e 1, 5 s. Para cada duração realizaram-se três ensaios.

Nos testes realizados, assistiu-se que a paragem repentina provocava o início de oscilação do tejadilho, que apenas após cerca de 1 s se atenuava o suficiente para que, por inspeção visual, pudesse ser considerada negligenciável. Desse modo, optou-se por definir uma duração de 1 s para a Fase 4

5.2.1.5 Fases 5, 6, 7, 8 e 9

Nas Fases 5 e 6, que correspondem ao movimento realizado pela sonda de forma a procurar o início da superfície (cordão), foi necessário definir a força que é exercida sobre o cordão. Realizaram-se testes com intensidades de 0, 5 N, 5 N e 10 N. Aquilo que foi observado foi que a força não possuía uma influência significativa no comportamento do conjunto robô e sonda, pelo que se optou pelo valor mais reduzido, de 0, 5 N.

Relativamente à Fase 7, uma vez que corresponde ao movimento livre do robô sobre o tejadilho, não se realizaram testes extensivos.

Uuma vez que o movimento na Fase 8 o movimento é análogo ao realizado nas Fases 5 e 6, optou-se pelo mesmo valor de força de contacto.

Por fim, a Fase 9, de ajustamento final segundo RY (*Tool Frame*), é semelhante à Fase 1.

5.2.2 Ensaios de ajustamento ao cordão

Após os diferentes testes realizados para definir os parâmetros para cada uma das fases do programa de ajustamento da sonda ao cordão, é necessário demonstrar a capacidade de funcionamento do programa de ajustamento como um todo. Para tal, projetou-se o conjunto de testes que será seguidamente detalhado.

Em primeiro lugar, definiu-se a posição alvo, que foi assinalada no tejadilho com uma caneta permanente, como demonstrado na Figura 5.12. Sendo esta a posição final do ajustamento, corresponde ao ponto de partida para a fase seguinte, a de inspeção. Nesta figura, estão assinaladas respetivamente a amarelo, branco e vermelho as localizações em que o apoio posterior do corpo da sonda, a ponta da sonda, e o apoio anterior devem estar no final do ajustamento.

De seguida, definiram-se três posições iniciais, que o programa deveria corrigir, de forma a ser alcançada a posição alvo consistentemente. Estas posições estão definidas na Tabela 5.4, em relação ao referencial definido na Figura 5.13, que tem origem no ponto em

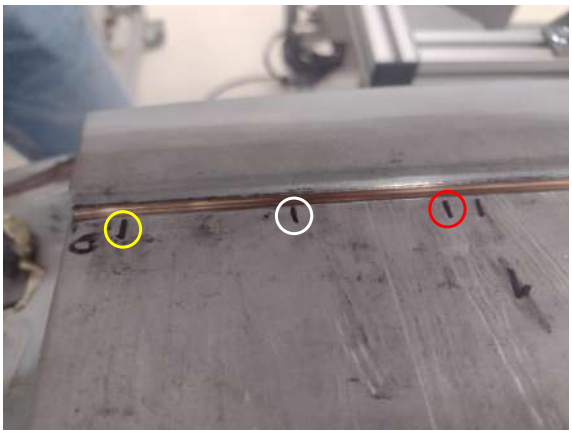


Figura 5.12: Marcação da posição alvo

que o TCP se encontra quando a sonda está na posição alvo. Nesta Figura 5.13 também se encontram assinaladas as três posições iniciais referidas.

Tabela 5.4: Desvios entre as posições iniciais e a posição alvo

Posição	X [mm]	Y [mm]	Z [mm]	RX [°]	RY [°]	RZ [°]
1	50	50	25	-5	0	-5
2	75	30	25	5	0	5
3	100	10	25	5	0	-5

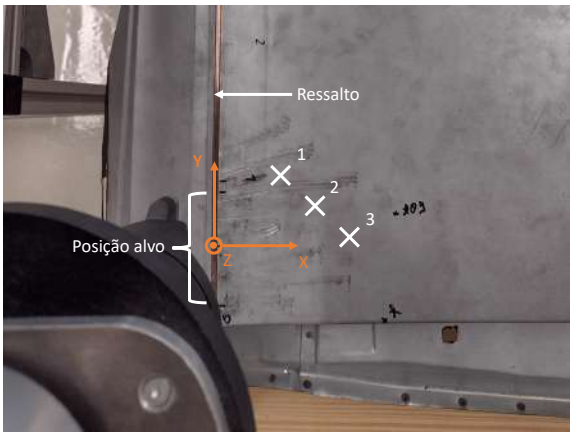


Figura 5.13: Diagrama representativo das posições iniciais dos ensaios

É relevante sublinhar que a razão pela qual as posições iniciais de teste apresentam desvios em termos de posição e orientação da posição alvo se devem a uma tentativa de simular desalinhamentos no tejadilho. A montagem experimental existente não permite facilmente criar e quantificar adequadamente esses desvios, pelo que, em vez de criar desalinhamentos no tejadilho, criaram-se desalinhamentos na posição inicial da sonda. Ainda assim, no subcapítulo 5.3.2.2, serão apresentados testes em que foram criados alguns desalinhamentos no tejadilho (quantificados tão rigorosamente quanto as condições o permitiram), de forma a procurar simular melhor as condições reais de funcionamento.

Definidas as posições iniciais, realizaram-se cinco ensaios para cada uma delas, nos quais a posição alvo foi sempre alcançada. Na Figura 5.14, encontra-se ilustrada uma sequência de ajustamento para a posição inicial 1, sendo que para as restantes posições iniciais as figuras seriam análogas.

(a)

(b)

(c)

Figura 5.14: Posição de teste 1: (a) Fim da Fase 0/Início da Fase 1; (b) Fim da Fase 3/Início da Fase 4; (c) Fim da Fase 9 (posição final)

De forma a poder quantificar-se a consistência do ajustamento realizado, para além de mera inspeção visual, calcularam-se os desvios (segundo o referencial da Figura 5.13) entre a posição alvo e as posições finais atingidas nos cinco ensaios de cada posição inicial. Estes dados estão presentes nas Tabelas 5.5, 5.6, 5.7 e 5.8. De notar que os valores para os desvios foram calculados através da simples subtração das componentes, como será exemplificado na Equação 5.3 para a componente de posição X.

$$\Delta X = X_{teste} - X_{alvo} \tag{5.3}$$

Começando a análise pelos desvios segundo X, estes são, de facto, os mais elevados. Contudo, o facto de serem sempre desvios negativos é encorajador, uma vez que significa que o ajustamento da sonda ao cordão foi feito de forma a que a sonda se encontrasse

Tabela 5.5: Desvios entre as posições finais e a posição alvo (*Base Frame*): posição inicial 1

Ensaio	ΔX [mm]	ΔY [mm]	ΔZ [mm]	ΔRX [rad]	ΔRY [rad]	ΔRZ [rad]
1	-1,00	-0,42	0,25	0,00098	0,00217	-0,00123
2	-0,97	-0,67	0,24	0,00103	0,00305	-0,00116
3	-1,00	-0,57	0,50	0,00063	0,00103	-0,00078
4	-0,94	-0,67	0,32	0,00050	0,00221	-0,00105
5	-0,96	-0,36	0,09	0,00099	0,00163	-0,00123

Tabela 5.6: Desvios entre as posições finais e a posição alvo (*Base Frame*): posição inicial 2

Ensaio	ΔX [mm]	ΔY [mm]	ΔZ [mm]	ΔRX [rad]	ΔRY [rad]	ΔRZ [rad]
1	-1,31	-0,69	0,08	-0,00142	-0,00003	0,00066
2	-1,26	-0,29	0,16	-0,00169	0,00106	-0,0009
3	-1,31	-0,28	0,21	-0,0016	0,00249	-0,00215
4	-1,13	-0,38	0,25	-0,00122	0,00232	-0,00151
5	-1,17	-0,74	0,25	0,0012	0,00299	-0,00125

Tabela 5.7: Desvios entre as posições finais e a posição alvo (*Base Frame*): posição inicial 3

Ensaio	ΔX [mm]	ΔY [mm]	ΔZ [mm]	ΔRX [rad]	ΔRY [rad]	ΔRZ [rad]
1	-0,56	-0,29	0,19	-0,00192	0,00218	-0,00009
2	-0,31	-0,08	0,15	-0,00219	0,00231	-0,00122
3	-0,50	-0,32	0,17	-0,0024	0,0002	-0,00044
4	-0,27	-0,38	0,35	-0,00188	0,00423	-0,00307
5	-0,42	-0,03	0,32	-0,00214	0,00341	-0,00262

Tabela 5.8: Valores médios dos desvios de todas as posições iniciais

ΔX [mm]	ΔY [mm]	ΔZ [mm]	ΔRX [rad]	ΔRY [rad]	ΔRZ [rad]
-0,87	-0,41	0,24	0,00098	0,00208	0,0012

ainda mais próxima do ressalto do tejadilho do que o previsto na posição alvo. Ora a fase seguinte, de inspeção, envolve controlo de força no sentido de manter a sonda encostada a este ressalto para não existir perda do cordão por parte da sonda. Desse modo, estes resultados para o desvio segundo X acabam por conferir alguma segurança no que toca a garantir que essa perda de contacto com o cordão não acontece.

Passando aos valores dos desvios segundo Y, o facto de estes serem bastante reduzidos, com um valor médio de $-0,41$ mm e um valor máximo de $-0,74$ mm, permite validar a parte do programa responsável por garantir que a sonda começa a inspeção sempre na mesma posição ao longo da direção longitudinal do cordão.

Já os valores reduzidos de desvio segundo Z são positivos, por indicarem que, no final do ajustamento, a sonda tem um bom contacto com o cordão.

Por fim, os valores de desvio em termos de orientação em RX, RY e RZ são provavelmente os resultados mais importantes no que concerne à viabilidade da fase seguinte, de inspeção, por serem os que mais influenciam a estimativa do trajeto a realizar. Ao analisar as Tabelas 5.5 a 5.7, é possível concluir que o desvio mais elevado é verificado para RY, no

ensaio 4 da posição inicial 3, tendo o valor de $0,00423$ rad, o que corresponde a cerca de $0,24°$.

O facto de globalmente os valores para os desvios terem sido reduzidos e o facto de o ajustamento ter passado na inspeção visual parecem permitir concluir que a solução desenvolvida é capaz de realizar a tarefa pretendida: partindo de uma posição aleatória sobre o tejadilho, ajustar-se com sucesso ao cordão.

Relativamente ao tempo de ajustamento, apresentam-se na Tabela 5.9 os resultados obtidos. Estes tempos referem-se ao intervalo entre o instante em que a sonda se encontra na posição inicial e o instante em que a fase do programa relativa ao ajustamento termina.

Tabela 5.9: Tempos de ajustamento

Posição	Tempo [s]
1	25
2	26
3	25

Estes tempos de ajustamento são relativamente elevados, mas tal pode ser parcialmente explicado pelo facto de corresponderem a correções de desvios bastante acentuados. Assim, apesar de desvios elevados serem úteis para testar os limites do programa, decidiu-se realizar um teste a partir de uma quarta posição inicial, com desalinhamentos menos pronunciados. Os dados para este teste encontram-se na Tabela 5.10, sendo utilizado o referencial da Figura 5.13.

Tabela 5.10: Desvios da posição inicial 4 e respetivo tempo de ajustamento

X [mm]	Y [mm]	Z [mm]	RX [°]	RY [°]	RZ [°]	Tempo [s]
10	10	20	1	0	1	18

Conclui-se que existe uma diferença significativa entre os tempos de ajustamento apresentados nas Tabelas 5.9 e 5.10, podendo então balizar-se aproximadamente o mesmo entre 18 s para posições mais fáceis de corrigir e 26 s para desvios mais elevados.

Por fim, uma vez que a maioria das correções efetuadas durante o ajustamento envolvem controlo de força, apresentam-se na Figura 5.15 os gráficos correspondentes à força FZ e ao momento MY. Estes são os que melhor permitem identificar as diferentes fases que decorrem durante o ajustamento ao cordão, que se encontram numeradas na figura. Os gráficos foram retirados do ensaio 1 da posição inicial de teste 1.

A Fase 1 (ver Figuras 5.4b e 5.4c) corresponde ao instante após o contacto ser estabelecido com o tejadilho, e algo que imediatamente salta à vista é a oscilação de FZ em torno de -45 N, algo que se coaduna com o valor definido para o controlo de força (ver 5.2.1.1).

Já o valor de MY inicia negativo, o que é coerente com o modo como a sonda contacta com o tejadilho (ver Figura 5.14a), passando pouco depois a positivo, devido a uma ligeira sobrecorreção.

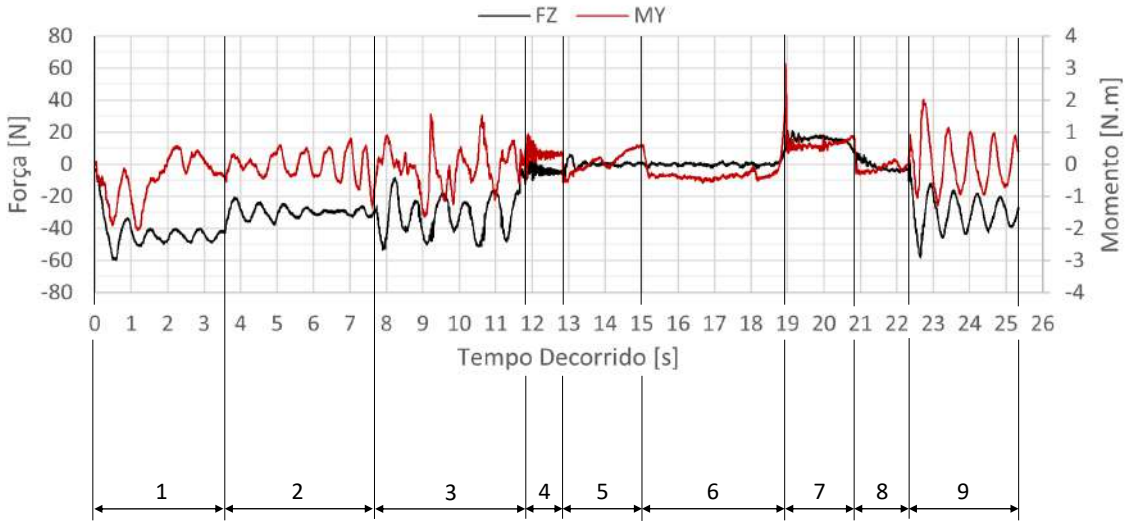


Figura 5.15: Gráfico da força FZ e do momento MY exercidos sobre a sonda ao longo das Fases 1 a 9, em função do tempo decorrido durante a fase de ajustamento ao cordão a partir da posição inicial 1 (*Tool Frame)*

Contudo, o seu valor aproxima-se para 0 N antes do final desta fase, o que significa que a sonda se reorientou de forma a ter os dois apoios em contacto com o tejadilho, como desejado.

Por sua vez, na Fase 2, uma vez que corresponde à aproximação à zona do cordão, os valores de FZ e MY deveriam ser constantes, em teoria. O que se observa na verdade é que oscilam em torno de um valor constante, o que é natural, dada a trepidação verificada durante o movimento.

A Fase 3 (ver Figura 5.11) corresponde ao período de tempo após existir o primeiro contacto entre a sonda e o ressalto no tejadilho, no qual a sonda se procura ajustar ao cordão, sendo por isso natural existir flutuação nos valores medidos. Por sua vez, a Fase 4 corresponde a uma pausa de 1 s antes do início da etapa do programa responsável por encontrar a posição inicial de inspeção adequada ao longo da direção longitudinal do cordão. Essa etapa está representada pelas Fases 5, 6, 7 e 8.

Uma explicação para o facto de na Fase 4 existir uma flutuação intensa dos valores pode dever-se ao facto de a pausa ser iniciada pela ativação dos travões do robô, pelo que se verifica uma paragem muito repentina. Uma vez que a estrutura que suporta o tejadilho não está fixa rigidamente ao chão e que o próprio tejadilho não está rigidamente fixo à estrutura, essa paragem repentina impõe oscilação no conjunto. Assim, encontrando-se o conjunto estrutura+tejadilho a oscilar e encontrando-se o robô travado, é natural que surja uma flutuação nos valores de força e momento medidos.

No início da Fase 5, parece existir uma pequena oscilação de MY, mas tal deve-se ao facto de a força FZ demorar uns instantes a estabilizar-se.

De seguida, ainda durante a Fase 5, a partir da abcissa 14 s, nota-se um aumento no valor de MY. O motivo pelo qual tal acontece não é claro, mas talvez tenha a ver com

alguma incerteza acerca de que apoio está em contacto com o cordão, por ser uma altura de transição em que o apoio anterior da sonda começa a perder o contacto com o cordão devido ao movimento ilustrado na Figura 5.16 (Fases 5, 6 e 7).

(a) (b) (c)

Figura 5.16: Encontro do ponto ideal: (a) Posição inicial (Fim da Fase 4/Início da Fase 5); (b) Sonda parcialmente fora do cordão (Fase 6); (c) Sonda totalmente fora do cordão (Fim da Fase 6/Início da Fase 7)

Quando essa perda é total, o valor de MY diminui repentinamente, tornando-se negativo, algo que faz sentido, uma vez que toda a força de contacto está no apoio posterior, fazendo com que a sonda tenha a tendência de sofrer uma rotação no sentido negativo de RY (ver Figura 5.16b).

Após esta queda abrupta, inicia a Fase 6. Uma vez que a força de contacto (FZ) é constante e o atrito também, devido ao facto de a área de contacto se manter relativamente constante, o valor de MY mantém-se constante durante a maior parte desta fase. Contudo, no final, quando o apoio posterior também começa a perder contacto com o cordão, o valor de MY começa a aumentar ligeiramente, podendo este fenómeno ter uma explicação análoga à que foi referida para a perda de contacto do apoio anterior (redução do atrito).

A Fase 7 inicia com a paragem súbita da sonda quando se deteta a perda total do cordão (Figura 5.16c), verificando-se um pico em MY devido à instabilidade causada. De seguida, existe um reposicionamento da sonda numa região sobre o cordão. Nesta fase, em que a sonda está em movimento livre, verifica-se força e momento não nulos, algo que poderá ser devido a inércia gerada pelo movimento rápido do robô.

Já a Fase 8 corresponde à aproximação final segundo a direção longitudinal do cordão à posição alvo (posição inicial de inspeção), sendo natural a sua semelhança com as Fases 5 e 6, por serem movimentos análogos. A única diferença é que na Fase 8 o movimento é interrompido quando a sonda chega à posição alvo, não quando sai totalmente do cordão.

Por fim, a oscilação visível na Fase 9 tem a ver com a fase final de ajustamento segundo MY, cujo objetivo é garantir um bom contacto entre a sonda e o cordão. Como previamente referido, nesta fase utiliza-se uma combinação de força no sentido positivo de Z (*Tool*

Frame) e compliance segundo RY para garantir que ambos os apoios da sonda se encontram em contacto com o cordão.

Idealmente, a oscilação referida no parágrafo anterior tenderia para um valor nulo, mas devido à incerteza relativamente elevada ao controlo de força [47] e ao facto de se ter de encontrar um equilíbrio entre rapidez e qualidade do ajustamento, tal não se verifica na prática.

5.2.3 Desgaste do tejadilho durante o ajustamento ao cordão

Um aspeto a assinalar, visível na Figura 5.13, é a existência de marcas de desgaste provocadas pelo contacto deslizante entre a sonda e o tejadilho. Em relação a isso, aquilo que a observação permitiu concluir foi que, maioritariamente, tais marcas se devem à repetição sucessiva de testes semelhantes. Esta situação não corresponde ao que sucederia na realidade, em que o percurso de aproximação da sonda ao cordão seria apenas efetuado uma vez, pelo que esta situação acabou por não sofrer um estudo mais prolongado.

Contudo, em relação a este assunto, destaca-se o facto de a sonda utilizada já ter, previamente ao início do desenvolvimento da presente tese, algum desgaste. Esse desgaste foi ainda agravado pela quantidade bastante elevada de uso da sonda durante todo o projeto, além de alguns testes em que se utilizaram forças de contacto de magnitude mais elevada do que as que acabaram por figurar na versão final do sistema proposto.

Este desgaste levou a que, sobretudo nas zonas assinaladas pelas setas a vermelho na Figura 5.17, o material começasse a apresentar uma rugosidade mais elevada e, inclusive, algumas arestas ligeiramente afiadas, potenciando o dano provocado ao tejadilho.

Uma outra questão prende-se com as zonas assinaladas a amarelo na Figura 5.17, responsáveis pela maior parte do desgaste infligido ao tejadilho. Uma vez o corpo da sonda é todo produzido por SLS, com o mesmo material (poliamida), tal pode significar que é a geometria das zonas assinaladas a amarelo, e não o seu material, que faz com que as mesmas provoquem mais desgaste do que as zonas assinaladas a vermelho. Assim, poderá ser interessante explorar *designs* alternativos.

Além disso, uma vez que o programa teve de sofrer várias iterações até chegar à sua forma final, e tendo em conta que durante todo esse processo inúmeros testes foram necessários, torna-se um pouco difícil distinguir o desgaste provocado por fases iniciais do programa menos corretas e testes mal sucedidos, do desgaste eventualmente provocado por apenas uma passagem com a versão final do programa.

5.3 Fase de inspeção

A fase que se segue ao ajustamento da sonda ao cordão é a realização da inspeção propriamente dita. Para tal, como previamente referido, a sonda deve percorrer o cordão de forma tão estável quanto possível.

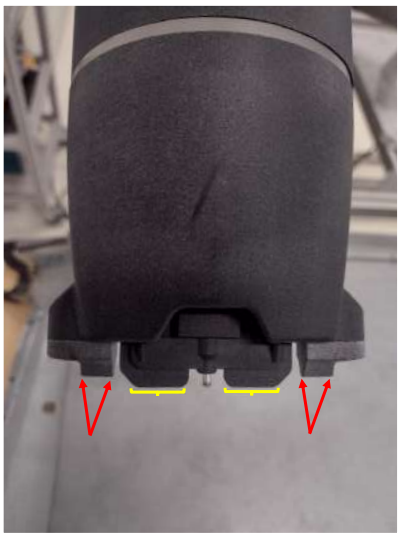


Figura 5.17: Zonas provocadoras de desgaste no tejadilho

Existem dois aspetos fundamentais que serão explorados neste subcapítulo: o controlo de força utilizado durante a fase de inspeção, procurando-se demonstrar a sua utilidade, e os resultados da inspeção por correntes induzidas propriamente dita.

5.3.1 Análise do controlo de força

A utilidade do controlo de força durante a fase de ajustamento inicial ao cordão é inquestionável, na medida em que constitui a base de praticamente todos os movimentos que ocorrem.

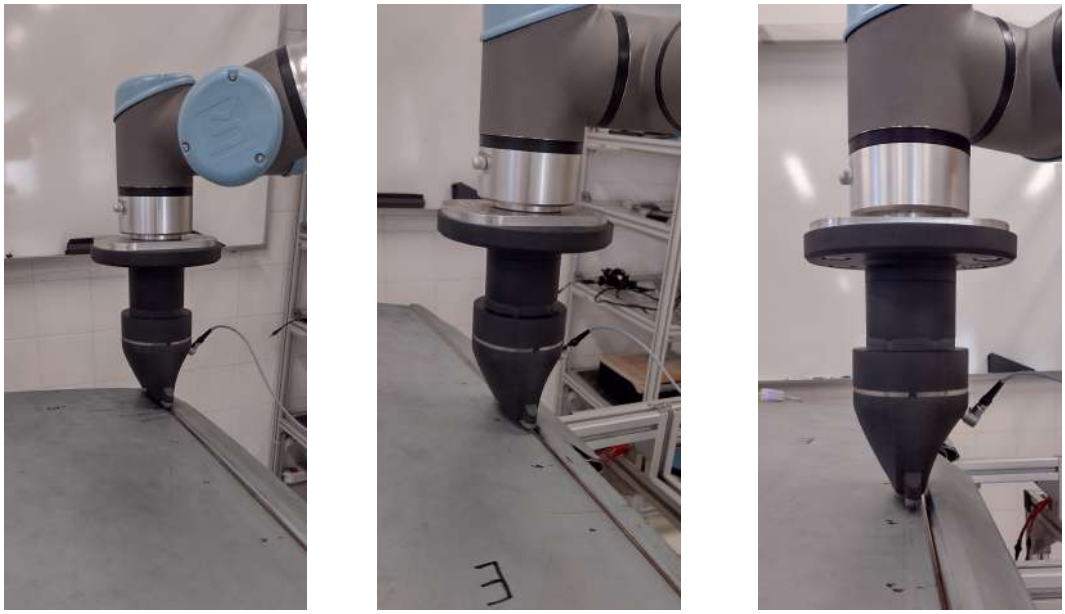
Contudo, na fase de inspeção a relevância do controlo de força poderá não ser tão evidente à primeira vista. Como tal, apresentar-se-ão de seguida alguns resultados que confirmam a necessidade da sua utilização, que, na versão final do programa incorpora controlo das forças segundo as direções Y e Z (*Tool Frame*).

Para cada tipo de controlo de força, os testes consistiram em executar a totalidade do sistema desenvolvido (ajustamento ao cordão seguido da fase em que é percorrido o caminho), realizando-se três ensaios a partir de cada uma das três posições iniciais definidas na Figura 5.13 e na Tabela 5.4.

5.3.1.1 Inspeção sem controlo de força segundo FY (*Tool Frame*)

As razões para a utilização de controlo de força em FY foram previamente referidas em 3.4, e têm a ver com a incerteza associada à definição da trajetória a partir do ponto inicial da inspeção.

De facto, os testes realizados apenas com controlo de força segundo Z corroboram esta necessidade, sendo reduzido o número de ocasiões em que a sonda percorreu todo o cordão sem perder o contacto, algo ilustrado pela sequência presente na Figura 5.18



(a) (b) (c)

Figura 5.18: Perda de contacto numa inspeção apenas com controlo de FZ

As perdas de contacto foram sempre da mesma natureza, para o lado esquerdo do cordão na Figura 5.18c. Uma explicação possível é que, ao contrário do lado direito, em que existe o ressalto, no lado esquerdo do cordão não existe uma barreira desse tipo.

5.3.1.2 Inspeção sem controlo de força segundo FZ (*Tool Frame*)

O controlo de força segundo a direção Z (*Tool Frame*) tem a função de garantir um contacto constante com o cordão, contribuindo para mitigar variações de *lift-off*, que causam flutuações indesejadas e inesperadas nos valores retirados de ensaios de correntes induzidas.

De facto, isto é especialmente evidente na fase final da inspeção, em que a curvatura do cordão é mais acentuada, sendo por isso mais desafiante manter um contacto tão constante quanto possível, algo que é evidente na comparação entre os gráficos presentes nas Figuras 5.19 e 5.20.

De notar que estes gráficos foram obtidos utilizando os mesmos parâmetros de frequência, ganho e rotação de fase, o que permite concluir que o controlo de força segundo Z provoca uma melhoria no sinal obtido, sobretudo em termos da amplitude máxima verificada.

Além disso, também parece existir uma redução do ruído. Uma possível explicação para tal pode ser o facto de que, se não houver controlo segundo Z, então a força de contacto poderá tornar-se demasiado elevada, caso a trajetória definida inicialmente não seja a mais adequada. Tal poderia causar alguma trepidação durante o movimento da sonda, o que se poderia traduzir por exemplo nas oscilações observadas na Figura 5.19, no intervalo de tempo entre os 23, 5 s e os 24 s.

Figura 5.19: Fase final de uma inspeção por correntes induzidas de uma parte da peça sem defeito utilizando apenas controlo de força segundo Y (*Tool Frame*)

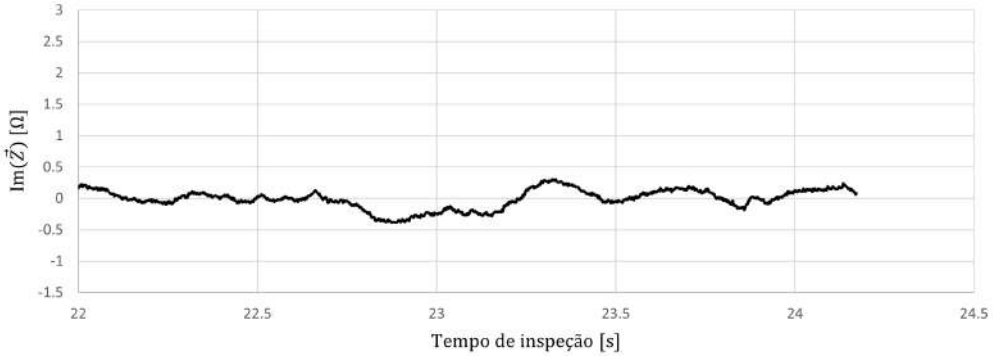


Figura 5.20: Fase final de uma inspeção por correntes induzidas de uma parte da peça sem defeito utilizando controlo de força segundo Y e Z (*Tool Frame*)

5.3.1.3 Inspeção sem controlo de força

Tendo em conta os resultados apresentados em 5.3.1.2 e 5.3.1.1, o expectável é que uma inspeção sem qualquer controlo de força combine os problemas detetados em cada um dos subcapítulos referidos.

De facto, foi isso que se observou, com perda do cordão por parte da sonda e um sinal de inspeção por correntes induzidas mais ruidoso. Uma vez que os resultados (figuras e gráficos) são análogos ao que foi apresentado em 5.3.1.2 e 5.3.1.1, optou-se por omiti-los.

5.3.1.4 Inspeção com controlo de força segundo FY e FZ (*Tool Frame*)

Os resultados apresentados em 5.3.1.1 a 5.3.1.3 confirmam que a utilização de controlo de força é fundamental para a realização de uma inspeção com sucesso.

Nas Figuras 5.21 e 5.22, é possível observar os dados obtidos para as forças segundo Y e Z, respetivamente, ao longo da fase de inspeção, saltando imediatamente à vista o facto de apresentarem uma oscilação em torno do valor definido.

No caso de FY, o programa está definido de forma a que o robô exerça uma força de 3 N, daí que na Figura 5.21 os valores da força oscilem em torno de -3 N, por estes serem

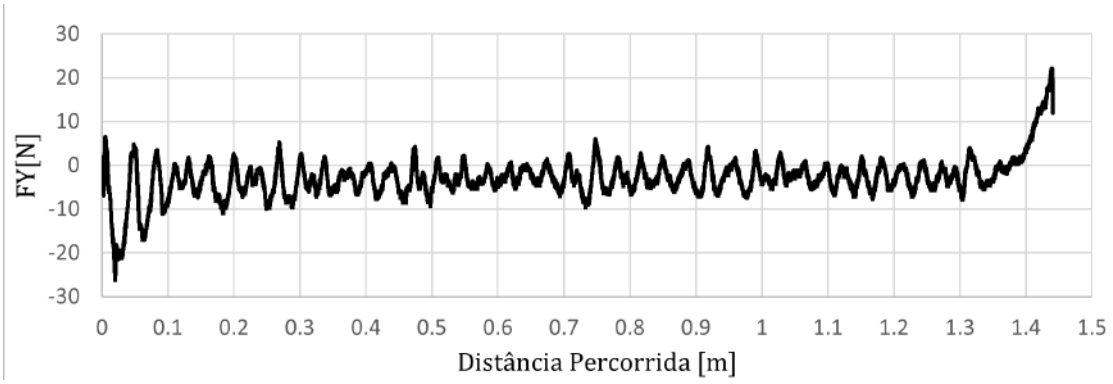


Figura 5.21: Gráfico da força segundo Y em função da distância percorrida (*Tool Frame*)

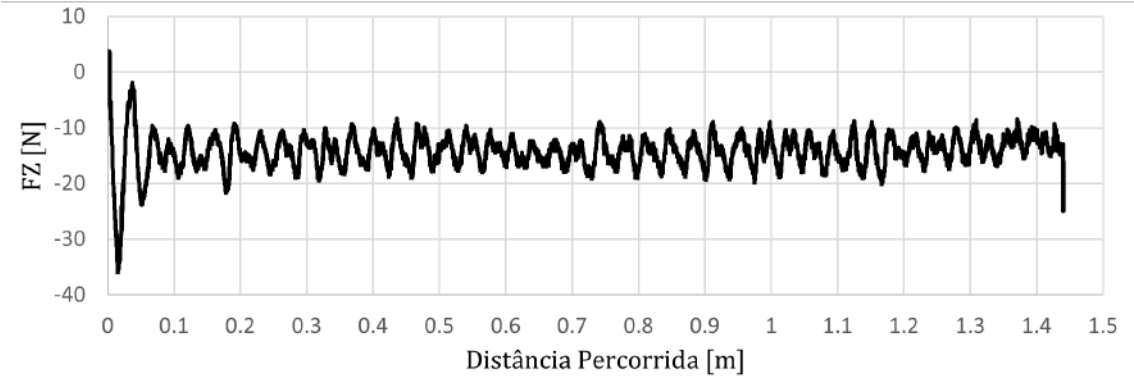


Figura 5.22: Gráfico da força segundo Z em função da distância percorrida (*Tool Frame*)

relativos à reação sofrida pela sonda. Na Figura 5.22, isto é mais evidente, oscilando os valores da força em torno dos -15 N, o que é coerente com o facto de se ter definido, através de controlo de força, que deve ser mantida uma força de contacto entre a sonda e o cordão de 15 N (sentido positivo de Z, *Tool Frame*).

Algo que também está presente em ambas as figuras mas que é especialmente evidente na Figura 5.22 é o facto de a amplitude de oscilação ser, durante a maior parte do tempo, de cerca de 10 N. Ora, a incerteza fornecida pelo fabricante para o sensor de força incorporado no robô é de ± 10 N [47]. Desse modo, à exceção das fases inicial e final do movimento, pode considerar-se que, durante a inspeção o controlo de força agiu dentro do expectável.

Na fase inicial, existe para ambas as forças uma fase inicial de acomodação, algo perfeitamente natural, uma vez que a aceleração durante o arranque inicial provoca instabilidade.

Já no último instante da fase final, nota-se nas Figuras 5.21 e 5.22 uma alteração abrupta nas forças medidas, algo que também é coerente com o facto de existir uma paragem súbita do movimento da sonda em simultâneo com o fim do controlo de força.

Outro aspeto a assinalar é o aumento progressivo da força medida segundo Y, desde cerca da abcissa $1,35$ m da Figura 5.21 até ao último instante, em que ocorre a queda abrupta descrita no parágrafo anterior.

O facto de este fenómeno envolver uma força positiva segundo Y, contrariamente

ao que ocorre durante as condições normais de operação, é um pouco contraintuitivo, sobretudo por indicar que a sonda perdeu o contacto com o ressalto do tejadilho. Uma explicação pode ser que, nesta fase final, a geometria do cordão é mais complexa, ou seja, o desvio entre a trajetória definida e a ideal pode ser mais acentuado. Ora, a velocidade de correção segundo Y foi definida como sendo relativamente reduzida (2 mm/s), de forma a minimizar oscilações excessivas da sonda segundo Y ao longo do cordão. Assim, é possível que, nesta secção final, o programa não seja capaz de reagir rápido o suficiente e a sonda, em vez de encostar ao ressalto do tejadilho, acabe por encostar ao lado do oposto do cordão, não o perdendo apenas por uma combinação entre a curvatura do cordão e a a aplicação de força segundo o sentido positivo de Z.

Tendo isso em conta, talvez pudesse ter sido explorada a possibilidade de um controlo de força um pouco mais dinâmico, com parâmetros mais personalizados para cada zona do cordão. Contudo, como será possível ver nas Figuras 5.25 a 5.27, apesar de ser de facto possível perceber que as flutuações nos ensaios de correntes induzidas são um pouco superiores no troço final do cordão, a diferença para o troço intermédio não é significativa. Assim, pode afirmar-se que o desvio da força medida segundo Y verificado na Figura 5.21 não têm efeitos significativos nas medições de correntes induzidas realizadas durante a inspeção, o que acaba por ser o mais importante.

5.3.2 Ensaios de inspeção por correntes induzidas

Após a validação da fase de ajustamento ao cordão e da melhoria provocada pelo controlo de força à fase de inspeção, apresentar-se-ão de seguida os resultados relativos à inspeção por correntes induzidas propriamente dita, no que concerne à sua capacidade de detetar os defeitos artificiais introduzidos no cordão.

Na Figura 5.23, é visível a localização de cada um dos defeitos referidos no parágrafo anterior, além de ser dada uma indicação sobre se o defeito é superficial ou subsuperficial.

De notar que os defeitos foram criados através da utilização de uma broca manual de $0,9$ mm. Isso significa que a sua profundidade não é particularmente uniforme, mas estimou-se que em todos os casos essa profundidade seja entre $0,5$ mm e $0,8$ mm.

Além disso, o facto de ter sido utilizada uma broca manual acabou por impossibilitar que os defeitos ficassem perfeitamente centrados no cordão, algo que acaba por não ser grave, na medida em que dessa forma existe um pouco mais de correspondência com a realidade. Na Figura 5.24, encontra-se ilustrado este aspeto.

Relativamente à sonda utilizada, a mesma foi desenvolvida previamente à presente dissertação [4]. Consiste numa sonda de tipo absoluto, com uma bobina de teste constituída por 200 enrolamentos de fio de cobre com 40 μm. Possui um núcleo de ferrite 78 com diâmetro de $0,75$ mm e um *shield* com diâmetro externo de 2 mm e $1,65$ mm. Para tornar a sonda mais sensível, a mesma apresenta ainda uma bobina de referência.

No que concerne aos parâmetros de inspeção, que foram mantidos constantes ao longo dos ensaios que serão subsequentemente apresentados, os mesmos estão presentes na

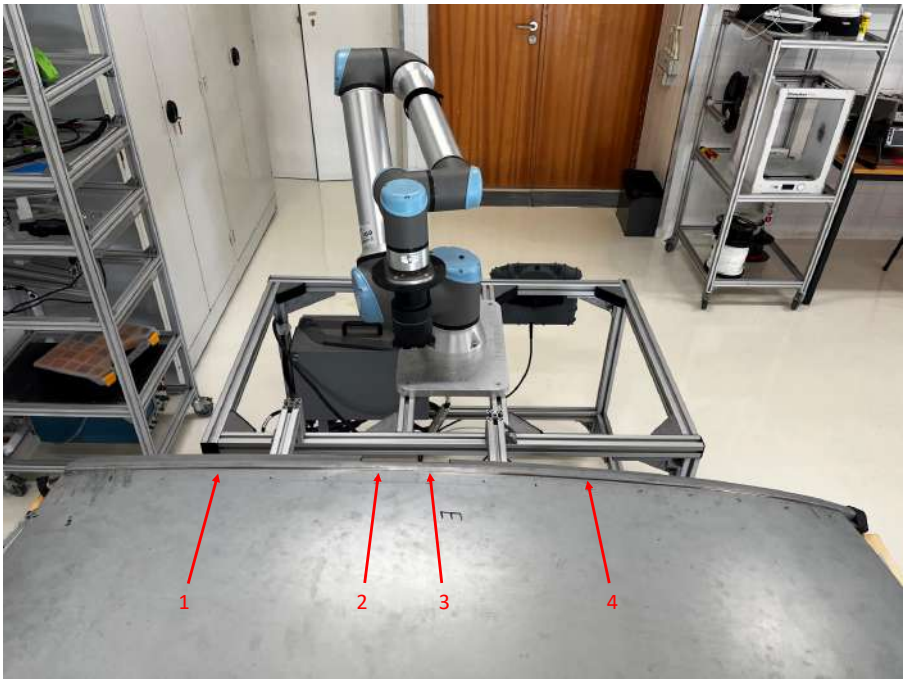


Figura 5.23: Localização dos defeitos
1: Superficial 2: Subsuperficial 3: Superficial 4: Superficial

(a) (b)

Figura 5.24: Comparação da centralidade dos defeitos: (a) Defeito 1; (b) Defeito 4

Tabela 5.11.

Tabela 5.11: Parâmetros de inspeção

Frequência [kHz]	Ganho [dB]	Velocidade de inspeção [mm/s]
250	90	50

Em primeiro lugar, a frequência foi definida de forma empírica, tendo se chegado à conclusão de que era esta a frequência de excitação que levava a uma melhor relação sinal-ruído, pelo menos à vista desarmada, e que permitia resultados mais consistentes. Esta conclusão foi de certa forma corroborada por [4], em que se realiza uma análise a uma junta brasada idêntica por correntes induzidas.

Já o ganho foi meramente definido de forma a obter resultados bem visíveis, sem ocorrer saturação.

Por fim, a velocidade de inspeção também foi escolhida de forma empírica, testando-se inspeções com velocidades de 30 mm/s, 50 mm/s e 100 mm/s. Por um lado, com uma velocidade de 30 mm/s os resultados eram menos consistentes. Por outro, o sinal tornava-se mais ruidoso a 100 mm/s. Assim, optou-se por 50 mm/s.

5.3.2.1 Ensaios apenas com desalinhamento da sonda

Em primeiro lugar, realizou-se um conjunto de testes nos quais não foi realizada qualquer alteração à posição do tejadilho, o que significa que é apenas o desalinhamento inicial da sonda que permite testar as capacidades do programa de lidar com a incerteza no posicionamento do tejadilho.

À semelhança dos testes de ajustamento ao cordão, realizaram-se 5 ensaios para cada uma das posições iniciais apresentadas na Tabela 5.4 e Figura 5.13. Estes ensaios envolveram a totalidade do programa desenvolvido ao longo da tese, desde o ajustamento inicial ao cordão e encontro da posição inicial ideal, até ao percurso de inspeção propriamente dito. Assim, simulam o que seria realizado caso o programa fosse utilizado em condições reais.

Para efeitos de legibilidade e clareza de apresentação dos resultados, considerou-se que os mesmos deveriam ser divididos em três gráficos distintos: o primeiro correspondente aos primeiros $0,5$ m de inspeção, o segundo correspondente aos segundos $0,5$ m, e o terceiro correspondente ao restante percurso. Desse modo, os resultados para a posição de teste 1 (ver Figura 5.13 e Tabela 5.4) estão presentes nos gráficos das Figuras 5.25, 5.26 e 5.27.

Antes de mais nada convém apontar que a razão pela qual estes gráficos têm um aspeto distinto do que foi apresentado na 5.20 prende-se, sobretudo, por nesta se apresentar um período de inspeção bastante mais reduzido. De facto, os cerca de $2,2$ s de inspeção apresentados na Figura 5.20 correspondem aproximadamente ao intervalo entre $1,31$ m e $1,44$ m de distância percorrida no gráfico apresentado na Figura 5.27. Desse modo, ao

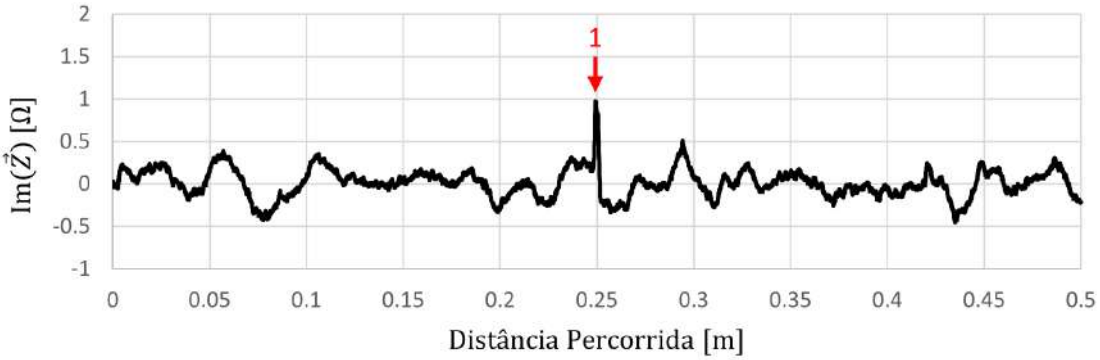


Figura 5.25: Primeiro troço de inspeção à frequência de 250 kHz (presença do defeito superficial 1). Resultados obtidos partindo da posição inicial 1, no ensaio 1

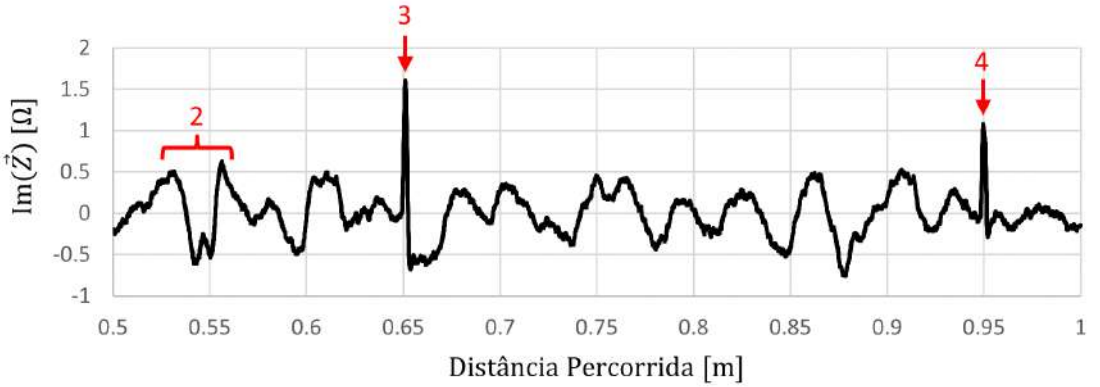


Figura 5.26: Segundo troço de inspeção à frequência de 250 kHz (presença do defeito subsuperficial 2 e dos defeitos superficiais 3 e 4). Resultados obtidos partindo da posição inicial 1, no ensaio 1

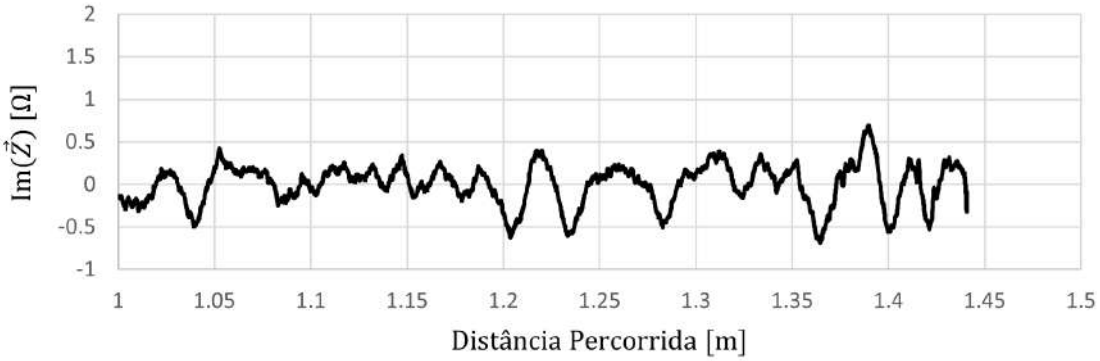


Figura 5.27: Terceiro troço de inspeção à frequência de 250 kHz. Resultados obtidos partindo da posição inicial 1, no ensaio 1

estar mais "esticado", o gráfico da Figura 5.20 aparenta ter uma ondulação muito mais suave.

Nos gráficos apresentados, em primeiro lugar, é fácil identificar os três picos correspondentes às perturbações provocadas pelos defeitos superficiais: na abcissa 0, 25 m da Figura 5.25 e nas abcissas 0, 65 m e 0, 95 m da Figura 5.26.

Contudo, o defeito subsuperficial (defeito 2, Figura 5.26) é indistinguível do ruído, pelo que não se pode considerar como detetado. Um modo de facilitar a sua deteção poderia eventualmente ter sido utilizar uma frequência mais reduzida, na medida em que assim existe uma maior profundidade de penetração das correntes induzidas. No entanto, os testes realizados com frequências no intervalo $70 - 100$ kHz revelaram-se inutilizáveis, na medida em que nenhum dos defeitos era detetável. Tal pode dever-se por exemplo ao facto de essa maior profundidade de penetração referida poder implicar que a raiz do cordão é atingida. Desse modo, as irregularidades que aí existam são detetadas e possuem uma influência nas medições que leva ao aumento do ruído.

Outro dos aspetos que se destacam é o facto de existir uma ondulação considerável, que se deve ao facto de o trajeto percorrido pela sonda não ser perfeitamente correspondente ao trajeto definido pelo cordão brasado. Tal é sobretudo visível a partir da abcissa 0, 6 m, da Figura 5.26, algo expectável, na medida em que é por volta desta localização que o cordão começa a sofrer uma alteração na geometria, tornando-se menos reto. Efetivamente, é clara a diferença nesse sentido entre a Figura 5.25 e as Figuras 5.26 e 5.27.

Outro aspeto a salientar é que, apesar de as flutações visíveis serem consideráveis, as mesmas são coerentes de ensaio para ensaio, mesmo utilizando posições iniciais diferentes, como é percetível nas Figuras 5.28, 5.29 e 5.30. De facto, os picos e vales presentes nestas figuras encontram-se, na sua esmagadora maioria, em posições análogas.

O facto de isto acontecer mesmo no terceiro troço de inspeção, em que a curvatura do cordão é bastante exigente, parece indicar que o ruído existente na inspeção não é aleatório, mas sim sistemático.

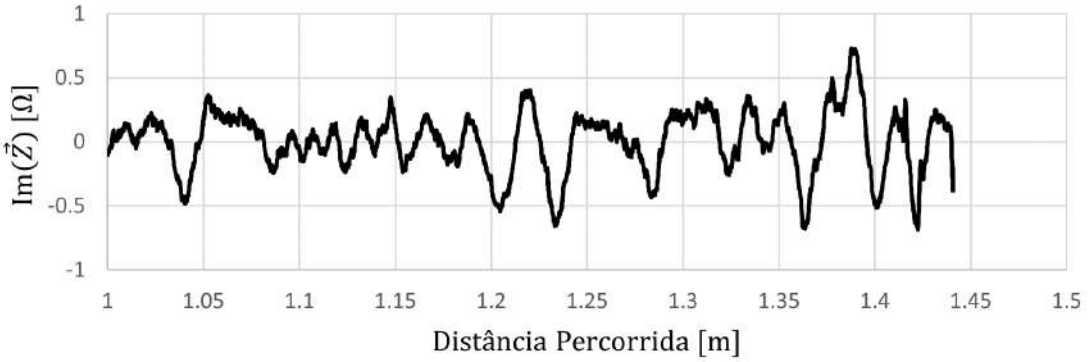


Figura 5.28: Terceiro troço de inspeção à frequência de 250 kHz. Resultados obtidos partindo da posição inicial 1, no ensaio 3

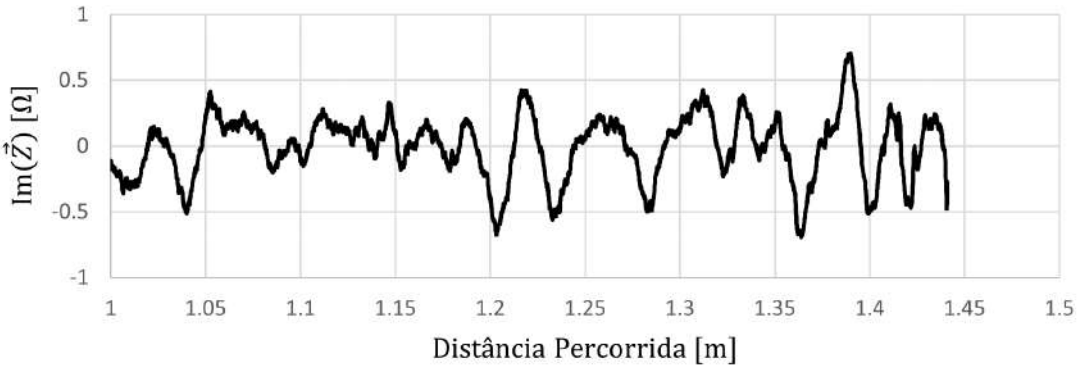


Figura 5.29: Terceiro troço de inspeção à frequência de 250 kHz. Resultados obtidos partindo da posição inicial 2, no ensaio 2

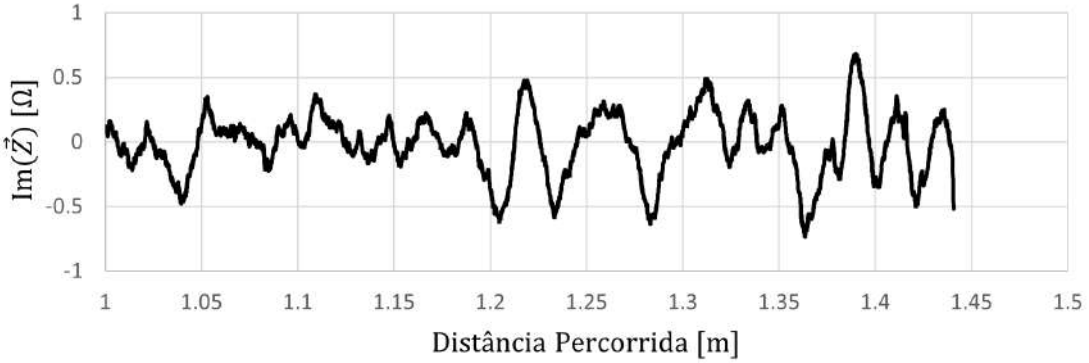


Figura 5.30: Terceiro troço de inspeção à frequência de 250 kHz. Resultados obtidos partindo da posição inicial 3, no ensaio 5

5.3.2.2 Ensaios com desalinhamento da sonda e do tejadilho

Como referido no subsubcapítulo 5.2.2, além de testar o programa com desalinhamentos iniciais da sonda, também se decidiu experimentar o mesmo provocando manualmente desalinhamentos na própria posição do tejadilho.

Em virtude do processo utilizado para provocar esses desalinhamentos, que envolveu apenas pegar no tejadilho e deslocá-lo manualmente, não é possível quantificar rigorosamente os desalinhamentos provocados, mas procurou-se ao máximo que os mesmos fossem de cerca de 5 mm no ponto de início da inspeção e de 50 mm no ponto final, segundo a direção X (*Base Frame*). De notar que o processo foi repetido de forma a serem realizados ensaios para desalinhamentos segundo ambos os sentidos da direção X. Uma representação dos desalinhamentos provocados encontra-se na Figura 5.31.

Nas Figuras 5.32 e 5.33 encontram-se os resultados para os ensaios efetuados.

À semelhança do que se observou nos ensaios anteriores, estão perfeitamente destacadas as posições dos três defeitos superficiais, nas abcissas $0,25$ m, $0,65$ m e $0,95$ m. Contudo, novamente, o defeito subsuperficial não é distinguível do ruído.

Mais uma vez é visível nestes resultados que, de ensaio para ensaio, e mesmo agora que se provocou um desalinhamento no tejadilho, os gráficos continuam a ter uma aparência

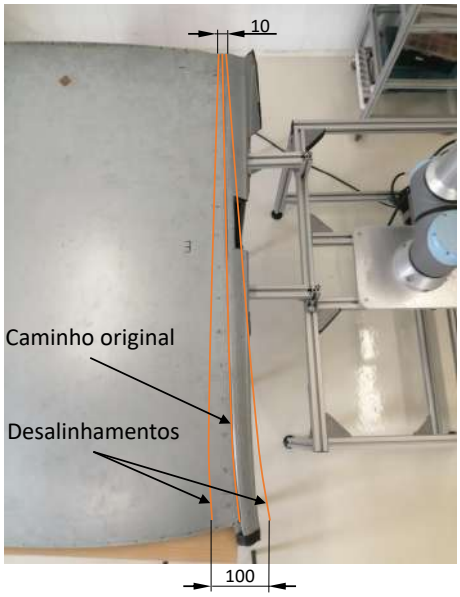


Figura 5.31: Caminhos previstos após desalinhamentos do tejadilho (comprimentos em mm)

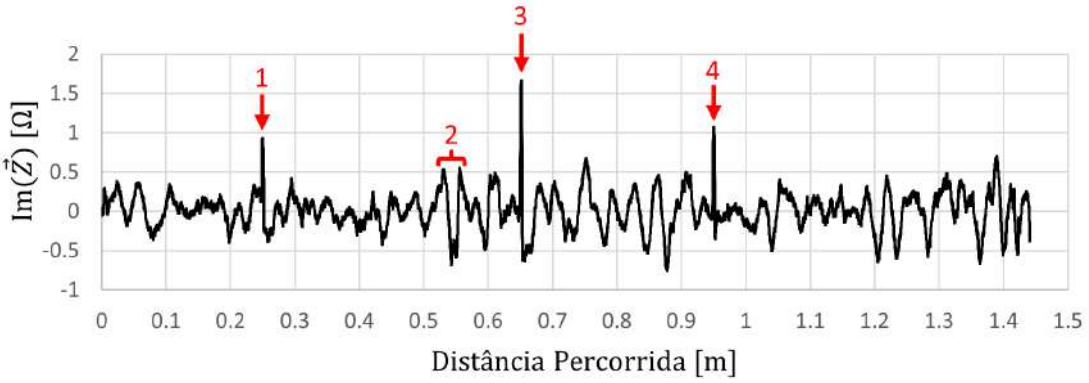


Figura 5.32: Inspeção à frequência de 250 kHz: desvio no sentido positivo de X (*Base Frame*)

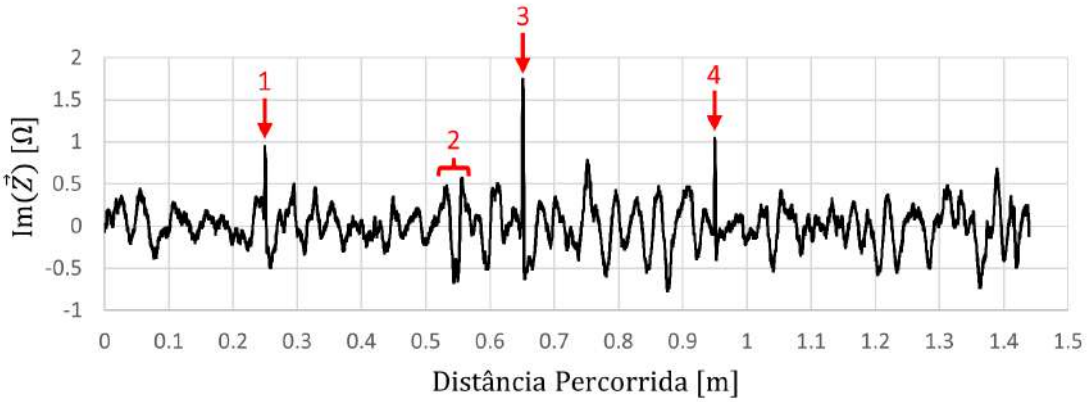


Figura 5.33: Inspeção à frequência de 250 kHz: desvio no sentido negativo de X (*Base Frame*)

bastante coerente entre si, à semelhança do que foi referido na secção anterior (5.3.2.1).

5.3.2.3 Comentários finais

Uma das explicações que pode ser avançada para a questão de os gráficos terem uma aparência bastante coerência entre si é o facto de o programa de inspeção ter envolvido uma etapa de mapeamento do cordão em que, através da movimentação manual do robô, se obtiveram uma série de pontos na superfície do cordão que serviram para definir o trajeto. Caso o ajustamento ao cordão, e consequente definição da posição inicial de inspeção, seja realizado de forma bastante consistente, algo que os testes realizados em 5.2.2 parecem corroborar, então para o mesmo desalinhamento inicial espera-se a definição de uma trajetória de inspeção muito semelhante. Como tal, espera-se que o erro na definição da trajetória seja mantido constante de ensaio para ensaio e, consequentemente, as flutações observadas nos ensaios de correntes induzidas sejam provocadas sempre pelas mesmas correções de controlo de força, daí que os gráficos apresentados sejam análogos.

Um dos modos de mitigar este problema seria por exemplo realizar o mapeamento do percurso ideal com um *end-effector* diferente, como por exemplo uma esfera de diâmetro ligeiramente inferior à largura do cordão. Tal permitiria maior manobrabilidade do que a sonda utilizada, por exemplo pelo facto de esta possuir dois apoios e necessitar de uma dimensão relativamente elevada para alojar os componentes associados a uma inspeção por correntes induzidas. Em teoria, desse modo seria possível mapear o cordão de modo mais fiel, e, consequentemente, obter melhores resultados.

Algo também relevante de salientar é que a metodologia desenvolvida de criar um trajeto ideal e de estimar o trajeto de inspeção através de uma combinação do trajeto ideal e da posição inicial de inspeção assenta no pressuposto de que os cordões analisados são todos bastante semelhantes. Poderá ser necessário realizar testes com diferentes tejadilhos, para confirmar que a metodologia, além de se adaptar a diferentes orientações de cordão, se consegue adaptar a diferentes cordões.

Por fim, refere-se que, relativamente à inspeção por correntes induzidas, é possível então concluir que a metodologia utilizada possui potencial para a deteção de defeitos superficiais, mas que, no seu estado atual, a deteção de defeitos subsuperficiais não se encontra validada.

6

Conclusão e Trabalhos Futuros

Nesta dissertação, foi apresentada uma abordagem à inspeção de juntas brasadas em carroçarias de automóveis com dois objetivos: em primeiro lugar, o de levar, através de um robô, uma sonda até a uma posição alvo no cordão brasado, partindo de uma posição desconhecida sobre uma superfície irregular (tejadilho). Em segundo lugar, o objetivo de, partindo da posição alvo referida, percorrer um caminho definido por um ressalto na superfície irregular. Este caminho corresponde ao cordão brasado do tejadilho.

A abordagem referida procura fazer face ao facto de estas inspeções ocorrerem num ambiente parcialmente estruturado, em que o ponto inicial de inspeção (referido no parágrafo anterior como posição alvo) e o caminho de inspeção variam de carroçaria para carroçaria.

Assim, propôs-se que os movimentos do robô sejam controlados em força e posição, de forma a que o par composto pelo robô e pela sonda possua uma maior sensibilidade relativamente à superfície, por exemplo no que concerne à deteção de colisões ou de manutenção do contacto com a mesma.

Para se poder analisar e validar a metodologia proposta, projetou-se um programa em python, de forma a estabelecer a comunicação entre os vários elementos envolvidos, possibilitando a recolha de dados essenciais.

A validação da metodologia teve por base um conjunto de testes, tendo sido projetada uma montagem experimental em que se movimentou através de um robô (Universal Robots UR10e) uma sonda de correntes induzidas sobre um tejadilho de um automóvel.

Relativamente ao primeiro objetivo, de ajustar a sonda ao cordão a partir de uma posição genérica sobre o tejadilho, realizaram-se uma série de testes para otimizar os parâmetros de controlo de força e posição, sendo de seguida apresentados que se consideram mais relevantes.

Na Fase 1, de reorientação da sonda após o primeiro contacto com o tejadilho, estabeleceu-se no modelo de controlo de força uma força de intensidade 45 N no sentido positivo de Z, bem como *compliance* segundo RY, com uma velocidade de correção máxima de RY de $0,07$ rad/s (*Tool Frame*, ver Figura 5.3b). A duração desta fase de correção foi definida como $3,5$ s.

Já na Fase 2, de aproximação da sonda ao cordão, definiu-se no modelo de controlo de força uma força de magnitude 20 N no sentido positivo de Z bem como *compliance* segundo RY, com uma velocidade de correção máxima de RY de $0,02$ rad/s (*Tool Frame*).

Por fim, na Fase 3, em que se realiza um ajustamento de forma a alinhar a sonda no cordão, definiu-se uma sequência de ajustamentos em que se alterna correção RY+RZ com correção apenas em RY (*Tool Frame*). Estabeleceu-se que a etapa RY+RZ devia decorrer até uma de duas condições serem cumpridas: passarem $0,4$ s ou a rotação de RZ atingir o valor de $1,5°$. Já a etapa de correção RY foi definida com uma duração de 1 s. Considera-se que o ajustamento segundo RZ está concluído quando, numa determinada etapa de correção RY+RZ, a rotação efetuada segundo RZ é inferior a $0,5°$.

Terminada a otimização de parâmetros, concluiu-se que através do controlo de força é possível detetar o primeiro contacto da sonda com o tejadilho e corrigir, com consistência, desvios de até 8° segundo RY (*Tool Frame*). Por outro lado, os testes realizados permitiram estabelecer que desvios até 5° segundo RZ (*Tool Frame*) são corrigidos pelo sistema projetado.

Em termos de desvios posicionais, testaram-se desvios de até 100 mm segundo a direção X e de até 50 mm segundo Y (ver referencial da Figura 5.13).

Por fim, os testes realizados permitiram balizar o tempo total de ajustamento entre 18 s e 26 s, dependendo da severidade do desvio que é necessário corrigir.

Passando para a fase de inspeção (em que é percorrido o caminho ao longo do cordão), foi necessário estudar duas questões principais. Por um lado, se a metodologia proposta permite percorrer o cordão sem existirem perdas de contacto com o mesmo, e, se sim, qual o modelo de controlo de força necessário para tal acontecer. Por outro lado, se, ao percorrer o cordão, a sonda é capaz de detetar os defeitos artificialmente produzidos no mesmo.

Em primeiro lugar, os testes realizados permitiram concluir que a utilização de controlo de força segundo as direções Y e Z (*Tool Frame*) é necessária para que seja percorrido o cordão sem existirem perdas de contacto.

Consequentemente, isso permite concluir que a metodologia proposta de criar um mapeamento relativo do cordão e de aliar isso a controlo de força permite, de facto, que o caminho pretendido seja percorrido sem perdas de contacto.

Contudo, apesar de, à vista desarmada, a sonda percorrer o cordão da forma pretendida, sem oscilações ou desvios da trajetória, os dados provenientes da inspeção por correntes induzidas apresentam ruído significativo, não sendo possível discernir o defeito subsuperficial criado. Apesar disso, os três defeitos superficiais foram consistentemente detetados.

Assim, é relevante assinalar que as inspeções por correntes induzidas estão sempre sujeitas a algum ruído, dada a sua sensibilidade elevada. Além disso, como foi referido previamente, o facto de os sinais obtidos nas diversas inspeções serem sempre bastante semelhantes, mesmo quando a sonda partiu de posições iniciais diferentes para a fase de ajustamento ou quando se criaram desalinhamentos no tejadilho, é encorajador.

De facto, tal parece significar que o sistema de inspeção proposto apresenta consistência e que consegue acomodar desalinhamentos no tejadilho ou na sonda, como era pretendido, ainda que não permita realizar inspeções capazes de detetar defeitos subsuperficiais.

Por fim, em termos de trabalhos futuros, algumas observações durante os testes de ajustamento ao cordão levam a crer que possam ser corrigidos desvios mais elevados do que os 5° segundo RZ testados. Contudo, para estabelecer este tipo de conclusões devem ser realizados testes mais extensivos com desvios superiores.

Por outro lado, propõe-se o estudo de modos alternativos de mapear o cordão, desde a utilização de um *end-effector* diferente, mais adequado para essa fase, até à utilização de sistemas de visão.

Além disso, poderá ser interessante testar esta metodologia em várias juntas brasadas, para confirmar se as correções previstas pela mesma são capazes de lidar com o facto de, naturalmente, diferentes juntas apresentarem superfícies distintas.

Bibliografia

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (ver p. iii).

[2] C. Mineo et al. «Fast ultrasonic phased array inspection of complex geometries delivered through robotic manipulators and high speed data acquisition instrumentation». Em: *2016 IEEE International Ultrasonics Symposium (IUS)*. 2016. DOI: 10.1109/ULTSYM.2016.7728746 (ver p. 1).

[3] C. Mineo, D. Cerniglia e A. Poole. «Autonomous Robotic Sensing for Simultaneous Geometric and Volumetric Inspection of Free-Form Parts». Em: *Journal of Intelligent Robotic Systems* 105 (2022). DOI: 10.1007/s10846-022-01673-6 (ver pp. 1, 5).

[4] M. Machado et al. «New directions for inline inspection of automobile laser welds using non-destructive testing». Em: *The International Journal of Advanced Manufacturing Technology* 118 (2021). DOI: 10.1007/s00170-021-08007-0 (ver pp. 1, 4, 5, 55, 57).

[5] M. Machado et al. «Multisensor Inspection of Laser-Brazed Joints in the Automotive Industry». Em: *Sensors* 21 (2021). DOI: 10.3390/s21217335 (ver pp. 3, 5).

[6] Y. Zhao et al. «A new array eddy current testing probe for inspection of small-diameter tubes in Tokamak fusion devices». Em: *Fusion Engineering and Design* 157 (2020). URL: https://www.sciencedirect.com/science/article/pii/S0920379620301757 (ver pp. 3, 4).

[7] J. Lanzagorta et al. «New inspection approaches for railway based on Eddy Current». Em: ECNDT 2018, Gotenburgo. URL: https://www.ndt.net/article/ecndt2018/papers/ecndt-0091-2018.pdf (ver p. 3).

[8] T. Santos. *Processos Avançados de Fabrico e Ensaios não Destrutivos - Correntes Induzidas (CI)*. Powerpoint (ver p. 3).

[9] R. Smid, A. Docekal e M. Kreidl. «Automated classification of eddy current signatures during manual inspection». Em: *NDT E International* 38.6 (2005). DOI: https://doi.org/10.1016/j.ndteint.2004.12.004 (ver p. 4).

[10] E. Foster et al. «Automated Real-Time Eddy Current Array Inspection of Nuclear Assets». Em: *Sensors* 820 (2022). URL: https://www.mdpi.com/1424-8220/22/16/6036 (ver p. 4).

[11] M. Morozov et al. «A computational technique for automated recognition of subsurface cracks in aeronautical riveted structures». Em: *16th World Conference on NDT (WCNDT 2004)*. 2004. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ce8e846153176e7bf25d1187218c890b16f0673e (ver p. 4).

[12] T. Reyno et al. «Surface Profiling and Core Evaluation of Aluminum Honeycomb Sandwich Aircraft Panels Using Multi-Frequency Eddy Current Testing». Em: *Sensors* 17.9 (2017). DOI: 10.3390/s17092114 (ver p. 4).

[13] «Application of eddy currents induced by permanent magnets for pipeline inspection». Em: *NDT E International* 40.1 (2007). DOI: https://doi.org/10.1016/j.ndteint.2006.07.002 (ver p. 4).

[14] Eddyfi. *Spyne Array Probe*. URL: https://eddyfi.com/en/product/spyne-array-pipeline-crack-assessment (acedido em 2023-02-17) (ver p. 4).

[15] G. Afonso, J. N. Pires e N. Estrela. «Force control experiments for industrial applications: a test case using an industrial deburring example». Em: *Assembly Automation* 27 (2 2007). DOI: 10.1108/01445150710733414 (ver pp. 5, 6).

[16] C. Mineo et al. «Flexible integration of robotics, ultrasonics and metrology for the inspection of aerospace components». Em: *43rd Review of Progress in Quantitative Nondestructive Evaluation*. Vol. 1806. 2017. DOI: 10.1063/1.4974567 (ver p. 5).

[17] O. Patrouix, S. Bottecchia e J. Canou. «Improving Robotized Non Destructive Testing for Large Parts with Local Surface Approximation and Force Control Scheme». Em: *The 19th International Conference on Composite Materials*. 2013. URL: https://hal.science/hal-00912656 (ver pp. 5, 6).

[18] B. Siciliano e L. Villani. *Robot Force Control*. Springer, 1999. DOI: 10.1007/978-1-4615-4431-9 (ver pp. 6, 7).

[19] K. Lynch e F. Park. *Modern Robotics: Mechanics, Planning, and Control*. English (US). Cambridge Univeristy Press, 2017. ISBN: 978-1107156302 (ver pp. 6, 7, 34).

[20] T. Yoshikawa. «Force control of robot manipulators». Em: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*. 2000. DOI: 10.1109/ROBOT.2000.844062 (ver p. 6).

[21] M. E. de Mendonça. «A Position-based Approach for Force/ Moment Control of an Industrial Manipulator». Tese de mestrado. Instituto Superior Técnico, 2019 (ver pp. 7, 8).

[22] D. Leite. «Real-Time Hybrid Force/Position Control with a KUKA Industrial Robot». Tese de mestrado. Instituto Superior Técnico, 2020 (ver pp. 7–11).

[23] S. Chiaverini e L. Sciavicco. «The parallel approach to force/position control of robotic manipulators». Em: *IEEE Transactions on Robotics and Automation* 9.4 (1993). DOI: 10.1109/70.246048 (ver pp. 7, 8).

[24] M. H. Raibert e J. J. Craig. «Hybrid Position/Force Control of Manipulators». Em: *Journal of Dynamic Systems, Measurement, and Control* 103.2 (1981). DOI: 10.1115/1.3139652 (ver p. 8).

[25] A. Winkler e J. Suchý. «Position feedback in force control of industrial manipulators - An experimental comparison with basic algorithms». Em: *2012 IEEE International Symposium on Robotic and Sensors Environments Proceedings*. 2012. DOI: 10.1109/ROSE.2012.6402604 (ver pp. 8, 9).

[26] M. Bélanger-Barrette. *Robot Force Torque Sensor - How does it work?* 2015. URL: https://blog.robotiq.com/bid/72444/Robot-Force-Torque-Sensor-How-does-it-work (acedido em 2023-02-15) (ver p. 9).

[27] Mauser. *Sensor de pressão redondo Ø18mm - Force Sensitive Resistor - FSR400*. URL: https://mauser.pt/catalog/product_info.php?products_id=096-6327&gclid=Cj0KCQiAorKfBhC0ARIsAHDzslsuS6GShjScSpW7vZoC6qHEhIZRjDOs6OlwXvWLi4BKNzid45tJIxcaAt7REALw_wcBk (acedido em 2023-02-15) (ver p. 10).

[28] Kistler. *Strain gauge force sensors*. URL: https://www.kistler.com/INT/en/strain-gauge-force-sensors/C00000157 (acedido em 2023-02-15) (ver p. 10).

[29] ArduinoPortugal. *Como usar o Sensor Piezo no Arduino*. 2017. URL: https://www.arduinoportugal.pt/usando-buzzer-arduino-sensor-piezo-eletrico/ (acedido em 2023-02-15) (ver p. 10).

[30] Kuka. *KUKA.ForceTorqueControl*. URL: https://www.kuka.com/pt-pt/produtos-servi%C3%A7os/sistemas-de-rob%C3%B4/software/software-de-aplica%C3%A7%C3%A3o/kuka-forcetorquecontrol (acedido em 2023-02-04) (ver p. 9).

[31] W. Workers. *UR10e Universal Robots*. URL: https://wiredworkers.io/product/ur10e/ (acedido em 2023-02-04) (ver p. 9).

[32] N. Mendes e P. Neto. «Indirect adaptive fuzzy control for industrial robots: A solution for contact applications». Em: *Expert Systems with Applications* 42.22 (2015). DOI: 10.1016/j.eswa.2015.07.047. URL: https://www.sciencedirect.com/science/article/pii/S0957417415005072 (ver p. 10).

[33] C. Lee. «Fuzzy logic in control systems: fuzzy logic controller. I». Em: *IEEE Transactions on Systems, Man, and Cybernetics* 20.2 (1990). DOI: 10.1109/21.52551 (ver p. 10).

[34] A. Winkler e J. Suchy. «Force Controlled Contour Following by an Industrial Robot on Unknown Objects with Tool Orientation Control». Em: *ISR/Robotik 2014; 41st International Symposium on Robotics*. 2014, pp. 1–6. DOI: 10.1109/ROSE.2013.6698444 (ver p. 11).

[35] A. Leite, F. Lizarralde e L. Hsu. «Hybrid vision-force robot control for tasks on unknown smooth surfaces». Em: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2006, pp. 2244–2249. DOI: 10.1109/ROBOT.2006.1642037 (ver p. 11).

[36] U. Robots. *REMOTE OPERATION OF ROBOTS*. 2022-09-29. URL: https://www.universal-robots.com/articles/ur/interface-communication/remote-operation-of-robots/ (acedido em 2023-05-24) (ver p. 25).

[37] U. Robots. *DASHBOARD SERVER E-SERIES, PORT 29999*. 2022-12-15. URL: https://www.universal-robots.com/articles/ur/dashboard-server-e-series-port-29999/ (acedido em 2023-05-24) (ver pp. 25, 74).

[38] U. Robots. *REAL-TIME DATA EXCHANGE (RTDE) GUIDE*. 2023-04-28. URL: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/ (acedido em 2023-05-24) (ver pp. 25, 29).

[39] Digilent. *Getting Started with WaveForms SDK*. URL: https://digilent.com/reference/test-and-measurement/guides/waveforms-sdk-getting-started (acedido em 2023-05-29) (ver p. 26).

[40] Olympus. *NORTEC 600*. URL: https://www.olympus-ims.com/pt/nortec600/ (acedido em 2023-09-12) (ver p. 26).

[41] U. Robots. *UR10e*. URL: https://www.universal-robots.com/products/ur10-robot/ (acedido em 2023-09-12) (ver p. 26).

[42] T. Solutions. *Universal Robots UR10e - Long-Reach Advanced Cobot*. URL: https://thinkbotsolutions.com/products/universal-robots-ur10e (acedido em 2023-09-12) (ver p. 26).

[43] Farnell. *Adapter Board*. URL: https://pt.farnell.com/digilent/410-263/bnc-adapter-board-analog-discovery/dp/2352612 (acedido em 2023-09-12) (ver p. 26).

[44] RS. *Osciloscopio basado en PC Digilent Analog Discovery 2, canales:2 A, 30MHZ, interfaz IIC, SPI, UART, USB*. URL: https://pt.rs-online.com/web/p/osciloscopios/1346480 (acedido em 2023-09-12) (ver p. 26).

[45] S. Verma. *How Fast Numpy Really is and Why?* 2019-12-16. URL: https://towardsdatascience.com/how-fast-numpy-really-is-e9111df44347 (acedido em 2023-06-26) (ver p. 30).

[46] U. Robots. *The URScript Programming Language for e-Series*. 2021 (ver pp. 33, 34).

[47] U. Robots. *URSCRIPT: DYNAMIC FORCE CONTROL*. 2022-12-20. URL: https://www.universal-robots.com/articles/ur/programming/urscript-dynamic-force-control/ (acedido em 2023-08-03) (ver pp. 50, 54).

I

Manual de Utilizador

Antes de poder utilizar a interface gráfica programada (ver capítulo 4), é necessário proceder à configuração do robô e do computador, de forma a que ambos possam comunicar da maneira prevista. É relevante salientar novamente que se utilizou um robô UR10e da Universal Robots e um computador ASUS VivoBook X513UA com o sistema operativo Windows 11.

I.1 Configuração do robô

I.1.1 Ativar o *Remote Control*

O primeiro passo é ativar a possibilidade de operar o robô no modo remoto (*Remote Control*). Para tal, é necessário abrir o *hamburger menu*, cuja localização está assinalada na Figura I.1. De seguida é necessário carregar na opção *Settings*.

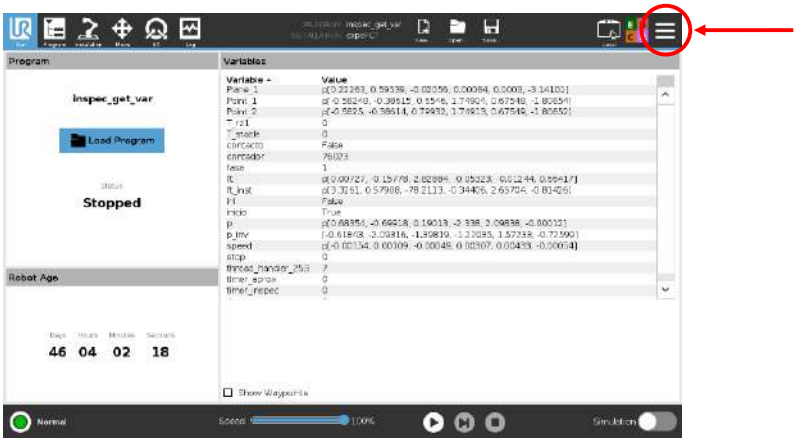


Figura I.1: Localização do *hamburger menu*

Ao abrir o menu *Settings*, deve carregar-se na opção *System*, assinalada com o número 1 na Figura I.2. Tal abre um menu *dropdown*, no qual é necessário escolher a opção *Remote Control*, assinalada com o número 2 na Figura I.2. Desse modo, chega-se à vista apresentada na Figura I.2, devendo carregar-se na opção *Enable*, assinalada com o número 3.

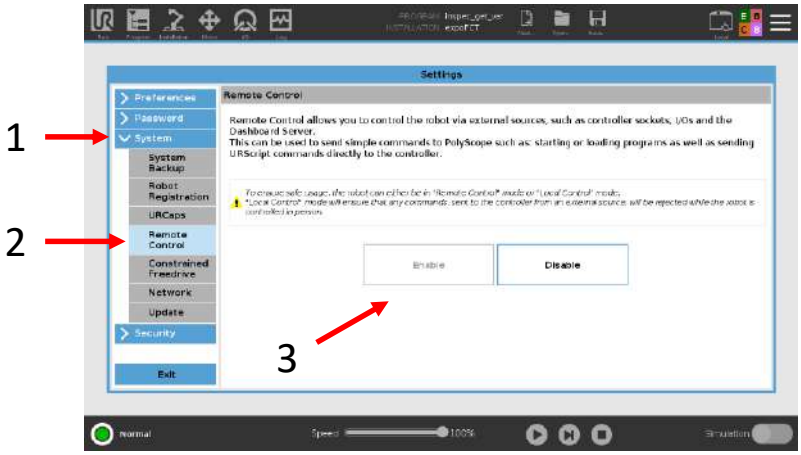


Figura I.2: Localização do botão para ativar o *Remote Control* no robô

I.1.2 Configurar as opções de rede do robô

Seguidamente, é necessário configurar as opções de rede do robô. Novamente, é necessário aceder ao *hamburger menu* (ver Figura I.1), carregar na opção *Settings* e, de seguida, na opção *System* (assinalada com o número 1 na Figura I.2).

Após isso, deve entrar-se no menu *Network*, assinalado na Figura I.3. Neste menu *Network*, deve selecionar-se a opção *Static Address* e definir os IP, *Subnet mask* e *Default gateway* pretendidos, como é visível na Figura I.3.

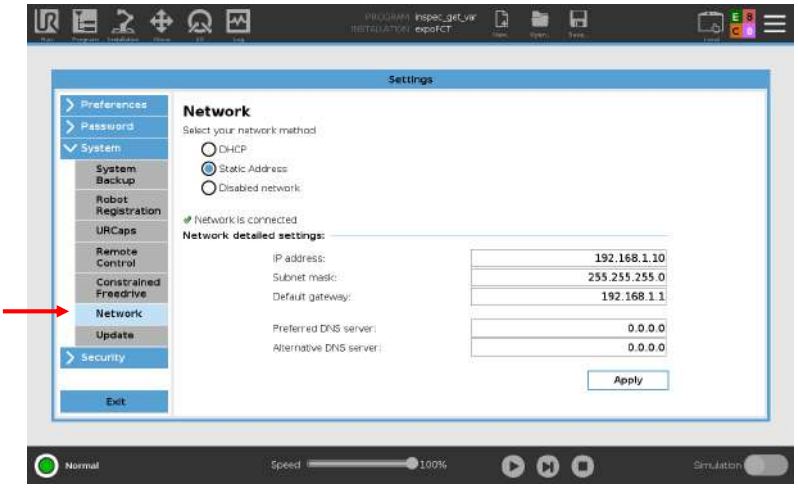


Figura I.3: Localização do menu *Network*

I.2 Configuração do computador

Após completar a configuração do robô, é necessário conectar o mesmo ao computador através de um cabo *ethernet* e proceder à configuração do computador.

Para tal, deve aceder-se ao Painel de Controlo do computador, pesquisar por "ver ligações de rede"e selecionar essa opção, assinalada na Figura I.4.

Figura I.4: Acesso ao menu "Ver ligações de rede"

Ao entrar no menu que contém as ligações de rede do computador, deve identificar-se a ligação com o robô, selecionar a mesma com o botão do lado direito do rato, e selecionar a opção "Propriedades", assinalada na Figura I.5.

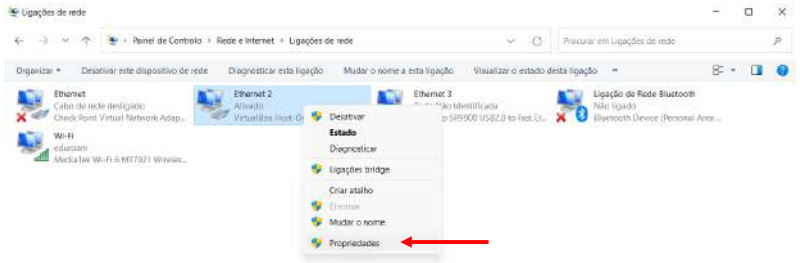


Figura I.5: Acesso às propriedades da ligação

Tal abre o menu presente na Figura I.6, no qual se deve carregar na opção "Protocolo IP Versão 4 (TCP/IPv4)", e, de seguida, no botão "Propriedades"assinalado na Figura I.6.

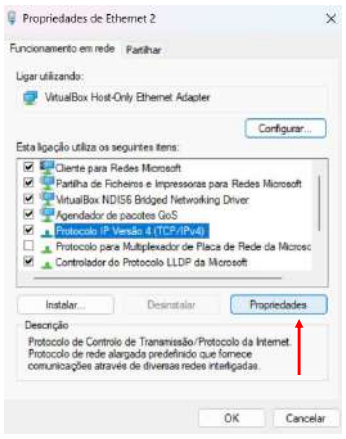


Figura I.6: Acesso às propriedades da ligação IPv4

Chega-se então ao menu apresentado na Figura I.7, no qual se deve selecionar a opção assinalada: "Utilizar o seguinte endereço de IP". De seguida, deve definir-se o endereço IP como sendo semelhante ao escolhido previamente no robô (por exemplo diferindo apenas

no último algarismo). Além disso, a máscara de sub-rede deve ser igual à utilizada no robô (no qual esta se denomina *Subnet mask*).

Figura I.7: Menu de definição das opções de rede

Por fim, deve ser instalado o *software* Waveforms (em particular o Waveforms SDK)

I.3 Funcionamento da interface gráfica

A lógica por detrás da programação da interface gráfica é que ao carregar num determinado botão, é chamada a função correspondente, que pode eventualmente conter como argumento ou informação auxiliar o texto presente nas caixas visíveis na Figura 4.2. Os *outputs* das funções e as instruções que guiam o utilizador pela utilização do programa são apresentados na caixa de texto presente na metade inferior da interface, denominada de *Feedback*. Todos estes aspetos são cobertos pelas funcionalidades da biblioteca tkinter.

Relativamente ao funcionamento do programa desenvolvido no decurso desta dissertação, o primeiro passo deve ser sempre conectar o robô ao computador, e, para isso, deve ser fornecido o IP do robô e a *communication port* a utilizar (30004 para o protocolo RTDE). Antes de tal ser realizado, não deve ser possível realizar qualquer outra ação, e, desse modo, todos os botões da interface menos o botão *Connect* se encontram desativados, algo que é visível na Figura 4.2.

Após isso, o programa procura certificar-se de que duas condições são cumpridas. Por um lado, que o robô se encontra no modo remoto, caso contrário não é possível utilizar a maior parte dos comandos disponíveis a partir do servidor *Dashboard*, algo que severamente reduz a utilidade da interface. Desse modo, o utilizador é instruído a fazê-lo, como é visível na Figura I.8. Nota-se que os comandos do servidor *Dashboard* podem ser enviados escrevendo os mesmos na caixa de texto vazia visível na Figura I.8 e carregando no botão "Send". A lista de comandos disponíveis pode ser consultada em [37].

Por outro lado, o programa procura confirmar que o robô se encontra disponível para operar, ou seja, que está a ser fornecida potência aos motores, que os travões estão desativados, e que não existe qualquer outro erro que impeça o normal funcionamento do robô. Caso esse não seja o caso, são fornecidas instruções na *Feedback box* para o utilizador

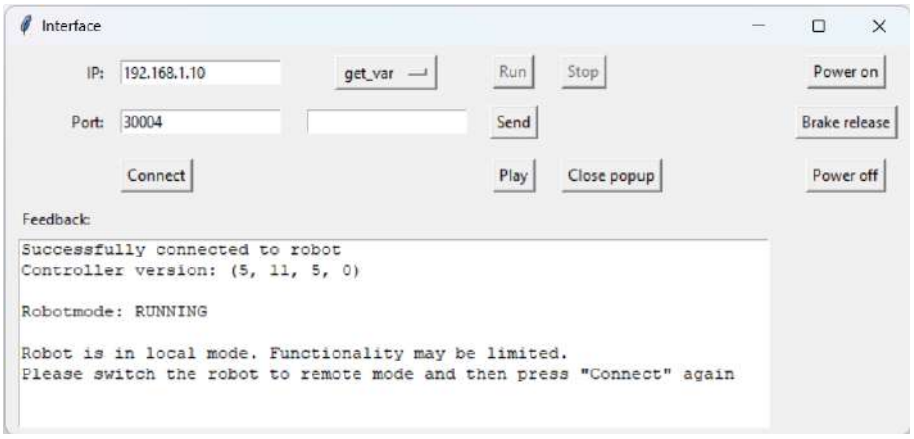


Figura I.8: Robô em controlo local

colocar o robô no seu modo normal de funcionamento, como é visível na Figura I.9. Os botões "Power on", "Brake release", presentes do lado direito da interface gráfica (ver Figura I.9), auxiliam nesta tarefa de colocar o robô em funcionamento, destinando-se o botão "Power off"a retirar potência aos motores do robô.

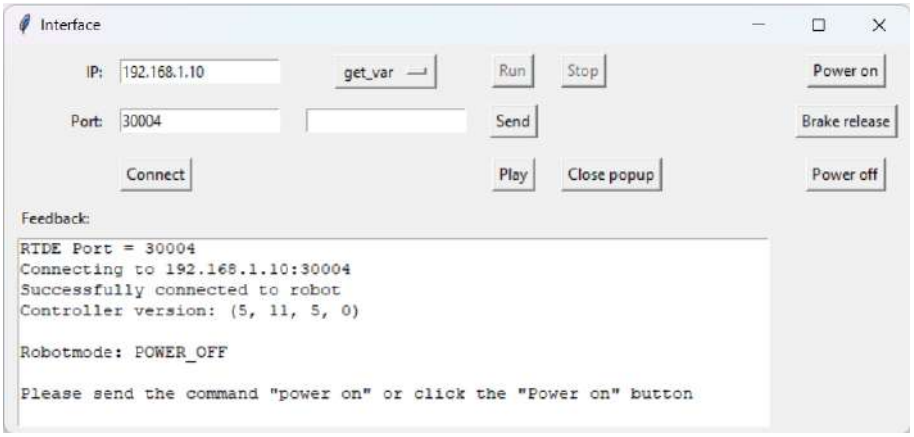


Figura I.9: Instruções para ativação do robô

Concluídas estas etapas, estão reunidas as condições para ser corrido o programa de inspeção, denominado get_var, algo que pode ser feito escolhendo o mesmo no menu *dropdown* em que se encontram todos os programas disponíveis na diretoria e depois carregando no botão *Run*. Tal é visível na Figura I.10. Algo a salientar é que, neste momento, o programa de inspeção .urp deve estar carregado no robô e pronto a correr.

Caso o programa .urp ainda não esteja em andamento quando o procedimento realizado no parágrafo anterior é realizado, a interface gráfica fornecerá instruções no sentido de o pôr a correr, utilizando-se o botão "Play"da interface gráfica.

De seguida, o programa .urp sofre uma paragem temporária, aparecendo a mensagem presente na Figura I.11. O objetivo desta paragem temporária antes da execução do programa tem a ver com o facto de ser crucial existir sincronismo no início e decorrer da execução do programa em Python e do programa que corre no próprio robô. Para

Figura I.10: Escolha do programa a correr

prosseguir, devem seguir-se as instruções fornecidas na *Feedback box*, ou seja, carregar no botão *Close popup* da interface.

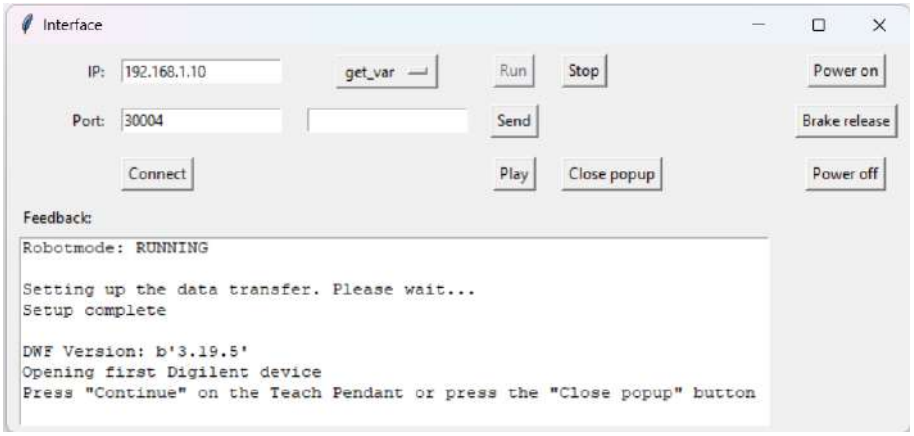


Figura I.11: Paragem antes da continuação da execução do programa .urp

Se se inverter a ordem, e se iniciar o programa .urp antes de carregar no botão "Run"para iniciar o programa get_var, então a interface gráfica é capaz de reconhecer isso e apresentar logo a instrução da Figura I.11.

II

Programa de interface gráfica

```
import tkinter as tk

import threading
import os
import sys
import logging
import rtde.rtde as rtde
import rtde.rtde_config as rtde_config
import datetime
import time
import openpyxl
import dashboard.Dashboard as dashboard

#from ctypes import *
from Digilent.dwfconstants import *
import numpy as np
#import matplotlib.pyplot as plt

ROBOT_HOST = '' #Variável global que armazena o IP do robô
ROBOT_PORT = 0  #Variável global que armazena a porta de comunicação definida

"""
Utilizam-se variáveis globais para que todas as funções implementadas
tenham acesso às variáveis de IP e communication port, que são definidas
no interior da função "Connect".
"""
```

```
#Classes
class AutoScrollText(tk.Text):
"""
para a feedback box ir deslizando para baixo à medida que
vai aparecendo novo texto
"""
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def insert(self, index, chars, *args):
        super().insert(index, chars, *args)
        self.see(tk.END)

#Funções

def connect():
    """
    Esta função é chamada quando o botão "Connect" da interface é premido.
    Recebe informação dos campos IP e Port da interface e estabelece
    a comunicação entre robô e computador.
    """
    global ROBOT_HOST
    global ROBOT_PORT
    send_button.config(state='normal')
    #Ativa a possibilidade do envio de mensagens para o robô através do botão "Send"
    ip = ip_entry.get()
    port = port_entry.get()
    feedback_box.insert(tk.END, f"Connecting to {ip}:{port}\n")

    ROBOT_HOST = ip #IP do robô
    ROBOT_PORT = int(port) #Communication port

    logging.getLogger().setLevel(logging.INFO)

    #-------------Estabelece comunicação com o robô

    try:
        con = rtde.RTDE(ROBOT_HOST, ROBOT_PORT)
        connection_state = con.connect()
        dash = dashboard.Dashboard(ROBOT_HOST)
        dash_connection_state = dash.connect()
```

```
# confirma que a conexão teve sucesso
while connection_state is not None and dash_connection_state is not None:
    connection_state = con.connect()
    dash_connection_state = dash.connect()
feedback_box.insert(tk.END, "Successfully connected to robot\n")
pwr_on_button.config(state='normal')
brake_button.config(state='normal')
pwr_off_button.config(state='normal')

#Obtém a versão atual do controlador e confirma se está atualizada o suficiente

con.get_controller_version()
cont_version = con.get_controller_version()
if con.get_controller_version()[0:4] == (None, None, None, None):
    feedback_box.insert(tk.END, "Please upgrade your controller to
                        minimally version 3.2.19171\n")
else:
    feedback_box.insert(tk.END, f"Controller version: {cont_version}\n")
    feedback_box.insert(tk.END, "\n")

#Confirma se o robô está no modo remoto.
#Se estiver, confirma se o robô está operacional.

robot_mode = dash.sendAndReceive('robotmode')
print(robot_mode)
feedback_box.insert(tk.END, f"{robot_mode}\n")
feedback_box.insert(tk.END, "\n")
remoteCheck = dash.sendAndReceive('is in remote control')
if 'false' in remoteCheck:
    print('Robot is in local mode. Functionality may be limited.\n')
    print('Please switch the robot to remote mode and then
          press \"Connect\" again\n')
    feedback_box.insert(tk.END, 'Robot is in local mode.
                        Functionality may be limited.\n')
    feedback_box.insert(tk.END, 'Please switch the robot to
                        remote mode and then press \"Connect\" again\n')
    feedback_box.insert(tk.END, "\n")

#Caso o robô não esteja operacional, é ativada
#uma sequência para guiar o utilizador a corrigir isso
```

```
        elif robot_mode == 'RUNNING':
            run_button.config(state='normal')
            stop_button.config(state='normal')

        #Essa sequência é ativada chamando a função robot_boot

        else:
            thread = threading.Thread(target=robot_boot)
            thread.start()

    except Exception as e:
        error_message = str(e)
        feedback_box.insert(tk.END, f"{error_message}\n")

def robot_boot():
    """
    Esta função pode ser chamada no decorrer de uma tentativa de conectar o robô
    o robô não estar operacional (por exemplo se não estiver a ser fornecida
    potência aos motores ou se os travões estiverem ativos).
    As instruções são fornecidas ao utilizador através da feedback box da interface e,
    salvo casos bastante excecionais, envolvem apenas carregar em botões presentes na
    própria interface.
    """
    global ROBOT_HOST
    global ROBOT_PORT
    #Informações para conexão com o servidor Dashboard
    dash = dashboard.Dashboard(ROBOT_HOST)
    dash.connect() #Estabelece a conexão com o servidor Dashboard
    robot_mode = dash.sendAndReceive('robotmode') #Adquire o modo atual do robô
    while 'RUNNING' not in robot_mode:
        if 'NO_CONTROLLER' in robot_mode or 'DISCONNECTED' in robot_mode
                or'CONFIRM_SAFETY' in robot_mode or 'BACKDRIVE' in robot_mode:
            print('Please take the appropriate action at the Teach Pendant
                  or the Controller Box\n')
            feedback_box.insert(tk.END, 'Take the appropriate action
                                at the Teach Pendant or the Controller Box\n')
        elif 'POWER_OFF' in robot_mode:
            print('Please send the command \"power on\"')
            feedback_box.insert(tk.END, 'Please send the command \"power on\" or
            ´                   click the \"Power on\" button\n')
            while 'IDLE' not in robot_mode:
```

```
                robot_mode = dash.sendAndReceive('robotmode')
        elif 'BOOTING' in robot_mode or 'POWER_ON' in robot_mode:
            print('Please wait while robot is booting up\n')
            feedback_box.insert(tk.END, 'Please wait while robot is booting up\n')
            while 'IDLE' not in robot_mode:
                robot_mode = dash.sendAndReceive('robotmode')
        elif 'IDLE' in robot_mode:
            print('Please send the command \"brake release\"\n')
            feedback_box.insert(tk.END, 'Please send the command \"brake release\" or
                                click the \"Brake release\" button\n')
            while 'RUNNING' not in robot_mode:
                robot_mode = dash.sendAndReceive('robotmode')
        robot_mode = dash.sendAndReceive('robotmode')
        print(robot_mode)
        feedback_box.insert(tk.END,f"{robot_mode}\n")

    # Após o robô estar operacional, deve ser permitido correr programas
    #(e, consequentemente, pará-los), daí as duas linhas seguintes,
    # que ativam os botões que desempenham esses papéis

    run_button.config(state='normal')
    stop_button.config(state='normal')

def run_button_clicked():
    """
    Esta função é chamada quando o botão "Run" da interface é premido e
    inicia uma thread com a função que estiver escolhida no menu dropdown.
    Caso seja necessário adicionar um programa diferente,
    deve ser criada uma função com as funcionalidades desejadas e
    devem ser adicionadas à função run_button_clicked() as seguintes linhas:
    selected_program = program_var.get()
    if selected_program == [nome da função]:
        thread = threading.Thread(target=[nome da função])
        thread.start()
    Além disso, [nome da função] deve ser adicionado ao vetor programs,
    na parte final do código, destinada à criação da interface gráfica
    """

    #esta variável recebe o nome do programa escolhido no menu dropdown:
    selected_program = program_var.get()
    if selected_program == 'get_var':
```

```
        thread = threading.Thread(target=get_var)
        thread.start()

def stop_button_clicked():
    """
    Esta função é chamada quando o botão "Stop" da interface é premido
    e interrompe o programa .urp em execução no robô, bem como
    a aquisição de dados pelo Digilent.
    """

    global ROBOT_HOST
    dash = dashboard.Dashboard(ROBOT_HOST)
    dash.connect()
    feedback_box.insert(tk.END, "\n")
    feedback_box.insert(tk.END, "Stop button was pressed\n")
    #confirmar o estado de execução do programa do robô:
    program_state = dash.sendAndReceive('programState')
    if 'STOPPED' in program_state:
        feedback_box.insert(tk.END, "Program is already stopped\n")
    else:
        #utilização do servidor Dashboard para enviar instruções ao robô:
        dash.sendAndReceive('stop')

    #As linhas seguintes destinam-se a parar o funcionamento do Digilent

    if sys.platform.startswith("win"):
        dwf = cdll.dwf
    elif sys.platform.startswith("darwin"):
        dwf = cdll.LoadLibrary("/Library/Frameworks/dwf.framework/dwf")
    else:
        dwf = cdll.LoadLibrary("libdwf.so")
    dwf.FDwfDeviceCloseAll()
    run_button.config(state="normal")


def pwr_on_button_clicked():
    """
    Esta função é chamada quando o botão "Power on" da interface é premido
    e abre uma thread com a função destinada a enviar potência
    aos motores do robô (pwr_on).
    """
```

```
    thread = threading.Thread(target=pwr_on) #Define a função a iniciar na thread
    thread.start() #Inicia a thread

def pwr_on():
    global ROBOT_HOST #Variável global que contém o IP do robô
    dash = dashboard.Dashboard(ROBOT_HOST) #Dados para a conexão ao servidor Dashboard
    dash.connect() #Inicia a conexão com o servidor Dashboard
    #Enviar o comando para fornecer potência aos motores do robô:
    dash.sendAndReceive('power on')
    feedback_box.insert(tk.END, "Powering on...\n") #Output na janela de feedback
    robot_mode = dash.sendAndReceive('robotmode') #Variável que recebe o estado do robô
    #Enquanto não for atingido determinado estado do robô, este ciclo deve continuar:
    while 'IDLE' not in robot_mode:
        robot_mode = dash.sendAndReceive('robotmode')
    feedback_box.insert(tk.END, "Power is on\n") #Output na janela de feedback
    feedback_box.insert(tk.END, "\n")

def brake_button_clicked():
    """
    Esta função é chamada quando o botão "Brake release" da interface é premido
    e abre uma thread com a função destinada a destravar o robô (brake_release).
    """
    thread = threading.Thread(target=brake_release)
    thread.start()

def brake_release():
    global ROBOT_HOST #Variável global que contém o IP do robô
    dash = dashboard.Dashboard(ROBOT_HOST) #Dados para a conexão ao servidor Dashboard
    dash.connect() #Inicia a conexão com o servidor Dashboard
    dash.sendAndReceive('brake release') #Envio do comando para destravar o robô
    feedback_box.insert(tk.END, "Brake releasing...\n") #Output na janela de feedback
    robot_mode = dash.sendAndReceive('robotmode')  #Variável que recebe o estado do robô
    #Enquanto não for atingido determinado estado do robô, este ciclo deve continuar:
    while 'RUNNING' not in robot_mode:
        robot_mode = dash.sendAndReceive('robotmode')
    feedback_box.insert(tk.END, "Brake is released\n") #Output na janela de feedback
    feedback_box.insert(tk.END, "\n")

def pwr_off_button_clicked():
    """
    Esta função é chamada quando o botão "Power off" da interface é premido
```

```
    e abre uma thread com a função destinada a retirar potência
    aos motores do robô (pwr_off).
    """
    thread = threading.Thread(target=pwr_off)
    thread.start()

def pwr_off():
    global ROBOT_HOST #Variável global que contém o IP do robô
    dash = dashboard.Dashboard(ROBOT_HOST)  #Dados para a conexão ao servidor Dashboard
    dash.connect()  #Inicia a conexão com o servidor Dashboard
    #Enviar o comando para retirar potência aos motores do robô:
    dash.sendAndReceive('power off')
    feedback_box.insert(tk.END, "Powering off...\n") #Output na janela de feedback
    robot_mode = dash.sendAndReceive('robotmode')  #Variável que recebe o estado do robô
    #Enquanto não for atingido determinado estado do robô, este ciclo deve continuar:
    while 'POWER_OFF' not in robot_mode:
        robot_mode = dash.sendAndReceive('robotmode')
    feedback_box.insert(tk.END, "Power is off\n") #Output na janela de feedback
    feedback_box.insert(tk.END, "\n")

def send():
    """
    Esta função é chamada quando o botão "Send" da interface é premido
    e envia através do servidor Dashboard o comando preenchido pelo utilizador
    na caixa de texto para esse fim destinada.
    """
    global ROBOT_HOST
    dash = dashboard.Dashboard(ROBOT_HOST)
    dash.connect()
    message = send_entry.get()
    feedback_box.insert(tk.END, f"Sending comand: {message}\n")
    print(dash.sendAndReceive(message))
    feedback_box.insert(tk.END, f"{dash.sendAndReceive(message)}\n")

def play_button_clicked():
    """
    Esta função é chamada quando o botão "Play" da interface é premido
    e envia através do servidor Dashboard um comando
    para colocar em execução o programa carregado no robô.
    """
    global ROBOT_HOST
```

```
    dash = dashboard.Dashboard(ROBOT_HOST)
    dash.connect()
    prog_state = dash.sendAndReceive('programState')
    if prog_state == 'PLAYING':
        feedback_box.insert(tk.END, "A Polyscope program is already playing")
    else:
        feedback_box.insert(tk.END, f"{dash.sendAndReceive('play')}\n")

def close_popup_button_clicked():
    """
    Esta função é chamada quando o botão "Play" da interface é premido
    e envia através do servidor Dashboard um comando para fechar um
    eventual popup que apareça durante a execução de um programa .urp no robô.
    """
    global ROBOT_HOST
    dash = dashboard.Dashboard(ROBOT_HOST)
    dash.connect()
    feedback_box.insert(tk.END, f"{dash.sendAndReceive('close popup')}\n")



def get_var():
    """
    Programa destinado a realizar o ajustamento ao cordão e
    a inspeção por correntes induzidas ao mesmo.
    Integra comunicação com o robô para receber dados de posição e força
    ao longo do decorrer da execução do programa .urp (inspec_get_var.urp),
    bem como comunicação com o Digilent Analog Discovery 2, de forma a
    receber dados relativos à inspeção por correntes induzidas.
    """

    global ROBOT_HOST
    global ROBOT_PORT

    #para não ser possível correr dois programas ao mesmo tempo:
    run_button.config(state="disable")

    # -------------Mensagem inicial
    feedback_box.insert(tk.END, "Setting up the data transfer. Please wait...\n")
    print("Setting up the data transfer. Please wait...\n")
    # -------------Informações para guardar o ficheiro .xlsx gerado
```

```
#variável que recebe a diretoria atual:
current_dir = os.path.dirname(os.path.abspath('Interface_inspec.py'))
#definir que os ficheiros xlsx sejam gravados na pasta "Dados" da diretoria atual
save_dir = f"{current_dir}/Dados/"

# -------------Informações para comunicação com o robô
# Alterar estas definições consoante necessário
config_filename = "get_var.xml"  # Ficheiro com as recipes

logging.getLogger().setLevel(logging.INFO)

conf = rtde_config.ConfigFile(config_filename)
state_names, state_types = conf.get_recipe("state")  # recipe key
watchdog_names, watchdog_types = conf.get_recipe("watchdog")  # recipe key

# -------------Funções auxiliares
def date_time():
    # Get the current date and time
    current_datetime = datetime.datetime.now()
    formatted_datetime = current_datetime.strftime("%Y-%m-%d_%H-%M-%S")
    return formatted_datetime

def apaga_7(linha, worksheet):
    row_number = linha  # Replace with the desired row number
    for column in range(1, 8):  # Iterate from column 1 to 7 (inclusive)
        cell = worksheet.cell(row=row_number, column=column)
        cell.value = None

# -------------Estabelece comunicação com o robô
con = rtde.RTDE(ROBOT_HOST, ROBOT_PORT)
connection_state = con.connect()
dash = dashboard.Dashboard(ROBOT_HOST)
dash_connection_state = dash.connect()

# -------------Confirma que a conexão teve sucesso

while connection_state != None and dash_connection_state != None:
    connection_state = con.connect()
    dash_connection_state = dash.connect()
feedback_box.insert(tk.END, "Setup complete\n")
print("Setup complete\n")
```

```
# -------------Obtém a versão atual do controlador
con.get_controller_version()

# -------------Setup recipes
#permite alterar a freq de receção de dados do robô (500 Hz no máximo):
FREQ = 500
con.send_output_setup(state_names, state_types, FREQ)
watchdog = con.send_input_setup(watchdog_names, watchdog_types)

# -------------Cria os objetos para escrever os dados
formatted_datetime = date_time()
workbook = openpyxl.Workbook()  # Cria um novo ficheiro excel
worksheet = workbook.active  # Seleciona a folha ativa (a 1ª)
worksheet.title = 'Aprox_TCP_pose'
worksheet.cell(1, 1).value = 'Tempo [s]'
worksheet.cell(1, 2).value = 'X [m]'
worksheet.cell(1, 3).value = 'Y [m]'
worksheet.cell(1, 4).value = 'Z [m]'
worksheet.cell(1, 5).value = 'RX [rad]'
worksheet.cell(1, 6).value = 'RY [rad]'
worksheet.cell(1, 7).value = 'RZ [rad]'
worksheet.cell(1, 8).value = 'BASE FRAME'
workbook.create_sheet(title='Aprox_TCP_force')
worksheet = workbook['Aprox_TCP_force']
worksheet.cell(1, 1).value = 'Tempo [s]'
worksheet.cell(1, 2).value = 'FX [N]'
worksheet.cell(1, 3).value = 'FY [N]'
worksheet.cell(1, 4).value = 'FZ [N]'
worksheet.cell(1, 5).value = 'MX [N.m]'
worksheet.cell(1, 6).value = 'MY [N.m]'
worksheet.cell(1, 7).value = 'MZ [N.m]'
worksheet.cell(1, 8).value = 'TOOL FRAME'
workbook.create_sheet(title='Inspec_TCP_pose')
worksheet = workbook['Inspec_TCP_pose']
worksheet.cell(1, 1).value = 'Tempo [s]'
worksheet.cell(1, 2).value = 'Dist [m]'
worksheet.cell(1, 3).value = 'X [m]'
worksheet.cell(1, 4).value = 'Y [m]'
worksheet.cell(1, 5).value = 'Z [m]'
worksheet.cell(1, 6).value = 'RX [rad]'
```

```
worksheet.cell(1, 7).value = 'RY [rad]'
worksheet.cell(1, 8).value = 'RZ [rad]'
worksheet.cell(1, 9).value = 'CH1 [V]'
worksheet.cell(1, 10).value = 'CH2 [V]'
worksheet.cell(1, 11).value = 'BASE FRAME'
workbook.create_sheet(title='Inspec_TCP_force')
worksheet = workbook['Inspec_TCP_force']
worksheet.cell(1, 1).value = 'Tempo [s]'
worksheet.cell(1, 2).value = 'Dist [m]'
worksheet.cell(1, 3).value = 'FX [N]'
worksheet.cell(1, 4).value = 'FY [N]'
worksheet.cell(1, 5).value = 'FZ [N]'
worksheet.cell(1, 6).value = 'MX [N.m]'
worksheet.cell(1, 7).value = 'MY [N.m]'
worksheet.cell(1, 8).value = 'MZ [N.m]'
worksheet.cell(1, 9).value = 'TOOL FRAME'

# -------------?
watchdog.input_int_register_0 = 0
watchdog.input_int_register_1 = 0

# -------------Inicia a sincronização dos dados
if not con.send_start():
    sys.exit()
con.send(watchdog)

# --------------Informação Digilent--------------#

if sys.platform.startswith("win"):
    dwf = cdll.dwf
elif sys.platform.startswith("darwin"):
    dwf = cdll.LoadLibrary("/Library/Frameworks/dwf.framework/dwf")
else:
    dwf = cdll.LoadLibrary("libdwf.so")

#Declarar variáveis ctype (documentação Digilent Waveforms SDK)
hdwf = c_int()
sts = c_byte()
hzAcq = c_double(1000)
rgdSamples1 = c_double()
rgdSamples2 = c_double()
```

```
# Apresenta a versão atual do Digilent Waveforms
version = create_string_buffer(16)
dwf.FDwfGetVersion(version)
print("DWF Version: " + str(version.value))
feedback_box.insert(tk.END,"\n")
feedback_box.insert(tk.END, f"DWF Version: {str(version.value)}\n")

#Estabelece a conexão entre o computador e o Digilent
print("Opening first Digilent device\n")
feedback_box.insert(tk.END, "Opening first Digilent device\n")
dwf.FDwfDeviceOpen(c_int(-1), byref(hdwf))

#Caso a conexão não tenha sucesso, este if fornece essa informação ao utilizador
if hdwf.value == hdwfNone.value:
    szerr = create_string_buffer(512)
    dwf.FDwfGetLastErrorMsg(szerr)
    print(str(szerr.value))
    print("failed to open Digilent device")
    feedback_box.insert(tk.END, f"DWF Version: {str(szerr.value)}\n")
    feedback_box.insert(tk.END, "failed to open Digilent device\n")
    quit()

#Configuração da aquisição de dados do Digilent
dwf.FDwfAnalogInChannelEnableSet(hdwf, c_int(0), c_int(1))
dwf.FDwfAnalogInChannelEnableSet(hdwf, c_int(0), c_int(2))
# Intervalo de tensão lido (neste caso: [0,5]):
dwf.FDwfAnalogInChannelRangeSet(hdwf, c_int(0), c_double(5))
dwf.FDwfAnalogInAcquisitionModeSet(hdwf, acqmodeRecord)
dwf.FDwfAnalogInFrequencySet(hdwf, hzAcq)
#Para não existir à partida limite para a duração da gravação:
dwf.FDwfAnalogInRecordLengthSet(hdwf, c_double(0))


# -------------Espera que se dê autorização no Polyscope para retomar o programa
state = con.receive()
first_run1 = True
while state.runtime_state != 2:
    state = con.receive()
    if first_run1 == True:
        first_run1 = False
```

```
        print("")
        print("Polyscope program is stopped, please start it
              before running get_var.py")
        feedback_box.insert(tk.END, "\n")
        feedback_box.insert(tk.END, "Polyscope program is stopped\n")
        feedback_box.insert(tk.END, "Press the \"Play\" button to
                            run the Polyscope program\n")

time.sleep(0.1)

feedback_box.insert(tk.END, "Press \"Continue\" on the Teach Pendant
                    or press the \"Close popup\" button\n")
print("Press \"Continue\" on the Teach Pendant
      or press the \"Close popup\" button\n")
while True:
    state = con.receive()
    con.send(watchdog)
    if state.output_bit_registers0_to_31 == True:
        feedback_box.insert(tk.END, "Program starting...\n")
        print("Program starting...\n")
        break
    if state.runtime_state == 1:
        print("")
        print("Polyscope program is stopped, please start it
              before running get_var.py")
        # print('Terminating the connection...')
        feedback_box.insert(tk.END, "\n")
        feedback_box.insert(tk.END, "Polyscope program is stopped\n")
        feedback_box.insert(tk.END, "Send the command \"play\" to
                            run the Polyscope program\n")
        sys.exit(0)

# -------------Fase de aproximação ao cordão

watchdog.input_int_register_0 = 1
con.send(watchdog)  # fase de aproximação == 1

print("Executing the approach phase...")
feedback_box.insert(tk.END, "Executing the approach phase...\n")

first_run = True
```

```
counter_aprox = 1

"""
Iniciar os vetores onde serão armazenados os dados provenientes do robô
relativos à sua operação:
"""

aprox_tempo = []
aprox_tcp_pose_x = []
aprox_tcp_pose_y = []
aprox_tcp_pose_z = []
aprox_tcp_pose_rx = []
aprox_tcp_pose_ry = []
aprox_tcp_pose_rz = []
aprox_tcp_force_x = []
aprox_tcp_force_y = []
aprox_tcp_force_z = []
aprox_tcp_force_rx = []
aprox_tcp_force_ry = []
aprox_tcp_force_rz = []


while True:
    state = con.receive()
    con.send(watchdog)
    while state.output_bit_register_64 == True and state.runtime_state != 1
              and con.is_connected():
        #Inicia-se aqui a leitura e gravação dos dados provenientes do robô
        if first_run == True:
            print('Probe contact established')
            feedback_box.insert(tk.END, "Probe contact established\n")
            first_run = False
        #sinal do robô de que a fase de ajustamento terminou:
        if state.output_bit_registers0_to_31 == False:
            print('Approach phase completed')
            feedback_box.insert(tk.END, "Approach phase completed\n")
            break
        state = con.receive()
        con.send(watchdog)
        counter_aprox = counter_aprox + 1
        # Coluna Tempo
```

```
            aprox_tempo.append(state.output_double_register_30)
            # Colunas Pose
            aprox_tcp_pose_x.append(state.actual_TCP_pose[0])
            aprox_tcp_pose_y.append(state.actual_TCP_pose[1])
            aprox_tcp_pose_z.append(state.actual_TCP_pose[2])
            aprox_tcp_pose_rx.append(state.actual_TCP_pose[3])
            aprox_tcp_pose_ry.append(state.actual_TCP_pose[4])
            aprox_tcp_pose_rz.append(state.actual_TCP_pose[5])
            # Colunas Força
            aprox_tcp_force_x.append(state.output_double_register_24)
            aprox_tcp_force_y.append(state.output_double_register_25)
            aprox_tcp_force_z.append(state.output_double_register_26)
            aprox_tcp_force_rx.append(state.output_double_register_27)
            aprox_tcp_force_ry.append(state.output_double_register_28)
            aprox_tcp_force_rz.append(state.output_double_register_29)

        #para o caso de o programa .urp ser parado a meio:
        if state.runtime_state == 1:
            print("")
            print("Polyscope program was stopped")
            feedback_box.insert(tk.END, "\n")
            feedback_box.insert(tk.END, "Polyscope program was stopped\n")
            sys.exit(0)
        if state.output_bit_registers0_to_31 == False:
            break



# -------------Fase de inspeção do cordão

watchdog.input_int_register_0 = 2
con.send(watchdog)  # fase de inspeção == 2

print("Executing inspection phase...")
feedback_box.insert(tk.END, "Executing inspection phase...\n")

#Número limite de dados que podem ser adquiridos
N = 20000
counter = 0

#Vetores para armazenar os dados relativos à operação do robô
```

```
insp_tempo = np.zeros(N)
insp_tcp_pose_x = np.zeros(N)
insp_tcp_pose_y = np.zeros(N)
insp_tcp_pose_z = np.zeros(N)
insp_tcp_pose_rx = np.zeros(N)
insp_tcp_pose_ry = np.zeros(N)
insp_tcp_pose_rz = np.zeros(N)
insp_tcp_force_x = np.zeros(N)
insp_tcp_force_y = np.zeros(N)
insp_tcp_force_z = np.zeros(N)
insp_tcp_force_rx = np.zeros(N)
insp_tcp_force_ry = np.zeros(N)
insp_tcp_force_rz = np.zeros(N)

#Vetores para armazenar os dados relativos à inspeção por correntes induzidas

digi_ch1 = np.zeros(N)
digi_ch2 = np.zeros(N)

while True:
    state = con.receive()
    con.send(watchdog)
    # Início de aquisição Digilent:
    dwf.FDwfAnalogInConfigure(hdwf, c_int(0), c_int(1))
    while state.output_bit_register_64 == False and state.runtime_state != 1:
        # ------------- PARTE RTDE
        state = con.receive()
        con.send(watchdog)
        # Coluna Tempo
        insp_tempo[counter] = state.output_double_register_30
        # Colunas Pose
        insp_tcp_pose_x[counter] = state.actual_TCP_pose[0]
        insp_tcp_pose_y[counter] = state.actual_TCP_pose[1]
        insp_tcp_pose_z[counter] = state.actual_TCP_pose[2]
        insp_tcp_pose_rx[counter] = state.actual_TCP_pose[3]
        insp_tcp_pose_ry[counter] = state.actual_TCP_pose[4]
        insp_tcp_pose_rz[counter] = state.actual_TCP_pose[5]
        # Colunas força
        insp_tcp_force_x[counter] = state.output_double_register_24
        insp_tcp_force_y[counter] = state.output_double_register_25
        insp_tcp_force_z[counter] = state.output_double_register_26
```

```
            insp_tcp_force_rx[counter] = state.output_double_register_27
            insp_tcp_force_ry[counter] = state.output_double_register_28
            insp_tcp_force_rz[counter] = state.output_double_register_29

            # ---------------- PARTE DIGILENT
            dwf.FDwfAnalogInStatus(hdwf, c_int(0), byref(sts))
            #Canal 1 Digilent:
            dwf.FDwfAnalogInStatusSample(hdwf, c_int(0), byref(rgdSamples1))
            #Canal 2 Digilent:
            dwf.FDwfAnalogInStatusSample(hdwf, c_int(1), byref(rgdSamples2))
            digi_ch1[counter] = rgdSamples1.value
            digi_ch2[counter] = rgdSamples2.value

            counter = counter + 1

            #sinal do robô de que a fase de ajustamento terminou:
            if state.output_bit_registers0_to_31 == True:
                print('Inspection phase completed')
                feedback_box.insert(tk.END, "Inspection phase completed\n")
                break
        if state.runtime_state == 1:
            print("")
            print("Polyscope program was stopped")
            feedback_box.insert(tk.END, "\n")
            feedback_box.insert(tk.END, "Polyscope program was stopped\n")
            sys.exit(0)
        if state.output_bit_registers0_to_31 == True:
            break

    #print('O ciclo while correu as seguintes vezes: ',counter)

    feedback_box.insert(tk.END, "Polyscope program concluded\n")

    #TERMINAR CONEXÃO COM O DIGILENT
    dwf.FDwfDeviceCloseAll()

    print('Writing and saving data...')
    feedback_box.insert(tk.END, 'Writing and saving data...\n')
```

```
#Escrever dados robô
worksheet = workbook['Aprox_TCP_pose']
for i in range(len(aprox_tempo)):
    worksheet.cell(i + 2, 1).value = aprox_tempo[i]
    worksheet.cell(i + 2, 2).value = aprox_tcp_pose_x[i]
    worksheet.cell(i + 2, 3).value = aprox_tcp_pose_y[i]
    worksheet.cell(i + 2, 4).value = aprox_tcp_pose_z[i]
    worksheet.cell(i + 2, 5).value = aprox_tcp_pose_rx[i]
    worksheet.cell(i + 2, 6).value = aprox_tcp_pose_ry[i]
    worksheet.cell(i + 2, 7).value = aprox_tcp_pose_rz[i]
    if aprox_tcp_pose_x[i] == 0:
        break

worksheet = workbook['Aprox_TCP_force']
for i in range(len(aprox_tempo)):
    worksheet.cell(i + 2, 1).value = aprox_tempo[i]
    worksheet.cell(i + 2, 2).value = aprox_tcp_force_x[i]
    worksheet.cell(i + 2, 3).value = aprox_tcp_force_y[i]
    worksheet.cell(i + 2, 4).value = aprox_tcp_force_z[i]
    worksheet.cell(i + 2, 5).value = aprox_tcp_force_rx[i]
    worksheet.cell(i + 2, 6).value = aprox_tcp_force_ry[i]
    worksheet.cell(i + 2, 7).value = aprox_tcp_force_rz[i]
    if aprox_tcp_pose_x[i] == 0:
        break

# A última linha contém valores em atraso que devem ser desprezados
apaga_7(counter_aprox, workbook['Aprox_TCP_pose'])
apaga_7(counter_aprox, workbook['Aprox_TCP_force'])

#Cálculo da distância percorrida
VecDist = [0]
n_elem = 0
for i in range(1,len(insp_tempo)):
    if insp_tcp_pose_x[i] == 0:
        break
    dist_x = insp_tcp_pose_x[i]-insp_tcp_pose_x[0]
    dist_y = insp_tcp_pose_y[i]-insp_tcp_pose_y[0]
    dist_z = insp_tcp_pose_z[i]-insp_tcp_pose_z[0]
    dist = (dist_x**2+dist_y**2+dist_z**2)**(1/2)
    VecDist.append(dist)
    n_elem = n_elem +1
```

```
worksheet = workbook['Inspec_TCP_pose']
for i in range(len(insp_tempo)):
    if insp_tcp_pose_x[i] == 0:
        break
    worksheet.cell(i + 2, 1).value = insp_tempo[i]
    worksheet.cell(i + 2, 2).value = VecDist[i]
    worksheet.cell(i + 2, 3).value = insp_tcp_pose_x[i]
    worksheet.cell(i + 2, 4).value = insp_tcp_pose_y[i]
    worksheet.cell(i + 2, 5).value = insp_tcp_pose_z[i]
    worksheet.cell(i + 2, 6).value = insp_tcp_pose_rx[i]
    worksheet.cell(i + 2, 7).value = insp_tcp_pose_ry[i]
    worksheet.cell(i + 2, 8).value = insp_tcp_pose_rz[i]
    worksheet.cell(i + 2, 9).value = digi_ch1[i]
    worksheet.cell(i + 2, 10).value = digi_ch2[i]

worksheet = workbook['Inspec_TCP_force']
for i in range(len(insp_tempo)):
    if insp_tcp_pose_x[i] == 0:
        break
    worksheet.cell(i + 2, 1).value = insp_tempo[i]
    worksheet.cell(i + 2, 2).value = VecDist[i]
    worksheet.cell(i + 2, 3).value = insp_tcp_force_x[i]
    worksheet.cell(i + 2, 4).value = insp_tcp_force_y[i]
    worksheet.cell(i + 2, 5).value = insp_tcp_force_z[i]
    worksheet.cell(i + 2, 6).value = insp_tcp_force_rx[i]
    worksheet.cell(i + 2, 7).value = insp_tcp_force_ry[i]
    worksheet.cell(i + 2, 8).value = insp_tcp_force_rz[i]

# ----------Gravação dos dados
workbook.save(f"{save_dir}{formatted_datetime}.xlsx")
print('Robot data is accessible
      at: ', f"{save_dir}{formatted_datetime}.xlsx\n")
feedback_box.insert(tk.END, f"Robot data is accessible
                    at: {save_dir}{formatted_datetime}.xlsx\n")
#O programa terminou, logo já se deve poder correr outro:
run_button.config(state='normal')
```

```
"""
    #Apresenta um gráfico
    VecDist, digi_ch2 = zip(*zip(VecDist, digi_ch2)) #garante que os elementos em excesso
    plt.plot(VecDist, digi_ch2)
    plt.xlabel("Distância percorrida [m]")
    plt.ylabel("Tensão [V]")
    plt.show()
    #Há um erro relacionado com o facto de o plot ser chamado fora da main thread
"""

# Create the main window
window = tk.Tk()
window.title("Interface")
window.geometry("700x300")

# IP input
ip_label = tk.Label(window, text="IP:")
ip_label.grid(row=0, column=0, sticky=tk.E)
ip_entry = tk.Entry(window)
ip_entry.insert(0, "192.168.1.10")
ip_entry.grid(row=0, column=1, padx=10, pady=5)

# Port input
port_label = tk.Label(window, text="Port:")
port_label.grid(row=1, column=0, sticky=tk.E)
port_entry = tk.Entry(window)
port_entry.insert(0, "30004")
port_entry.grid(row=1, column=1, padx=10, pady=5)

# Connect button
connect_button = tk.Button(window, text="Connect", command=connect)
connect_button.grid(row=2, column=1, columnspan=2,padx=10, pady=10, sticky=tk.W)

#Programas no dropdown menu
programs = ['get_var']

# Dropdown menu
program_var = tk.StringVar(window)
program_var.set(programs[0])  # Set the default program
program_dropdown = tk.OptionMenu(window, program_var, *programs)
program_dropdown.grid(row=0, column=2, padx=10, pady=5)
```

```
# Run button
run_button = tk.Button(window, text="Run", command=run_button_clicked, state='disabled')
run_button.grid(row=0, column=3, padx=10, pady=5, sticky=tk.W)

#Stop button
#Cria o botão e define a função que chama
stop_button = tk.Button(window, text="Stop", command=stop_button_clicked,
                        state='disabled')
#Define a posição do botão
stop_button.grid(row=0, column=4, padx=10, pady=5, sticky=tk.W)

# Send input
#send_label = tk.Label(window, text="Send:")
#send_label.grid(row=1, column=2, sticky=tk.E)
send_entry = tk.Entry(window)
send_entry.grid(row=1, column=2, padx=10, pady=5)

# Send button
send_button = tk.Button(window, text="Send", command=send, state='disabled')
send_button.grid(row=1, column=3, pady=5)

# Feedback box
feedback_label = tk.Label(window, text="Feedback:")
feedback_label.grid(row=3, column=0, padx=10, sticky=tk.W)
feedback_box = AutoScrollText(window, width=30, height=10)
feedback_box.grid(row=4, column=0, columnspan=5, padx=10, pady=5,
                  sticky=tk.N+tk.S+tk.E+tk.W)

# Power on
#Cria o botão e define a função a chamar
pwr_on_button = tk.Button(window, text="Power on",
                          command=pwr_on_button_clicked,state='disabled')
#Define a posição do botão
pwr_on_button.grid(row=0, column=5, padx=10, pady=5)

#Brake release
brake_button = tk.Button(window, text="Brake release", command=brake_button_clicked,
                         state='disabled')
brake_button.grid(row=1, column=5, padx=10, pady=5)
```

```
#Power off
pwr_off_button = tk.Button(window, text="Power off",
                           command=pwr_off_button_clicked,state='disabled')
pwr_off_button.grid(row=2, column=5, padx=10, pady=5)

#Play
play_button = tk.Button(window, text="Play", command=play_button_clicked)
play_button.grid(row=2, column=3, padx=10, pady=5)

#Close popup
close_popup_button = tk.Button(window, text="Close popup",
                               command=close_popup_button_clicked)
close_popup_button.grid(row=2, column=4, padx=10, pady=5, sticky=tk.W)

window.grid_columnconfigure(0, weight=0)
window.grid_columnconfigure(4, weight=1)
window.grid_rowconfigure(4, weight=1)

feedback_box.insert(tk.END,'Insert robot data and click Connect\n')
feedback_box.insert(tk.END,'RTDE Port = 30004\n')

window.mainloop()
```

III

Programa de inspeção .urp

```
def inspec_get_var():
  set_target_payload(
    1.430000, [0.002000, -0.003000, 0.043000],
    [0.002794, 0.002794, 0.002794, 0.000000, 0.000000, 0.000000]
  )
  set_gravity([0.0, 0.0, 9.82])
  step_count_4488afd7_e9ea_47b4_ad0f_6727377e87ce = 0.0
  thread Step_Counter_Thread_82b731ac_4dd5_480a_b92c_5476013904c7():
    while (True):
      step_count_4488afd7_e9ea_47b4_ad0f_6727377e87ce = (
        step_count_4488afd7_e9ea_47b4_ad0f_6727377e87ce + 1.0
      )
      sync()
    end
  end
  run Step_Counter_Thread_82b731ac_4dd5_480a_b92c_5476013904c7()
  set_tcp(p[0.0,0.0,0.21,0.0,0.0,0.0])
  set_safety_mode_transition_hardness(1)
  set_standard_analog_input_domain(0, 1)
  set_standard_analog_input_domain(1, 1)
  set_tool_analog_input_domain(0, 1)
  set_tool_analog_input_domain(1, 1)
  set_analog_outputdomain(0, 0)
  set_analog_outputdomain(1, 0)
  set_input_actions_to_default()
  set_tool_communication(True, 1000000, 2, 1, 1.5, 3.5)
  set_tool_output_mode(0)
  set_tool_digital_output_mode(0, 1)
  set_tool_digital_output_mode(1, 1)
```

```
set_tool_voltage(24)
global ft_inst=p[3.3261, 0.57908, -78.2113, -0.34406, 2.65704, -0.81426]
global contador=76023
global Plane_1=p[
  0.22262826040404676,
  0.5953938626185933,
  -0.02056367250718036,
  8.431909629236707E-4,
  3.00806554123457E-4,
  -3.1410113432837923
]
global Point_1=p[
  -0.5824806528994685,
  -0.3861541906511535,
  0.6546023535429051,
  1.7490384516664392,
  0.6754774208257791,
  -1.8085408755860224
]
global Point_2=p[
  -0.5825004381563887,
  -0.38614321591467293,
  0.7993227850703662,
  1.7491267445013594,
  0.6754853233000239,
  -1.8085152040566532
]
# begin: URCap Installation Node
#   Source: Remote TCP & Toolpath, 1.3.0.build15, Universal Robots A/S
#   Type: Remote TCP & Toolpath
# end: URCap Installation Node
global contacto=  False
global fase=0
global ini=  False
global inicio=  False
global T_rz1=0
global T_stable=0
global timer_aprox=0
global timer_inspec=0
global timer_x=0
```

```
global Waypoint_3_p=p[
  .683589677315,
  -.699182485083,
  .254168362429,
  2.338009235995,
  -2.098407440645,
  -.000157816718
]
global Waypoint_3_q=[
  -0.6184094587909144,
  -2.061794420281881,
  -1.3292608261108398,
  -1.3208249372294922,
  1.572402834892273,
  -0.7260788122760218
]
global Waypoint_26_p=p[
  .683595452670,
  -.699190651877,
  .184552386232,
  -2.337938447533,
  2.098391255290,
  -.000095516130
]
global Waypoint_26_q=[
  -0.6184261480914515,
  -2.096442838708395,
  -1.4034295082092285,
  -1.2118401390365143,
  1.5723633766174316,
  -0.725933853779928
]
global Teste_tese_1_p=p[
  .701198081387,
  -.651527993585,
  .177822587132,
  2.104136229106,
  -2.305337036118,
  .047310743949
]
```

```
global Teste_tese_1_q=[
  -0.5569432417498987,
  -2.055892606774801,
  -1.4850561618804932,
  -1.1470733147910614,
  1.5448063611984253,
  -0.465705696736471
]
def calculate_point_to_move_towards(feature, direction, position_distance):
  local posDir=[direction[0], direction[1], direction[2]]
  if (norm(posDir) < 1e-6):
    return get_target_waypoint()
  end
  local direction_vector_normalized=normalize(posDir)
  local displacement_pose=p[
    direction_vector_normalized[0] * position_distance,
    direction_vector_normalized[1] * position_distance,
    direction_vector_normalized[2] * position_distance,
    0, 0, 0
  ]
  local wanted_displacement_in_base_frame=pose_sub(
    pose_trans(feature, displacement_pose), feature
  )
  return pose_add(get_target_waypoint(), wanted_displacement_in_base_frame)
end
global Waypoint_1_p=p[
  .549919762804,
  -.699249303299,
  .168155265163,
  2.306026626434,
  -2.133459871530,
  -.000136518545
]
global Waypoint_1_q=[
  -0.7079408804522913,
  -1.971997400323385,
  -1.6239714622497559,
  -1.115899221306183,
  1.572313666343689,
  -0.7853859106646937
```

```
]
global timer_x_is_counting=False
global T_rz1_is_counting=False
global timer_inspec_is_counting=False
global T_stable_is_counting=False
global timer_aprox_is_counting=False
thread Timer_Thread():
  while (True):
    if (timer_x_is_counting):
      timer_x = timer_x + get_steptime()
    end
    if (T_rz1_is_counting):
      T_rz1 = T_rz1 + get_steptime()
    end
    if (timer_inspec_is_counting):
      timer_inspec = timer_inspec + get_steptime()
    end
    if (T_stable_is_counting):
      T_stable = T_stable + get_steptime()
    end
    if (timer_aprox_is_counting):
      timer_aprox = timer_aprox + get_steptime()
    end
    sync()
  end
end
run Timer_Thread()
def exp_force_mode_ajustaX_su():
  $ 104 "exp_force_mode_ajustaX_su" "noBreak"
  $ 105 "Script: baseframe-toolframe.txt.script"
  def get_tcp_force_tool():
      force_torque = get_tcp_force()
      force_B = p[
            force_torque[0],
            force_torque[1],
            force_torque[2],
            0, 0, 0
          ]
      torque_B = p[
            force_torque[3],
```

```
            force_torque[4],
            force_torque[5],
            0, 0, 0
          ]
      tcp = get_actual_tcp_pose()
      rotation_BT = p[ 0, 0, 0, tcp[3], tcp[4], tcp[5] ]
      force_T = pose_trans( pose_inv(rotation_BT), force_B )
      torque_T = pose_trans( pose_inv(rotation_BT), torque_B )
      force_torque_T = p[
            force_T[0],
            force_T[1],
            force_T[2],
            torque_T[0],
            torque_T[1],
            torque_T[2]
          ]
      return force_torque_T
end
$ 106 "Set Payload: Payload_ADAM_1"
set_target_payload(
  1.430000,
  [0.002000, -0.003000, 0.043000],
  [0.002794, 0.002794, 0.002794, 0.000000, 0.000000, 0.000000]
)
$ 107 "p_final:=p"
global p_final=p
$ 108 "MoveL"
$ 111 "Direction: Tool Z+"
global move_thread_flag_111=0
thread move_thread_111():
  enter_critical
  move_thread_flag_111 = 1
  local towardsPos=calculate_point_to_move_towards(
    get_forward_kin(),
    [0.0,0.0,1.0],
    1000.0
  )
  movel(towardsPos, a=1.2, v=0.005)
  move_thread_flag_111 = 2
  exit_critical
```

```
end
move_thread_flag_111 = 0
move_thread_han_111 = run move_thread_111()
while (True):
  local targetTcpDirection=get_target_tcp_speed()
  local stepsToRetract=tool_contact(direction=targetTcpDirection)
  if (stepsToRetract > 0):
    kill move_thread_han_111
    stopl(3.0)
    local backTrackMovement=get_actual_joint_positions_history(
      stepsToRetract
    )
    local contactPose=get_forward_kin(backTrackMovement)
    local posDir=[
      targetTcpDirection[0],
      targetTcpDirection[1],
      targetTcpDirection[2]
    ]
    local retractTo=contactPose
    if (norm(posDir) > 1e-6):
      local normalizedPosDir=normalize(posDir)
      local additionalRetraction=p[
        normalizedPosDir[0] * 0.0,
        normalizedPosDir[1] * 0.0,
        normalizedPosDir[2] * 0.0,
        0, 0, 0
      ]
      retractTo = pose_sub(contactPose, additionalRetraction)
    end
    movel(retractTo, a=3.0, v=0.1)
    $ 112 "Until (tool_contact_detection)"
    break
  end
  sync()
end
$ 113 "Wait: 1.0"
sleep(1.0)
$ 114 "timer_x: Start"
timer_x_is_counting = True
$ 115 "force_mode(
```

```
    tool_pose(),
    [0,0,1,0,0,0],
    [0,0,0,0,0,0],
    2,
    [0.5,0.003,0.05,0.05,0.05,0.05]
  )"
  force_mode(tool_pose(),
    [0,0,1,0,0,0],
    [0,0,0,0,0,0],
    2,
    [0.5,0.003,0.05,0.05,0.05,0.05]
  )
  $ 116 "Direction: Tool X+"
  global move_thread_flag_116=0
  thread move_thread_116():
    enter_critical
    move_thread_flag_116 = 1
    local towardsPos=calculate_point_to_move_towards(
      get_forward_kin(),
      [1.0,0.0,0.0],
      1000.0
    )
    movel(towardsPos, a=1.2, v=0.02)
    move_thread_flag_116 = 2
    exit_critical
  end
  move_thread_flag_116 = 0
  move_thread_han_116 = run move_thread_116()
  while (True):
    if (norm(speed[2])>0.02):
      kill move_thread_han_116
      stopl(1.2)
      $ 117 "Until (expression)"
      break
    end
    sync()
  end
  $ 118 "end_force_mode()"
  end_force_mode()
  $ 119 "timer_x: Stop"
```

```
timer_x_is_counting = False
$ 120 "t_d_x:=timer_x"
global t_d_x=timer_x
$ 121 "timer_x: Reset"
timer_x = 0
$ 122 "'Atualizar com a velocidade'"
# 'Atualizar com a velocidade'
$ 123 "tcp_speed:=0.02"
global tcp_speed=0.02
$ 124 "d_x:=tcp_speed*t_d_x-0.078-0.015"
global d_x=tcp_speed*t_d_x-0.078-0.015
$ 125 "t_x:=d_x/tcp_speed"
global t_x=d_x/tcp_speed
$ 126 "MoveL"
$ 127 "Direction: Tool Z-"
global move_thread_flag_127=0
thread move_thread_127():
  enter_critical
  move_thread_flag_127 = 1
  local towardsPos=calculate_point_to_move_towards(
    get_forward_kin(),
    [0.0,0.0,-1.0],
    0.005
  )
  movel(towardsPos, a=1.2, v=0.1)
  move_thread_flag_127 = 2
  exit_critical
end
move_thread_flag_127 = 0
move_thread_han_127 = run move_thread_127()
while (True):
  sleep(1.0E-10)
  if (move_thread_flag_127 > 1):
    join move_thread_han_127
    $ 128 "Until (distance)"
    break
  end
  sync()
end
$ 129 "Direction: Tool X-"
```

```
global move_thread_flag_129=0
thread move_thread_129():
  enter_critical
  move_thread_flag_129 = 1
  local towardsPos=calculate_point_to_move_towards(
    get_forward_kin(),
    [-1.0,0.0,0.0],
    0.08
  )
  movel(towardsPos, a=1.2, v=0.1)
  move_thread_flag_129 = 2
  exit_critical
end
move_thread_flag_129 = 0
move_thread_han_129 = run move_thread_129()
while (True):
  sleep(1.0E-10)
  if (move_thread_flag_129 > 1):
    join move_thread_han_129
    $ 130 "Until (distance)"
    break
  end
  sync()
end
$ 131 "MoveL"
$ 132 "p_final" "breakAfter"
movel(p_final, a=1.2, v=0.05)
$ 133 "If d_x≥0"
if (d_x >= 0):
  $ 134 "timer_x: Start"
  timer_x_is_counting = True
  $ 135 "MoveL"
  $ 136 "Direction: Tool X+"
  global move_thread_flag_136=0
  thread move_thread_136():
    enter_critical
    move_thread_flag_136 = 1
    local towardsPos=calculate_point_to_move_towards(
      get_forward_kin(),
      [1.0,0.0,0.0],
```

```
      1000.0
    )
    movel(towardsPos, a=1.2, v=0.02)
    move_thread_flag_136 = 2
    exit_critical
  end
  move_thread_flag_136 = 0
  move_thread_han_136 = run move_thread_136()
  while (True):
    if (timer_x >= t_x):
      kill move_thread_han_136
      stopl(1.2)
      $ 137 "Until (expression)"
      break
    end
    sync()
  end
  $ 138 "timer_x: Stop"
  timer_x_is_counting = False
  $ 139 "timer_x: Reset"
  timer_x = 0
else:
  $ 140 "Else" "noBreak"
  $ 141 "timer_x: Start"
  timer_x_is_counting = True
  $ 142 "MoveL"
  $ 143 "Direction: Tool X-"
  global move_thread_flag_143=0
  thread move_thread_143():
    enter_critical
    move_thread_flag_143 = 1
    local towardsPos=calculate_point_to_move_towards(
      get_forward_kin(),
      [-1.0,0.0,0.0],
      1000.0
    )
    movel(towardsPos, a=1.2, v=0.02)
    move_thread_flag_143 = 2
    exit_critical
  end
```

```
      move_thread_flag_143 = 0
      move_thread_han_143 = run move_thread_143()
      while (True):
        if (timer_x >= t_x):
          kill move_thread_han_143
          stopl(1.2)
          $ 144 "Until (expression)"
          break
        end
        sync()
      end
      $ 145 "timer_x: Stop"
      timer_x_is_counting = False
      $ 146 "timer_x: Reset"
      timer_x = 0
    end
  end
  def exp_force_mode_caminho3_1():
    $ 147 "exp_force_mode_caminho3_1" "noBreak"
    $ 148 "Script: baseframe-toolframe.txt.script"
    def get_tcp_force_tool():
        force_torque = get_tcp_force()
        force_B = p[
              force_torque[0], force_torque[1], force_torque[2],
              0, 0, 0
            ]
        torque_B = p[
              force_torque[3], force_torque[4], force_torque[5],
              0, 0, 0
            ]
        tcp = get_actual_tcp_pose()
        rotation_BT = p[ 0, 0, 0, tcp[3], tcp[4], tcp[5] ]
        force_T = pose_trans( pose_inv(rotation_BT), force_B )
        torque_T = pose_trans( pose_inv(rotation_BT), torque_B )
        force_torque_T = p[
              force_T[0], force_T[1], force_T[2],
              torque_T[0], torque_T[1], torque_T[2]
            ]
        return force_torque_T
    end
```

```
$ 149 "Set Payload: Payload_ADAM_1"
set_target_payload(
  1.430000,
  [0.002000, -0.003000, 0.043000],
  [0.002794, 0.002794, 0.002794, 0.000000, 0.000000, 0.000000]
)
$ 150 "Wait fase=?2"
while (not(fase == 2)):
  sync()
end
$ 151 "Wait: 2.0"
sleep(2.0)
$ 152 "p_ini:=p"
global p_ini=p
$ 153 "dp2:=p[0.0019,0.0786,0.003,0,0.011,0.004]"
global dp2=p[0.0019,0.0786,0.003,0,0.011,0.004]
$ 154 "dp3:=p[0.0040,0.1568,0.0063,0,0.025,0.005]"
global dp3=p[0.0040,0.1568,0.0063,0,0.025,0.005]
$ 155 "dp4:=p[0.0067,0.2327,0.0078,0,0.034,0.011]"
global dp4=p[0.0067,0.2327,0.0078,0,0.034,0.011]
$ 156 "dp5:=p[0.0094,0.3294,0.0089,0,0.046,0.006]"
global dp5=p[0.0094,0.3294,0.0089,0,0.046,0.006]
$ 157 "dp6:=p[0.0120,0.4234,0.0089,0,0.056,0.007]"
global dp6=p[0.0120,0.4234,0.0089,0,0.056,0.007]
$ 158 "dp7:=p[0.0140,0.5130,0.0081,0,0.067,0.007]"
global dp7=p[0.0140,0.5130,0.0081,0,0.067,0.007]
$ 159 "dp8:=p[0.0168,0.6171,0.0062,0,0.078,0.006]"
global dp8=p[0.0168,0.6171,0.0062,0,0.078,0.006]
$ 160 "dp9:=p[0.0202,0.7553,0.0012,0.016,0.094,-0.004]"
global dp9=p[0.0202,0.7553,0.0012,0.016,0.094,-0.004]
$ 161 "dp10:=p[0.0224,0.8685,-0.0045,0.022,0.115,0]"
global dp10=p[0.0224,0.8685,-0.0045,0.022,0.115,0]
$ 162 "dp11:=p[0.0246,0.9726,-0.0118,0.017,0.135,-0.012]"
global dp11=p[0.0246,0.9726,-0.0118,0.017,0.135,-0.012]
$ 163 "dp12:=p[0.0255,1.0248,-0.0164,0.017,0.143,-0.007]"
global dp12=p[0.0255,1.0248,-0.0164,0.017,0.143,-0.007]
$ 164 "dp13:=p[0.0263,1.0780,-0.0216,0.018,0.156,-0.010]"
global dp13=p[0.0263,1.0780,-0.0216,0.018,0.156,-0.010]
$ 165 "dp14:=p[0.0261,1.1331,-0.0281,0.018,0.176,-0.023]"
global dp14=p[0.0261,1.1331,-0.0281,0.018,0.176,-0.023]
```

```
$ 166 "dp15:=p[0.0260,1.1783,-0.0341,0.018,0.197,-0.030]"
global dp15=p[0.0260,1.1783,-0.0341,0.018,0.197,-0.030]
$ 167 "dp16:=p[0.0262,1.2036,-0.0378,0.018,0.208,-0.031]"
global dp16=p[0.0262,1.2036,-0.0378,0.018,0.208,-0.031]
$ 168 "dp17:=p[0.0258,1.2309,-0.0422,0.018,0.222,-0.032]"
global dp17=p[0.0258,1.2309,-0.0422,0.018,0.222,-0.032]
$ 169 "dp18:=p[0.0252,1.2607,-0.0476,0.020,0.240,-0.035]"
global dp18=p[0.0252,1.2607,-0.0476,0.020,0.240,-0.035]
$ 170 "dp19:=p[0.0244,1.2981,-0.0552,0.021,0.266,-0.051]"
global dp19=p[0.0244,1.2981,-0.0552,0.021,0.266,-0.051]
$ 171 "dp20:=p[0.0237,1.3327,-0.0630,0.022,0.295,-0.061]"
global dp20=p[0.0237,1.3327,-0.0630,0.022,0.295,-0.061]
$ 172 "dp21:=p[0.0223,1.3649,-0.0713,0.023,0.324,-0.069]"
global dp21=p[0.0223,1.3649,-0.0713,0.023,0.324,-0.069]
$ 173 "dp22:=p[0.0202,1.3911,-0.0793,0.025,0.353,-0.088]"
global dp22=p[0.0202,1.3911,-0.0793,0.025,0.353,-0.088]
$ 174 "dp23:=p[0.0185,1.4175,-0.0877,0.029,0.380,-0.105]"
global dp23=p[0.0185,1.4175,-0.0877,0.029,0.380,-0.105]
$ 175 "dp24:=p[0.0172,1.4379,-0.095,0.031,0.403,-0.109]"
global dp24=p[0.0172,1.4379,-0.095,0.031,0.403,-0.109]
$ 176 "var_1:=pose_add(p_ini, dp2)"
global var_1= pose_add (p_ini, dp2)
$ 177 "var_2:=pose_add(p_ini, dp3)"
global var_2= pose_add (p_ini, dp3)
$ 178 "var_3:=pose_add(p_ini, dp4)"
global var_3= pose_add (p_ini, dp4)
$ 179 "var_4:=pose_add(p_ini, dp5)"
global var_4= pose_add (p_ini, dp5)
$ 180 "var_5:=pose_add(p_ini, dp6)"
global var_5= pose_add (p_ini, dp6)
$ 181 "var_6:=pose_add(p_ini, dp7)"
global var_6= pose_add (p_ini, dp7)
$ 182 "var_7:=pose_add(p_ini, dp8)"
global var_7= pose_add (p_ini, dp8)
$ 183 "var_8:=pose_add(p_ini, dp9)"
global var_8= pose_add (p_ini, dp9)
$ 184 "var_9:=pose_add(p_ini, dp10)"
global var_9= pose_add (p_ini, dp10)
$ 185 "var_10:=pose_add(p_ini, dp11)"
global var_10= pose_add (p_ini, dp11)
```

```
$ 186 "var_11:=pose_add(p_ini, dp12)"
global var_11= pose_add (p_ini, dp12)
$ 187 "var_12:=pose_add(p_ini, dp13)"
global var_12= pose_add (p_ini, dp13)
$ 188 "var_13:=pose_add(p_ini, dp14)"
global var_13= pose_add (p_ini, dp14)
$ 189 "var_14:=pose_add(p_ini, dp15)"
global var_14= pose_add (p_ini, dp15)
$ 190 "var_15:=pose_add(p_ini, dp16)"
global var_15= pose_add (p_ini, dp16)
$ 191 "var_16:=pose_add(p_ini, dp17)"
global var_16= pose_add (p_ini, dp17)
$ 192 "var_17:=pose_add(p_ini, dp18)"
global var_17= pose_add (p_ini, dp18)
$ 193 "var_18:=pose_add(p_ini, dp19)"
global var_18= pose_add (p_ini, dp19)
$ 194 "var_19:=pose_add(p_ini, dp20)"
global var_19= pose_add (p_ini, dp20)
$ 195 "var_20:=pose_add(p_ini, dp21)"
global var_20= pose_add (p_ini, dp21)
$ 196 "var_21:=pose_add(p_ini, dp22)"
global var_21= pose_add (p_ini, dp22)
$ 197 "var_22:=pose_add(p_ini, dp23)"
global var_22= pose_add (p_ini, dp23)
$ 198 "var_23:=pose_add(p_ini, dp24)"
global var_23= pose_add (p_ini, dp24)
$ 199 "force_mode(
  tool_pose(),
  [0,1,1,0,0,0],
  [0,3,15,0,0,0],
  2,
  [0.5,0.003,0.05,0.05,0.05,0.05]
)"
force_mode(
  tool_pose(),
  [0,1,1,0,0,0],
  [0,3,15,0,0,0],
  2,
  [0.5,0.003,0.05,0.05,0.05,0.05]
)
```

```
$ 201 "ini:= True "
global ini=  True
$ 202 "write_output_boolean_register(64, False )"
write_output_boolean_register(64,  False  )
$ 203 "MoveL"
$ 204 "var_1" "breakAfter"
movel(var_1, a=1.2, v=0.06, r=0.005)
$ 205 "var_2" "breakAfter"
movel(var_2, a=1.2, v=0.06, r=0.005)
$ 206 "var_3" "breakAfter"
movel(var_3, a=1.2, v=0.06, r=0.005)
$ 207 "var_4" "breakAfter"
movel(var_4, a=1.2, v=0.06, r=0.005)
$ 208 "var_5" "breakAfter"
movel(var_5, a=1.2, v=0.06, r=0.005)
$ 209 "var_6" "breakAfter"
movel(var_6, a=1.2, v=0.06, r=0.005)
$ 210 "var_7" "breakAfter"
movel(var_7, a=1.2, v=0.06, r=0.005)
$ 211 "force_mode(
  tool_pose(),
  [0,1,1,0,0,0],
  [0,2,15,0,0,0],
  2,
  [0.5,0.002,0.05,0.05,0.05,0.05]
)"
force_mode(
  tool_pose(),
  [0,1,1,0,0,0],
  [0,2,15,0,0,0],
  2,
  [0.5,0.002,0.05,0.05,0.05,0.05]
)
$ 212 "MoveL"
$ 213 "var_8" "breakAfter"
movel(var_8, a=1.2, v=0.06, r=0.005)
$ 214 "var_9" "breakAfter"
movel(var_9, a=1.2, v=0.06, r=0.005)
$ 215 "var_10" "breakAfter"
movel(var_10, a=1.2, v=0.06, r=0.005)
```

```
$ 216 "var_11" "breakAfter"
movel(var_11, a=1.2, v=0.06, r=0.005)
$ 217 "var_12" "breakAfter"
movel(var_12, a=1.2, v=0.06, r=0.005)
$ 218 "force_mode(
  tool_pose(),
  [0,1,1,0,0,0],
  [0,3,15,0,0,0],
  2,
  [0.5,0.003,0.05,0.05,0.05,0.05]
)"
force_mode(
  tool_pose(),
  [0,1,1,0,0,0],
  [0,3,15,0,0,0],
  2,
  [0.5,0.003,0.05,0.05,0.05,0.05]
)
$ 219 "MoveL"
$ 220 "var_13" "breakAfter"
movel(var_13, a=1.2, v=0.06, r=0.005)
$ 221 "var_14" "breakAfter"
movel(var_14, a=1.2, v=0.06, r=0.005)
$ 222 "var_15" "breakAfter"
movel(var_15, a=1.2, v=0.06, r=0.005)
$ 223 "var_16" "breakAfter"
movel(var_16, a=1.2, v=0.06, r=0.005)
$ 224 "var_17" "breakAfter"
movel(var_17, a=1.2, v=0.06, r=0.005)
$ 225 "var_18" "breakAfter"
movel(var_18, a=1.2, v=0.06, r=0.005)
$ 226 "var_19" "breakAfter"
movel(var_19, a=1.2, v=0.06, r=0.005)
$ 227 "var_20" "breakAfter"
movel(var_20, a=1.2, v=0.06, r=0.005)
$ 228 "var_21" "breakAfter"
movel(var_21, a=1.2, v=0.06, r=0.005)
$ 229 "var_22" "breakAfter"
movel(var_22, a=1.2, v=0.06, r=0.005)
$ 230 "var_23" "breakAfter"
```

```
movel(var_23, a=1.2, v=0.06)
$ 231 "end_force_mode()"
end_force_mode()
$ 232 "write_output_boolean_register(0, True )"
write_output_boolean_register(0,  True  )
$ 233 "ini:= False "
global ini=  False
$ 234 "Wait: 2.0"
sleep(2.0)
$ 235 "force_mode_set_gain_scaling(1)"
force_mode_set_gain_scaling(1)
$ 237 "MoveL"
$ 238 "Direction: Base Z+"
global move_thread_flag_238=0
thread move_thread_238():
  enter_critical
  move_thread_flag_238 = 1
  local towardsPos=calculate_point_to_move_towards(
    p[0.0,0.0,0.0,0.0,0.0,0.0],
    [0.0,0.0,1.0],
    0.2
  )
  movel(towardsPos, a=1.2, v=0.15)
  move_thread_flag_238 = 2
  exit_critical
end
move_thread_flag_238 = 0
move_thread_han_238 = run move_thread_238()
while (True):
  sleep(1.0E-10)
  if (move_thread_flag_238 > 1):
    join move_thread_han_238
    $ 239 "Until (distance)"
    break
  end
  sync()
end
$ 240 "MoveJ"
$ 241 "Waypoint_3" "breakAfter"
movej(get_inverse_kin(Waypoint_3_p, qnear=Waypoint_3_q),
```

```
    a=1.3962634015954636, v=0.5235987755982988
  )
end
$ 243 "Thread_1"
thread Thread_1():
  while (True):
    $ 244 "sync()"
    sync()
    $ 245 "ft:=get_tcp_force_tool()"
    global ft=get_tcp_force_tool()
    $ 246 "write_output_float_register(24,ft[0])"
    write_output_float_register(24,ft[0])
    $ 247 "write_output_float_register(25,ft[1])"
    write_output_float_register(25,ft[1])
    $ 248 "write_output_float_register(26,ft[2])"
    write_output_float_register(26,ft[2])
    $ 249 "write_output_float_register(27,ft[3])"
    write_output_float_register(27,ft[3])
    $ 250 "write_output_float_register(28,ft[4])"
    write_output_float_register(28,ft[4])
    $ 251 "write_output_float_register(29,ft[5])"
    write_output_float_register(29,ft[5])
    $ 252 "p:=get_actual_tcp_pose()"
    global p=get_actual_tcp_pose()
    $ 253 "p_inv:=get_inverse_kin(p)"
    global p_inv=get_inverse_kin(p)
    $ 254 "speed:=get_actual_tcp_speed()"
    global speed=get_actual_tcp_speed()
  end
end
threadId_Thread_1 = run Thread_1()
$ 255 "Thread_2"
thread Thread_2():
  while (True):
    $ 256 "Loop inicio≟ True "
    thread Thread_while_256():
      while (True):
        $ 257 "sync()"
        sync()
        $ 258 "fase:=read_input_integer_register(0)"
```

```
          global fase=read_input_integer_register(0)
          $ 259 "stop:=read_input_integer_register(1)"
          global stop=read_input_integer_register(1)
          $ 260 "If stop≟1"
          if (stop == 1):
            $ 261 "Popup: KeyboardInterrupt registered"
            popup("KeyboardInterrupt registered", "Message",
              False, False, blocking=False
            )
            halt
            $ 262 "Halt"
            halt
          end
        end
      end
      if (inicio ==  True  ):
        global thread_handler_256=run Thread_while_256()
        while (inicio ==  True  ):
          sync()
        end
        kill thread_handler_256
      end
    end
  end
  threadId_Thread_2 = run Thread_2()
  $ 263 "Thread_3"
  thread Thread_3():
    while (True):
      $ 264 "sync()"
      sync()
      $ 265 "Wait contacto≟ True "
      while (not(contacto ==  True  )):
        sync()
      end
      $ 266 "timer_aprox: Start"
      timer_aprox_is_counting = True
      $ 267 "Loop contacto≟ True "
      thread Thread_while_267():
        while (True):
          $ 268 "sync()"
```

```
        sync()
        $ 269 "write_output_float_register(30,timer_aprox)"
        write_output_float_register(30,timer_aprox)
      end
    end
    if (contacto ==  True  ):
      global thread_handler_267=run Thread_while_267()
      while (contacto ==  True  ):
        sync()
      end
      kill thread_handler_267
    end
    $ 270 "timer_aprox: Stop"
    timer_aprox_is_counting = False
    $ 271 "timer_aprox: Reset"
    timer_aprox = 0
    $ 272 "write_output_float_register(30,0)"
    write_output_float_register(30,0)
    $ 273 "Wait ini≟ True "
    while (not(ini ==  True  )):
      sync()
    end
    $ 274 "timer_inspec: Start"
    timer_inspec_is_counting = True
    $ 275 "Loop ini≟ True "
    while (ini ==  True  ):
      $ 276 "sync()"
      sync()
      $ 277 "write_output_float_register(30,timer_inspec)"
      write_output_float_register(30,timer_inspec)
    end
    $ 278 "timer_inspec: Stop"
    timer_inspec_is_counting = False
    $ 279 "timer_inspec: Reset"
    timer_inspec = 0
  end
end
threadId_Thread_3 = run Thread_3()
while (True):
  $ 2 "Robot Program"
```

```
$ 3 "Set Payload: Payload_ADAM_1"
set_target_payload(
  1.430000,
  [0.002000, -0.003000, 0.043000],
  [0.002794, 0.002794, 0.002794, 0.000000, 0.000000, 0.000000]
)
$ 4 "write_output_boolean_register(0, False )"
write_output_boolean_register(0,  False  )
$ 5 "write_output_boolean_register(64, False )"
write_output_boolean_register(64,  False  )
$ 6 "write_output_float_register(30,0)"
write_output_float_register(30,0)
$ 7 "Script: baseframe-toolframe.txt.script"
def get_tcp_force_tool():
    force_torque = get_tcp_force()
    force_B = p[
          force_torque[0], force_torque[1], force_torque[2],
          0, 0, 0
        ]
    torque_B = p[
          force_torque[3], force_torque[4], force_torque[5],
          0, 0, 0
        ]
    tcp = get_actual_tcp_pose()
    rotation_BT = p[ 0, 0, 0, tcp[3], tcp[4], tcp[5] ]
    force_T = pose_trans( pose_inv(rotation_BT), force_B )
    torque_T = pose_trans( pose_inv(rotation_BT), torque_B )
    force_torque_T = p[
          force_T[0], force_T[1], force_T[2],
          torque_T[0], torque_T[1], torque_T[2]
        ]
    return force_torque_T
end
$ 8 "Wait: 0.01"
sleep(0.01)
$ 9 "
  Popup: Correr o programa get_pose.py e depois carregar em 'Continue'
"
popup("Correr o programa get_pose.py e depois carregar em 'Continue'",
  "Message", False, False, blocking=True
```

```
)
$ 10 "inicio:= True "
global inicio= True
$ 11 "write_output_boolean_register(0, True )"
write_output_boolean_register(0, True )
$ 12 "zero_ftsensor()"
zero_ftsensor()
$ 13 "Wait fase?=1"
while (not(fase == 1)):
  sync()
end
$ 14 "MoveJ"
$ 15 "Waypoint_3" "breakAfter"
movej(get_inverse_kin(Waypoint_3_p, qnear=Waypoint_3_q),
  a=1.3962634015954636, v=0.5235987755982988
)
$ 17 "MoveL"
$ 18 "Waypoint_26" "breakAfter"
movel(Waypoint_26_p, a=1.2, v=0.1)
$ 19 "MoveL"
$ 36 "Testes tese"
$ 38 "Teste_tese_1" "breakAfter"
movel(Teste_tese_1_p, a=1.2, v=0.05)
$ 41 "p_inicial:=p"
global p_inicial=p
$ 42 "MoveL"
$ 43 "Direction: Base Z-"
global move_thread_flag_43=0
thread move_thread_43():
  enter_critical
  move_thread_flag_43 = 1
  local towardsPos=calculate_point_to_move_towards(
    p[0.0,0.0,0.0,0.0,0.0,0.0],
    [0.0,0.0,-1.0],
    1000.0
  )
  movel(towardsPos, a=1.2, v=0.01)
  move_thread_flag_43 = 2
  exit_critical
end
```

```
move_thread_flag_43 = 0
move_thread_han_43 = run move_thread_43()
while (True):
  local targetTcpDirection=get_target_tcp_speed()
  local stepsToRetract=tool_contact(direction=targetTcpDirection)
  if (stepsToRetract > 0):
    kill move_thread_han_43
    stopl(0.5)
    local backTrackMovement=get_actual_joint_positions_history(
      stepsToRetract
    )
    local contactPose=get_forward_kin(backTrackMovement)
    local posDir=[
      targetTcpDirection[0],
      targetTcpDirection[1],
      targetTcpDirection[2]
    ]
    local retractTo=contactPose
    if (norm(posDir) > 1e-6):
      local normalizedPosDir=normalize(posDir)
      local additionalRetraction=p[
        normalizedPosDir[0] * 0.0,
        normalizedPosDir[1] * 0.0,
        normalizedPosDir[2] * 0.0,
        0, 0, 0
      ]
      retractTo = pose_sub(contactPose, additionalRetraction)
    end
    movel(retractTo, a=0.5, v=0.1)
    $ 44 "Until (tool_contact_detection)"
    break
  end
  sync()
end
$ 45 "contacto:= True "
global contacto=  True
$ 46 "write_output_boolean_register(64, True )"
write_output_boolean_register(64,  True  )
$ 47 "T_stable: Start"
T_stable_is_counting = True
```

```
$ 48 "Script: f_stable_testes2.script"
force_mode_set_gain_scaling(1)
force_mode_set_damping(0)
force_mode(
  tool_pose(),
  [0,0,1,0,1,0],
  [0,0,45,0,0,0],
  2,
  [0.1,0.1,0.01,0.17,0.07,0.17]
)
$ 49 "MoveL"
$ 50 "Direction: Tool Z+"
global move_thread_flag_50=0
thread move_thread_50():
  enter_critical
  move_thread_flag_50 = 1
  local towardsPos=calculate_point_to_move_towards(
    get_forward_kin(),
    [0.0,0.0,1.0],
    1000.0
  )
  movel(towardsPos, a=1.2, v=0.07)
  move_thread_flag_50 = 2
  exit_critical
end
move_thread_flag_50 = 0
move_thread_han_50 = run move_thread_50()
while (True):
  if (T_stable >= 3.5):
    kill move_thread_han_50
    stopl(1.2)
    $ 51 "Until (expression)"
    break
  end
  sync()
end
$ 52 "end_force_mode()"
end_force_mode()
$ 53 "T_stable: Stop"
T_stable_is_counting = False
```

```
$ 54 "T_stable: Reset"
T_stable = 0
$ 55 "Script: f_move_y_testes3.script"
force_mode_set_gain_scaling(1)
force_mode(
  tool_pose(),
  [0,0,1,0,1,0],
  [0,5,30,0,0,0],
  2,
  [0.1,0.1,0.15,0.17,0.02,0.17]
)
$ 56 "Loop ft[1]≥-25"
thread Thread_while_56():
  while (True):
    $ 57 "MoveL"
    $ 58 "Waypoint_1" "breakAfter"
    set_tcp(p[0.0,0.0,0.21,0.0,0.0,0.0])
    movel(
      pose_trans(
        Point_1,
        pose_trans(
          p[
            .629725848440,
            .106116061017,
            -.713503853265,
            -1.749038451666,
            -.675477420826,
            1.808540875586
          ],
          Waypoint_1_p
        )
      ),
      a=1.2,
      v=0.01
    )
  end
end
if (ft[1] >= -25):
  global thread_handler_56=run Thread_while_56()
  while (ft[1] >= -25):
```

```
    sync()
  end
  kill thread_handler_56
end
$ 61 "end_force_mode()"
end_force_mode()
$ 62 "force_mode_set_gain_scaling(1)"
force_mode_set_gain_scaling(1)
$ 63 "d_p_inv:=2"
global d_p_inv=2
$ 64 "Loop d_p_inv>d2r(0.5)"
while (d_p_inv>d2r(0.5)):
  $ 65 "T_rz1: Start"
  T_rz1_is_counting = True
  $ 66 "Script: f_move_rz_testes.script"
  force_mode_set_gain_scaling(1.8)
  force_mode(
    tool_pose(),
    [0,1,1,0,1,1],
    [0,35,40,0,0,0],
    2,
    [0.1,0.008,0.15,0.17,0.01,0.2]
  )
  $ 67 "p_inv1:=p_inv[5]"
  global p_inv1=p_inv[5]
  $ 68 "Loop T_rz1≤0.4 and norm(p_inv[5]-p_inv1)<d2r(1.5)"
  thread Thread_while_68():
    while (True):
      $ 69 "MoveL"
      $ 70 "Direction: Tool"
      global move_thread_flag_70=0
      thread move_thread_70():
        enter_critical
        move_thread_flag_70 = 1
        local towardsPos=calculate_point_to_move_towards(
          get_forward_kin(),
          [0.0,1.0,1.0],
          0.1
        )
        movel(towardsPos, a=1.2, v=0.002)
```

```
      move_thread_flag_70 = 2
      exit_critical
    end
    move_thread_flag_70 = 0
    move_thread_han_70 = run move_thread_70()
    while (True):
      sleep(1.0E-10)
      if (move_thread_flag_70 > 1):
        join move_thread_han_70
        $ 71 "Until (distance)"
        break
      end
      sync()
    end
  end
end
if (T_rz1 <= 0.4 and norm(p_inv[5]-p_inv1)<d2r(1.5)):
  global thread_handler_68=run Thread_while_68()
  while (T_rz1 <= 0.4 and norm(p_inv[5]-p_inv1)<d2r(1.5)):
    sync()
  end
  kill thread_handler_68
end
$ 73 "p_inv2:=p_inv[5]"
global p_inv2=p_inv[5]
$ 74 "d_p_inv:=norm(p_inv2-p_inv1)"
global d_p_inv=norm(p_inv2-p_inv1)
$ 75 "end_force_mode()"
end_force_mode()
$ 76 "force_mode_set_gain_scaling(1)"
force_mode_set_gain_scaling(1)
$ 77 "T_rz1: Stop"
T_rz1_is_counting = False
$ 78 "T_rz1: Reset"
T_rz1 = 0
$ 79 "T_stable: Start"
T_stable_is_counting = True
$ 80 "Script: f_stable4_testes.script"
force_mode_set_gain_scaling(1)
force_mode(
```

```
    tool_pose(),
    [0,1,1,0,1,1],
    [0,0,30,0,0,0],
    2,
    [0.1,0.001,0.15,0.17,0.03,0.1]
  )
  $ 81 "Loop T_stable≤1"
  thread Thread_while_81():
    while (True):
      $ 82 "MoveL"
      $ 83 "Direction: Tool Z+"
      global move_thread_flag_83=0
      thread move_thread_83():
        enter_critical
        move_thread_flag_83 = 1
        local towardsPos=calculate_point_to_move_towards(
          get_forward_kin(),
          [0.0,0.0,1.0],
          0.1
        )
        movel(towardsPos, a=1.2, v=0.001)
        move_thread_flag_83 = 2
        exit_critical
      end
      move_thread_flag_83 = 0
      move_thread_han_83 = run move_thread_83()
      while (True):
        sleep(1.0E-10)
        if (move_thread_flag_83 > 1):
          join move_thread_han_83
          $ 84 "Until (distance)"
          break
        end
        sync()
      end
    end
  end
  if (T_stable <= 1):
    global thread_handler_81=run Thread_while_81()
    while (T_stable <= 1):
```

```
        sync()
      end
      kill thread_handler_81
    end
    $ 86 "end_force_mode()"
    end_force_mode()
    $ 87 "T_stable: Stop"
    T_stable_is_counting = False
    $ 88 "T_stable: Reset"
    T_stable = 0
  end
  $ 89 "Call exp_force_mode_ajustaX_su"
  exp_force_mode_ajustaX_su()
  $ 90 "T_stable: Start"
  T_stable_is_counting = True
  $ 91 "Script: f_stable3_testes3.script"
  force_mode_set_gain_scaling(1)
  force_mode(
    tool_pose(),
    [0,0,1,0,1,1],
    [0,5,30,0,0,0],
    2,
    [0.1,0.1,0.15,0.17,0.015,0.05]
  )
  $ 92 "Loop T_stable≤3"
  thread Thread_while_92():
    while (True):
      $ 93 "MoveL"
      $ 95 "Direction: Tool Z+"
      global move_thread_flag_95=0
      thread move_thread_95():
        enter_critical
        move_thread_flag_95 = 1
        local towardsPos=calculate_point_to_move_towards(
          get_forward_kin(),
          [0.0,0.0,1.0],
          0.1
        )
        movel(towardsPos, a=1.2, v=0.001)
        move_thread_flag_95 = 2
```

```
        exit_critical
      end
      move_thread_flag_95 = 0
      move_thread_han_95 = run move_thread_95()
      while (True):
        sleep(1.0E-10)
        if (move_thread_flag_95 > 1):
          join move_thread_han_95
          $ 96 "Until (distance)"
          break
        end
        sync()
      end
    end
  end
  if (T_stable <= 3):
    global thread_handler_92=run Thread_while_92()
    while (T_stable <= 3):
      sync()
    end
    kill thread_handler_92
  end
  $ 97 "end_force_mode()"
  end_force_mode()
  $ 98 "T_stable: Stop"
  T_stable_is_counting = False
  $ 99 "T_stable: Reset"
  T_stable = 0
  $ 100 "write_output_boolean_register(0, False )"
  write_output_boolean_register(0,  False  )
  $ 101 "contacto:= False "
  global contacto=  False
  $ 103 "Call exp_force_mode_caminho3_1"
  exp_force_mode_caminho3_1()
 end
end
```