Universidade de Trás-os-Montes e Alto Douro

Otimização e Controlo de um Braço Robótico com Técnicas Bio-inspiradas

Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores

Stefan Correia Vidal

Orientadores:

Prof. José Paulo Barroso de Moura Oliveira
Prof. Josenalde Barbosa de Oliveira

Vila Real, 2017

Universidade de Trás-os-Montes e Alto Douro

Otimização e Controlo de um Braço Robótico com Técnicas Bio-inspiradas

Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores

Mestrando: Stefan Correia Vidal

Orientador: José Paulo Barroso de Moura Oliveira

Coorientador: Josenalde Barbosa de Oliveira

Composição do Júri:

Prof. Doutor João Agostinho Batista de Lacerda Pavão

Prof.ª Doutora Filomena Maria Rocha Menezes Oliveira Soares

Prof. Doutor José Paulo Barroso de Moura Oliveira

Vila Real, 2017

A presente dissertação foi desenvolvida com o propósito de ser apresentada na Universidade de Trás-os-Montes e Alto Douro para obtenção do grau de Mestre em Engenharia Eletrotécnica e de Computadores. Declaro que todo o conteúdo presente é de minha inteira responsabilidade, sendo que as contribuições não originais se encontram devidamente referenciadas.

Agradecimentos

Primeiramente quero agradecer aos meus orientadores por toda a ajuda, dedicação e tempo disponibilizado para o bom desenvolvimento deste trabalho, pelas palavras motivadoras nos momentos de maiores dificuldades e acima de tudo pela confiança depositada nas minhas escolhas. Quero agradecer também ao INESC-TEC pela disponibilização do braço robótico UR5 bem como ao Prof. Boaventura Cunha.

Quero também agradecer à minha namorada por todos os conselhos ao longo destes anos que me ajudaram a traçar o meu caminho, aos meus pais por todo o apoio que me deram e pelo esforço que fizeram para possibilitar a minha formação, ao meu irmão por todas as palavras de incentivo e alento, aos pais da minha namorada pelo apoio incondicional, aos meus amigos por me acompanharem durante o meu percurso académico e a toda a minha restante família.

Resumo

Na robótica industrial o transporte de cargas suspensas tem de ser feito com garantias de segurança da carga e do meio envolvente, o que normalmente se traduz em movimentos lentos para não haver oscilação da carga. Este trabalho tem como objetivo a implementação de algoritmos de controlo que possibilitem o transporte rápido de uma massa suspensa de um ponto para outro criando a mínima oscilação possível da massa no final do movimento.

Neste trabalho foi utilizado o braço robótico UR5 da Universal Robots, ao qual foi acoplado um pêndulo na extremidade do mesmo, e foi definida uma trajetória linear a ser efetuada pela extremidade do braço de forma a poder criar as condições necessárias para o desenvolvimento do trabalho. Para diminuir a oscilação provocada pelo movimento rápido do braço robótico foi implementado um controlador *Posicast* que teve de ser adaptado tendo em conta a dinâmica do robô UR5. Para otimizar os parâmetros do controlador *Posicast* para o caso em estudo foi utilizado um algoritmo de otimização por enxame de partículas, adaptado a este problema, em conjunto com um simulador que permite prever o comportamento do braço robótico. Para validar a eficácia do algoritmo de otimização por enxame de partículas foi feito um conjunto de testes para diferentes condições de forma a poder comparar os resultados obtidos no simulador com os obtidos pelo robô e verificar se o parâmetro otimizado pelo algoritmo corresponde ao ideal a ser utilizado pelo UR5.

Palavras-chave: UR5, robótica, planeamento de trajetória, controlo de oscilação, *Posicast*, otimização por enxame de partículas.

Abstract

In industrial robotics the transport of suspended loads has to be performed with guarantees of safety for the load and the surrounding environment, which usually translates into slow movement so that there is no load oscillation. This work aim at the implementation of control algorithms that allow the rapid transport of a suspended mass from one point to another creating the minimum possible oscillation of the mass at the end of the movement.

In this work the robotic arm UR5 of Universal Robots was used, to which a pendulum was attached to the tool center point (TCP), and a linear trajectory was defined to be executed by the robot arm TCP in order to create the necessary conditions to perform the work. To reduce the oscillation caused by the rapid movement of the robotic arm a Posicast controller adapted taking into account the dynamics of the robot UR5 was implemented. To optimize the Posicast controller parameters for the case under study, a particle swarm optimization algorithm, was used in conjunction with a simulator that enables predicting robotic arm behavior. To validate the effectiveness of the particle swarm optimization algorithm, a set of tests was executed for different conditions in order to compare the results obtained in the simulator with those obtained by the robot and to verify if the optimized parameter by the algorithm corresponds to the ideal one to be used by UR5.

Keywords: UR5, robotics, planning of trajectory, oscillation control, Posicast, praticle swarm optimization.

Índice

Lista de figuras

Lista de tabelas

Lista de códigos

Pseudocódigo

Abreviaturas

ACO – *Ant Colony Optimization*
DH – Denevit-Hertenberg
PD – Proporcional Derivativo
PID – Proporcional Integral Derivativo
PSO – *Particle Swarm Optimization*
RCGA – *Real-Coded Genetic Algorithm*
ROS – *Robot Operating System*
RTDE – *Real-Time Data Exchange*
SI – *Swarm Intelligence*
TCP – *Tool Center Point*
UR – Universal Robots

1 - Introdução

A robótica industrial tem um impacto cada vez maior nas empresas e é uma área de investigação de grande atualidade na engenharia. Esta área tem sido alvo de investigação científica para diversos fins, sendo que a maioria visa aumentar a velocidade e a qualidade de produção de forma a acompanhar as necessidades da sociedade. A introdução de braços robóticos pelas empresas permite diminuir os custos de produção e, por consequência, diminuir o preço do produto final possibilitando a concorrência no mercado.

1.1 - Motivação e Objetivos

A movimentação de cargas suspensas utilizando manipuladores robóticos de uma forma rápida e com pequenas oscilações é uma tarefa complexa. Este trabalho procura encontrar uma forma de diminuir a oscilação de uma carga suspensa após um movimento rápido executado com o robô UR5 (Universal Robots, 2017a), na tentativa de poder contribuir para o desenvolvimento de técnicas cada vez mais precisas e eficazes para o transporte de cargas com este tipo de robôs. Para o efeito é utilizado o controlo *Posicast* de meio ciclo (Smith, 1957), que idealmente permite a movimentação de uma carga suspensa entre dois pontos sem oscilação na posição final.

Quando o sistema a controlar é modelado por um sistema de segunda ordem sub-amortecido, os resultados práticos obtidos com o controlo *Posicast* aproximam-se do comportamento ideal. No caso em que os modelos utilizados para o projeto são simplificados de alguma forma relativamente à dinâmica do sistema a controlar, a utilização de técnicas de otimização no projeto de controladores *Posicast* pode ser útil. No seguimento de bons resultados obtidos com o algoritmo de otimização com enxames de partículas (*Particle Swarm Optimization*, PSO) no projeto de controladores *Posicast* (Oliveira *et al.*, 2014; Oliveira *et al.*, 2017), esta técnica computacional bio-inspirada é utilizada neste trabalho como ferramenta de otimização.

O objetivo global deste trabalho é o desenvolvimento de técnicas computacionais que permitam o controlo da oscilação de uma massa suspensa acoplada ao robô UR5. Como objetivos específicos definiram-se:

- Estabelecer a comunicação via *socket* com o UR5 e fazer o controlo do movimento do mesmo através de *scripts* desenvolvidos em *Python* e *Matlab*;
- Definir uma trajetória propícia ao caso de estudo;
- Implementar um controlo *Posicast*, de forma a diminuir a oscilação da massa suspensa no final do movimento e adaptar o mesmo tendo em conta a dinâmica do robô;
- Otimizar os parâmetros do controlo *Posicast* com a utilização do algoritmo de otimização por enxame de partículas (*Particle Swarm Optimization*, PSO) adaptado para o caso de estudo;
- Ajustar o simulador para o caso de estudo;
- Analisar a eficácia do algoritmo PSO através de testes no robô com os parâmetros retornados pelo algoritmo;
- Comparar os resultados obtidos com o simulador com os obtidos pelo robô.

1.2 - Estrutura da Dissertação

Além deste capítulo, esta dissertação é constituída por mais sete:

- No capítulo 2 é apresentada uma visão geral sobre os conceitos fundamentais do robô UR5, abordando-se o tema da robótica industrial e a interação Homem-Robô. Nesse capítulo é apresentada também uma descrição do UR5, sua estrutura e especificações técnicas, sendo apresentados alguns exemplos de aplicação.
- No capítulo 3 é apresentado o funcionamento do controlo *Posicast* de meio ciclo e alguns exemplos de outros tipos de controlo *Posicast*.
- No capítulo 4 são analisados conceitos relacionados com o algoritmo de otimização por enxame de partículas e apresentados alguns exemplos de aplicação.
- No capítulo 5 são apresentados os conceitos fundamentais do simulador utilizado.
- No capítulo 6 é apresentado o estudo de caso, bem como a implementação do controlo *Posicast* no robô e no simulador, e são também apresentados os resultados obtidos para ambos os casos.
- No capítulo 7 é apresentada a comparação dos resultados obtidos com o simulador com os obtidos pelo robô para diferentes testes com diferentes condições.
- Por fim, no capítulo 8 são apresentadas as principais conclusões retiradas com o desenvolvimento deste trabalho.

2 - Braço Robótico UR5: Revisão de Conceitos Fundamentais

Neste capítulo apresentam-se os conceitos considerados fundamentais para a perceção do funcionamento do robô UR5 no contexto deste trabalho. O capítulo visa expor a importância da robótica industrial bem como da interação Homem-Robô e descrever o robô UR5 quanto à sua estrutura e características. Por fim, são mostrados alguns exemplos de aplicação do braço robótico UR5.

2.1 - Robótica industrial

O aumento constante da população tem levado a um crescimento significativo da sociedade de consumo nos últimos anos, o que gera uma maior necessidade de aumentar a eficiência dos processos produtivos. Os primeiros robôs começaram a surgir nos finais dos anos 50 (do século passado) devido à evolução dos componentes eletrónicos e também devido aos investimentos elevados das empresas automobilísticas que procuravam automatizar os seus processos de fabrico (Rocha, 2016). Atualmente, a robótica é aplicada não só nas indústrias de produção, mas também nas mais diversas áreas tais como na medicina, no ambiente doméstico, no auxílio de pessoas portadoras de deficiências, em ambientes perigosos para o ser humano, entre outros (Siciliano e Khatib, 2016).

Devido à grande variedade das áreas de utilização têm surgido diferentes definições para o que se entende por robô. Segundo o Instituto Americano de Robótica (1979) um robô é:

> *"Um manipulador reprogramável e multifuncional projetado para mover materiais, partes, ferramentas ou dispositivos especializados através de movimentos variáveis programados para desempenhar uma variedade de tarefas."* (Spong *et al.*, 2006).

A maioria dos robôs utilizados nos processos de fabrico industriais são os braços robóticos, ou seja, robôs com uma base fixa que realizam as suas tarefas numa área restrita, sendo que a sua flexibilidade depende do número de articulações que estes possuem. Os braços robóticos são maioritariamente utilizados pelas indústrias que têm um alto nível de produtividade, pois são utilizados processos tipicamente sequenciais e repetitivos, o que facilita a introdução destes robôs. Depois de programados corretamente, os braços robóticos em determinadas tarefas

apresentam uma maior eficiência em relação a um trabalhador pois são muito mais rápidos, não apresentam cansaço nem monotonia do trabalho, são mais precisos e podem operar de forma ininterrupta. Deste modo, a sua utilização permite que o trabalhador realize tarefas de mais alto nível, como as de supervisão (Rocha, 2016).

2.2 - Interação Homem-Robô

A interação Homem-Robô é uma preocupação que surgiu ainda antes dos robôs serem desenvolvidos. Isaac Asimov, um escritor russo-americano e um dos maiores autores de ficção científica, criou as leis da robótica, que se apresentam a seguir (Dhillon e Yang, 1996):

1. Um robô não pode ferir um humano ou, por omissão, permitir que o mesmo surja ferido.
2. Um robô deve obedecer sempre às ordens efetuadas pelos humanos, exceto quando estas entrem em conflito com a primeira lei.
3. Um robô deve proteger a sua própria existência, a não ser em cenários que contrarie a primeira ou a segunda lei.

O aparecimento dos primeiros robôs levou à necessidade de criar barreiras de proteção para garantir a integridade física do ser humano. Estes eram de grande dimensão e pouco precisos e um eventual contacto entre eles e as pessoas poderia provocar nestas elevados danos físicos. Assim, enquanto estes operavam, as pessoas não podiam permanecer na sua área de funcionamento (Müller *et al.*, 2014). No entanto, a cooperação entre o trabalhador e o robô é do interesse das empresas. Segundo Waltl (2015), "a cooperação direta entre homem e máquina é um elemento-chave da fábrica do futuro". Por um lado, a existência de robôs numa fábrica tem vantagens tais como a diminuição do desgaste físico dos trabalhadores, uma vez que estes já não necessitam de fazer trabalhos de esforço repetitivo que podem causar lesões a longo prazo. Por outro lado, a permanência dos humanos no processo produtivo permite manter o carácter criativo e subjetivo, que não é possível ser introduzido por robôs (Koller, 2015).

Com a evolução da tecnologia foi possível tornar realidade a interação Homem-Robô a operarem lado a lado, através do desenvolvimento dos primeiros robôs colaborativos, também designados de *cobots*. Este tipo de robôs incorporam sensores e mecanismos de segurança que permitem a sua utilização em conjunto com os seres humanos (Müller *et al.*, 2014).

Atualmente existem várias empresas que incorporam *cobots* a operar juntamente com os seus trabalhadores, principalmente na indústria automóvel. Empresas como a *Renault, Seat*, BMW, *Volkswagen* e *Audi* são alguns exemplos nos quais são utilizados robôs colaborativos da *Universal Robots* (Montes, 2015).

2.3 - Descrição do UR5

2.3.1 - Introdução

O UR5 é um robô desenvolvido pela empresa *Universal Robots* (UR), firma especializada na produção de robôs industriais. Para além deste robô, esta firma produz também o UR3 e o UR10, sendo que a principal diferença entre os três é a sua dimensão. Estes robôs têm o objetivo de manusear ferramentas e equipamentos de forma a processar determinados objetos. Um dos principais focos da UR é a segurança da utilização dos robôs em ambientes nos quais estão presentes seres humanos, de forma a poderem trabalhar em conjunto no mesmo espaço laboral. O UR5 possui mecanismos de segurança que permitem o contacto Homem-Robô sem que haja lesões, pois quando aplicada uma força superior a 150 N o robô automaticamente para de funcionar (Ostergaard, 2012).

Hoje em dia cada vez mais os robôs estão a ser integrados em diferentes áreas de trabalho. O fato de o UR5 ter a capacidade de se adaptar a diferentes necessidades de uma forma rápida e simples permite o seu destaque face a outros robôs. O UR5 é um robô de dimensão reduzida e possui um peso leve, o que facilita a sua deslocação de um lugar para outro, podendo realizar tarefas completamente diferentes (Ostergaard, 2012).

2.3.2 - Estrutura do UR5

O UR5 é constituído por um manipulador robótico e uma consola tátil, apresentados na Figura 2.1, e um controlador (Universal Robots, 2017a).

Figura 2.1: *a) Manipulador robótico e b) consola tátil (Universal Robots, 2017a).*

Na Figura 2.2, estão descriminadas as seis articulações que constituem o manipulador robótico UR5.

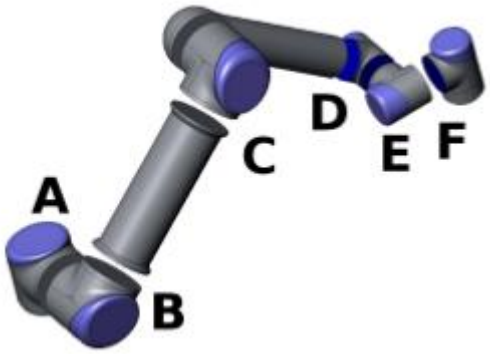

Figura 2.2: *Articulações do robô: A- Base; B- Ombro; C- Cotovelo; D,E,F- Pulso 1, 2, 3 (Universal Robots, 2017a).*

A consola tátil inclui uma interface gráfica com o utilizador, utilizando a ferramenta *PolyScope*, que permite operar o braço robótico, criar novos programas e executá-los. No entanto, a programação do robô pode ainda ser realizada por meio de *scripts*, com outras linguagens de programação, onde os comandos da linguagem *URScript* são passados por chamadas *sockets*.

2.3.3 - Especificações Técnicas

Na seguinte tabela apresentam-se as características técnicas mais relevantes do UR5:

Tabela 2.1: *Especificações técnicas do robô UR5. Universal Robots (2017a).*

Modelo	UR5
Peso	18,4 kg
Carga útil	5 kg
Alcance	850 mm
Amplitude das articulações	± 360° (todas as articulações)
Velocidade	Articulações: Máx. 180°/s Ferramenta: Aprox. 1 m/s
Repetibilidade	± 0.1 mm
Requisitos de espaço	Ø 149 mm
Graus de liberdade	6 Articulações giratórias
Tamanho da caixa de controlo	475 mm × 423 mm × 268 mm (L × A × P)
Portas de E/S da caixa de controlo	16 Entradas digitais 16 Saídas digitais 2 Entradas analógicas 2 Saídas analógicas
Portas de E/S da ferramenta	2 Entradas digitais 2 Saídas digitais 2 Entradas analógicas
Fonte de alimentação de E/S	24 V, 2 A (Caixa de controlo) 12 V / 24 V, 600 mA (Ferramenta)
Comunicação	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX soquete de Ethernet & Modbus TC
Programação	Interface gráfica do utilizador *PolyScope* em ecrã tátil de 12 polegadas
Ruído	Relativamente silencioso
Classificação IP	IP54
Consumo de energia	Aproximadamente 200W na utilização de um programa típico
Temperatura	0-50 °C
Fonte de alimentação	100-240 VAC, 50-60 HZ
Vida útil calculada	35000 horas

Das especificações técnicas deste robô são de destacar, para futura análise, a velocidade máxima atingida pela ferramenta e os graus de liberdade que esta possui.

2.3.4 - Exemplos de Aplicação

O braço robótico UR5 é utilizado por várias empresas de diferentes ramos devido às características flexíveis que apresenta. Este pode ser utilizado para realizar diversas atividades em diferentes áreas de trabalho. Na indústria automóvel é comum a utilização destes robôs para operações de montagem, soldadura, polimento, entre outras. Em empresas com grande produtividade estes robôs podem ser usados para executarem operações do tipo "*Pick and Place*" e de empacotamento, de forma a permitir um despacho do produto mais rápido e eficiente. Como o UR5 é um robô bastante preciso nos movimentos que executa, este é também frequentemente utilizado para análises e testes de laboratório, inspeções de qualidade e manutenção de máquinas (Universal Robots, 2017b). Alguns exemplos de empresas que utilizam este tipo de robôs são a *Nissan Motor Company*, que utiliza o braço robótico para a montagem de peças nos motores, a *Atria*, que é responsável por uma vasta gama de produção de produtos vegetarianos por dia e utiliza o UR5 para colocar os rótulos nos produtos e para fazer o empacotamento e a *Gentofte Hospital*, que utiliza dois robôs do tipo UR5 para otimizar o manuseamento e a triagem de amostras de sangue (Universal Robots, 2017b).

Existem vários projetos realizados com a utilização do braço robótico UR5 para diferentes ambientes de trabalho, sendo nesta secção apresentados alguns exemplos. Lei e Wisse (2014) utilizaram o UR5 para fazer operação do tipo "*Pick and Place*" com a ajuda de um sensor 3D instalado na extremidade do braço robótico que adquire uma nuvem de pontos referentes ao objeto a transportar e uma pinça Lacquey Fettch para agarrar o objeto. Šuligoj *et al.* (2015) mediram e avaliaram a aplicabilidade médica do braço robótico UR5 num sistema de rastreamento ótico (Polaris Vicra) medicamente certificado para posições registadas a partir de uma tomografia computadorizada. Wang *et al.* (2015) estudaram a cooperação da tecnologia de design de cirurgia virtual e navegação tridimensional de alta precisão, com o objetivo de implementar o projeto pré-operatório com maior precisão. Assim estes investigadores recorreram ao UR5 para melhorar a precisão e a qualidade operacional. Um dos projetos realizados com este robô que mais se assemelha ao presente trabalho é o projeto apresentado por Myhre e England (2016) que desenvolveram um método para fazer o rastreamento de um alvo oscilatório, utilizando rastreio visual (com marcação e câmara) do alvo, permitindo estimar a orientação do alvo e carregar um objeto no alvo oscilatório.

3 - Controlo *Posicast* no UR5

3.1 - Introdução

A utilização de gruas é muito comum nas aplicações industriais para fazer o transporte de cargas entre dois pontos, como por exemplo em terminais de contentores, em obras, em fábricas entre outros. Normalmente estas gruas não são automatizadas, necessitando de um operador humano para controlar o seu movimento, sendo tipicamente utilizados movimentos lentos para não ocorrer grandes oscilações da carga suspensa. Uma forma de aumentar o rendimento das empresas que utilizam gruas é o aumento da velocidade do transporte de cargas. No entanto, isso pode implicar a introdução de uma oscilação indesejada no movimento. Com a introdução de técnicas baseadas em computador é possível eliminar significativamente a oscilação final (Oliveira e Cunha, 2013).

Neste trabalho foi desenvolvido um sistema laboratorial, ilustrado na Figura 3.1, que acopla no ponto central da ferramenta do UR5 (*Tool Center Point*, TCP) uma haste metálica com uma massa suspensa (pêndulo). O objetivo global deste sistema é permitir o estudo do controlo da oscilação de uma massa suspensa acoplada a robô UR5. Para o efeito foi definida uma trajetória linear com o objetivo de executar o movimento de uma forma rápida, minimizando a oscilação após a chegada à posição final. Existem diversas formas que permitem diminuir a oscilação associada ao movimento da massa suspensa. Neste trabalho foi utilizado o controlo *Posicast*.

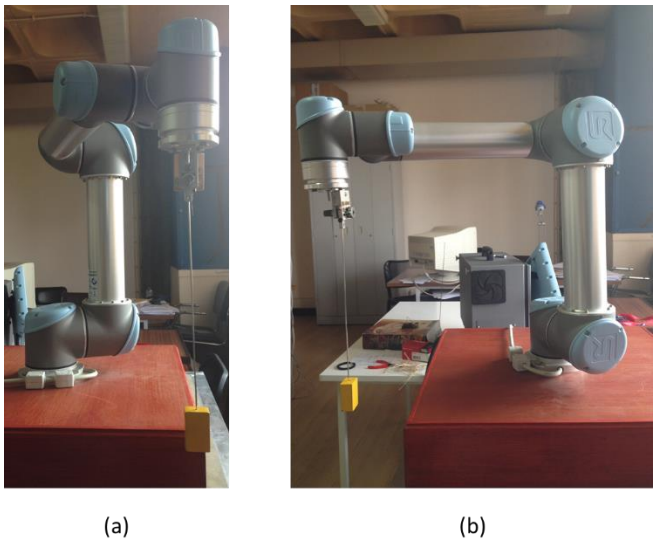

Figura 3.1: *a) Vista de frente e b) vista de perfil do sistema laboratorial utilizado neste trabalho.*

3.2 - Controlo *Posicast* de Meio Ciclo

O movimento do braço robótico gera uma oscilação do pêndulo cuja dinâmica pode ser modelada por um sistema de segunda ordem representado pelo seguinte modelo:

$$G_P(s) = \frac{\omega_n^2}{\omega_n^2 + 2\xi\omega_n s + s^2} \tag{3.1}$$

onde ω_n representa a frequência natural não amortecida e ξ o fator de amortecimento. Para o controlo deste movimento utilizou-se um controlador *Posicast*, que foi proposto originalmente por Smith (1957), permitindo o controlo em malha aberta (*Feedforward*), sendo que o órgão terminal do robô atuará como atuador que faz o movimento linear no qual o pêndulo está acoplado.

O controlo *Posicast* de meio-ciclo é uma técnica que pode ser implementada em malha aberta (e fechada) eficaz para supressão de vibrações indesejadas e consiste em dividir o sinal de entrada, que tipicamente representa apenas um degrau, em duas partes desfasadas no tempo. Neste trabalho foi aplicado este tipo de controlo, pelo que, em vez de consistir em um movimento constante da posição inicial para a final, passa a existir um ponto intermédio onde o movimento é interrompido por um determinado tempo (Oliveira e Vrančić, 2012). Como tal, é necessário encontrar a posição intermédia e o tempo de paragem nessa posição. O tempo de paragem na posição intermédia num sistema de segunda ordem sub-amortecido é também designado de tempo de pico (representado na Figura 3.2), e idealmente corresponde a metade do primeiro período de oscilação (T_d) podendo ser obtido pela seguinte equação (Vrančić e Oliveira, 2012):

$$T_p = \frac{T_d}{2} = \frac{\pi}{\omega_n\sqrt{1-\xi^2}} \tag{3.2}$$

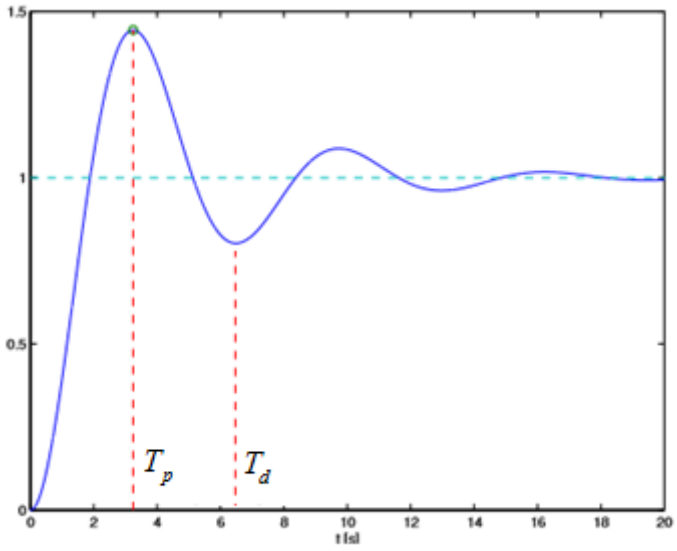

Figura 3.2: *Exemplo de uma resposta de um sistema de segunda ordem sub-amortecido quando a sua entrada é um degrau unitário (Vrančić e Oliveira, 2012, adaptada).*

Na Figura 3.3 estão representadas as posições do pêndulo ao longo do movimento necessárias para o cálculo da posição intermédia.

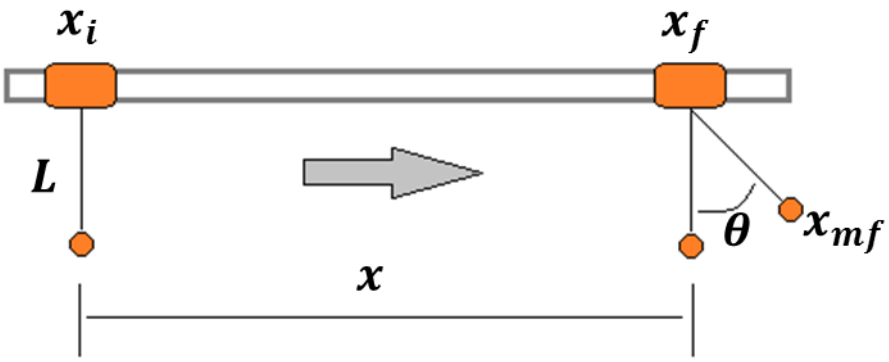

Figura 3.3: *Movimento do pêndulo sem controlo Posicast.*

Na Figura 3.3 x_i representa a posição inicial e x_f a posição final, correspondendo respetivamente a y=0 e y=1 na Figura 3.2, x representa a distância que se pretende que o pêndulo percorra, x_m é a posição da massa suspensa, x_{mf} é a posição da massa suspensa quando é atingido o ângulo máximo (θ) e L é o comprimento do cabo do pêndulo. Para determinar a posição intermédia de paragem é necessário calcular o valor da primeira sobre-elevação (δ, primeiro *overshoot*)

$$\delta = e^{-\frac{\xi\pi}{\sqrt{1-\xi^2}}} = \frac{x_{mf} - x}{x} \tag{3.3}$$

onde

$$x_{mf} = x + L\sin\theta \tag{3.4}$$

A função de transferência do controlador *Posicast* de meio ciclo pode ser definida como

$$G(s) = \frac{X_s(s)}{X(s)} = A_1 + A_2 e^{-\frac{T_d}{2}s} \tag{3.5}$$

Onde $X(s)$ representa a posição desejada (neste caso um sinal em degrau) e $X_s(s)$ o sinal modificado pelo *Posicast* de meio ciclo. A_1 e A_2 são definidas respetivamente por

$$A_1 = \frac{1}{1+\delta} \tag{3.6}$$

$$A_2 = 1 - A_1 \tag{3.7}$$

Nas Equações 3.6 e 3.7, os termos A_1 e A_2 consistem na amplitude das duas frações do movimento, ou seja, A_1 define a fração do movimento linear a percorrer até ao ponto intermédio de paragem, e A_2 define a fração pertencente ao restante movimento necessário para chegar à posição final. À segunda fração está associado um atraso equivalente a metade do período da oscilação, sendo este tipo de controlo designado de *Posicast* de meio ciclo.

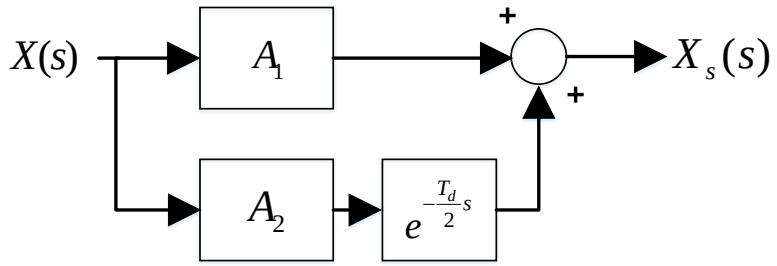

Figura 3.4: *Diagrama de blocos que representa o Posicast de meio ciclo.*

Após efetuar os cálculos é possível obter o ponto específico em que o movimento linear deve parar (x_p) e o tempo que ele deve permanecer nessa posição (T_p) para que o extremo do pêndulo atinga a posição final. No final do tempo de paragem calculado o movimento do manipulador é retomado até atingir o ponto final. Desta forma se o movimento for rápido é garantida a eliminação de grande parte da oscilação.

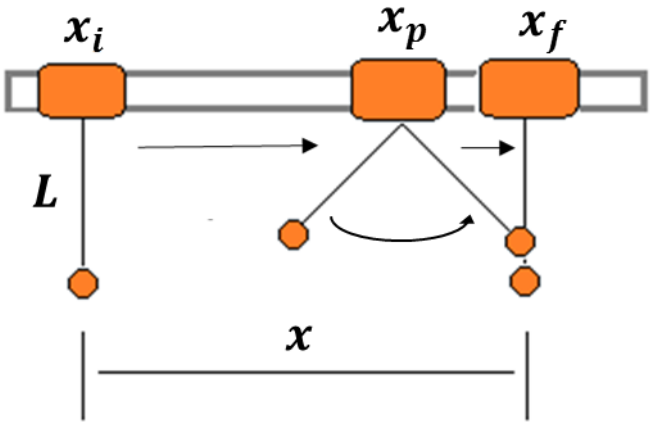

Figura 3.5: *Movimento do pêndulo com controlo Posicast.*

3.3 - Outros Tipos de Controlo *Posicast*

Embora a ideia original inicial do *Posicast* tenha surgido para o controlo em malha aberta, este tipo de controlo tem também grande aplicabilidade em malha fechada, tendo também sido proposta por Smith (1957) a utilização de uma estrutura com dois graus de liberdade que se apresenta na Figura 3.6.

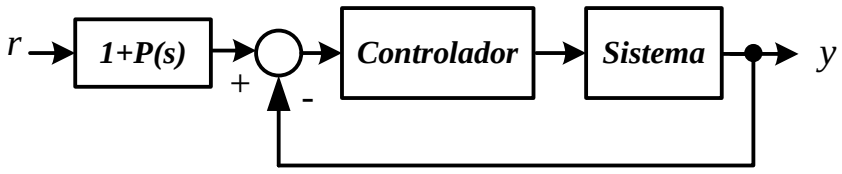

Figura 3.6: *Estrutura de controlo com dois graus de liberdade em que o Posicast é utilizado como pré-filtro.*

Na Figura 3.6, para o controlo *Posicast* de meio-ciclo, tem-se (Oliveira e Vrančić, 2012):

$$G_{hc}(s) = \frac{1}{1+\delta} + \frac{\delta}{1+\delta} e^{-\frac{T_d}{2}s} = 1 + P(s) \tag{3.8}$$

$$P(s) = \left(\frac{\delta}{1+\delta}\right)\left(-1 + e^{-\frac{T_d}{2}s}\right) \tag{3.9}$$

nas quais δ representa a primeira sobre-elevação. Este tipo de estrutura de controlo com dois graus de liberdade idealmente requer o projeto simultâneo do pré-filtro *Posicast* e do controlador da malha realimentada (e.g. um controlador proporcional integral derivativo, PID). Outras formas de utilização do controlo feedback com realimentação foram propostas desde o trabalho pioneiro de Smith (1957), tais como Hung (2007), Huey *et al.* (2008) e ainda Kucera e Hromcik (2011), nas quais o elemento de *Posicast* aparece inserido na malha de realimentação, como se ilustra na figura seguinte

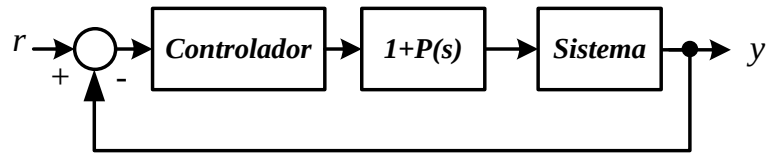

Figura 3.7: *Estrutura de controlo com dois graus de liberdade em que o Posicast é utilizado na malha de realimentação.*

Mais recentemente, Oliveira e Vrančić (2012) propuseram uma estrutura de controlo dual na qual o controlo *Posicast* é utilizado em malha aberta para assegurar uma resposta rápida no seguimento do sinal de referência, comutando-se depois para o controlo PID para assegurar a rejeição de perturbações. Este tipo de controlo está representado na Figura 3.8. Notar que este tipo de controlo só se deve utilizar em controlo de processos nos quais não ocorram muitas variações no sinal de referência.

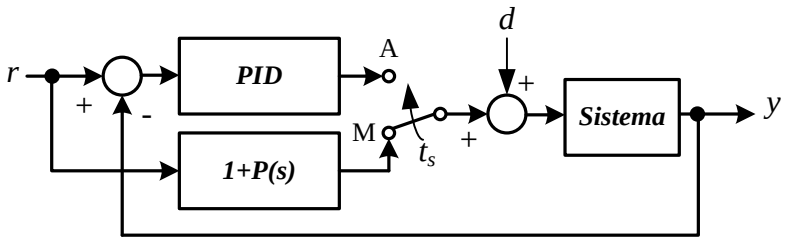

Figura 3.8: *Estrutura de controlo dual com controlo Posicast (Oliveira e Vrančić, 2012).*

4 - Algoritmo de Otimização por Enxame de Partículas

4.1 - Introdução

A inteligência de enxames (*Swarm Intelligence*, SI) é uma área da inteligência artificial que se preocupa com a implementação de sistemas inteligentes multi-agentes, que se baseiam na análise do comportamento de animais que vivem em sociedades, como por exemplo os insetos, formigas, abelhas, entre outros, bem como outras sociedades animais, tais como bandos e cardumes (Selvi e Umarani, 2010). Os algoritmos baseados em SI são utilizados com frequência (Ducatelle *et al.*, 2010; Senol *et al.*, 2016) para a resolução de problemas na engenharia devido ao fato de serem algoritmos flexíveis, robustos e simples de implementar. Estes são constituídos por um conjunto de agentes em que cada um atua localmente e realiza alguma interação com o grupo, sendo criado um ambiente facilitador para encontrar soluções globais do problema sem ser necessária uma entidade central de controlo. Os algoritmos mais populares que se baseiam na inteligência de enxames são a otimização por colónia de formigas (*Ant Colony Optimization*, ACO) e a otimização por enxame de partículas (*Particle Swarm Optimization*, PSO) (Serapião, 2009). Neste trabalho apenas será utilizado o PSO, devido à simplicidade da sua implementação e às provas dadas na resolução de problemas de pesquisa e otimização (Parsopoulos e Vrahatis, 2010).

Um estudo percursor na simulação gráfica do comportamento de grupos de agentes foi feito por Reynolds (1987), que através da observação do comportamento social de diferentes bandos e cardumes percebeu que estes executavam movimentos sincronizados sem que fosse necessário um controlo centralizado. Com base nas movimentações dos bandos o investigador criou um modelo (Boids) com apenas três regras de comportamentos individuais que correspondem às forças opostas de evitar colisões e ao desejo de unir o grupo (Reynolds, 1987):

1. **Separação:** Para evitar a colisão entre eles.
2. **Alinhamento:** Para que cada indivíduo siga a mesma direção dos vizinhos.
3. **Coesão:** Para que cada indivíduo siga a mesma posição dos vizinhos.

Mais tarde, Kennedy e Eberhart (1995) utilizaram o modelo criado por Reynolds como inspiração para implementar o PSO e aplicá-lo na resolução de problemas no domínio contínuo.

4.2 - PSO: Conceitos Fundamentais

O algoritmo PSO simula o comportamento de sociedades de animais que não possuem um líder. Normalmente todos os animais encontram o alimento em simultâneo por seguirem o membro do grupo que mais próximo se encontra da fonte de alimentação (solução ótima). Isto apenas é possível por haver uma constante comunicação entre eles para que todos saibam qual a melhor posição atual (Rini *et al.*, 2011). Todos os indivíduos de um grupo social possuem a sua própria experiência e sabem como quantificá-la e compará-la com a dos vizinhos, logo a tomada de decisão de um indivíduo depende do seu desempenho individual no passado bem como do desempenho de alguns (ou todos) dos seus vizinhos (Serapião, 2009).

No algoritmo PSO, cada indivíduo é representado por um ponto que se movimenta num espaço de pesquisa multidimensional, cuja topologia é definida por uma função objetivo específica do problema, e é um candidato à solução do problema, sendo designado de partícula. Cada partícula tem dois atributos associados: um valor de posição que indica o quanto a partícula é adequada para a resolução do problema e um valor de velocidade que determina a direção do movimento da partícula. As variações desses atributos implicam a movimentação das partículas no espaço que por sua vez influenciam as movimentações das partículas vizinhas, pois a velocidade varia com a melhor posição obtida pela experiência da partícula e com a melhor posição global obtida pela experiência do grupo (Medeiros, 2005).

Uma partícula i desloca-se para uma nova posição consoante a sua posição atual $x_i(t)$, a velocidade $v_i(t+1)$. Uma das equações fundamentais para o PSO é a Equação 4.1 que permite atualizar a velocidade da partícula (Kennedy e Eberhart, 1995; Waintraub, 2009):

$$v_i(t+1) = v_i(t) + c_1\varphi_1 \cdot (b_1(t) - x_i(t)) + c_2\varphi_2 \cdot (g(t) - x_i(t)) \tag{4.1}$$

onde $(b_1(t) - x_i(t))$ representa a própria experiência da partícula, sendo $b_1(t)$ a posição na qual obteve melhor desempenho até ao momento e $(g(t) - x_i(t))$ representa a partilha social com o grupo, onde $g(t)$ é a posição na qual houve um melhor desempenho global até ao momento. Os termos c_1 e c_2 são constantes que geralmente são designadas respetivamente de componente

cognitiva e social, φ_1 e φ_2 são valores gerados aleatoriamente entre 0 e 1 para perturbar as duas diferenças na pesquisa.

Após calcular a velocidade da partícula i, a sua nova posição na próxima iteração será o resultado da adição da posição atual com a velocidade calculada

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{4.2}$$

Uma forma de diminuir o efeito da partícula sair do espaço de pesquisa é impondo limites para a velocidade ($v_{\max}$) para cada partícula, ou seja, se $v_i > v_{\max}$ então v_i passa a ser igual a $v_{\max}$ caso contrário se $v_i < -v_{\max}$ então v_i passa a ser igual a $-v_{\max}$ (Serapião, 2009). O pseudocódigo do algoritmo de otimização por enxame de partículas encontra-se representado no Pseudocódigo 4.1.

Pseudocódigo 4.1: *Pseudocódigo do PSO.*

---

t=0 % Inicializar o contador das iterações

Inicializar o enxame X(t)

Avaliar X(t)

Enquanto *(! (Não satisfaz critério de paragem))*

t=t+1
Atualizar as melhores posições local e global
Atualizar as velocidades das partículas
Atualizar as posições das partículas
Determinar X(t+1)
Avaliar X(t+1)

fim enquanto

4.3 - Aplicação do PSO na Robótica

Inicialmente o PSO começou por ser utilizado por Kennedy e Eberhart (1995) para formar arquiteturas de redes neuronais artificiais. No entanto, devido ao sucesso que o algoritmo obteve, atualmente este é utilizado em diversas áreas como por exemplo em sistemas fotovoltaicos, distribuição de energia elétrica, telecomunicações, entre outros (Wu *et al*., 2016). Na robótica, o PSO é aplicado para a resolução de diferentes problemas, apresentando-se de seguida alguns exemplos:

- Alavandar *et al.* (2009) utilizaram o algoritmo PSO para otimizar o parâmetros de um controlador proporcional derivativo (PD), para fazer o controlo de trajetória de um braço robótico com duas articulações.
- Huang *et al.* (2012) utilizaram o algoritmo PSO para resolver o problema de cinemática inversa redundante para um braço robótico com sete graus de liberdade (7-DOF) de uma forma mais eficiente, pois para robôs deste tipo a cinemática inversa envolve funções trigonométricas inversas complexas e não possui uma solução única.
- Tsai *et al.* (2014) para determinar os sete ângulos das articulações de um braço robótico 7-DOF durante a deslocação de um ponto para outro utilizaram a combinação do algoritmo PSO com o RCGA (*Real-Coded Genetic Algorithm*), de forma a poderem ter uma trajetória linear livre de colisões.

5 - Conceitos Fundamentais Utilizados no Simulador

Neste capítulo são apresentados vários conceitos referentes ao UR5 que são utilizados pelo simulador de forma a poder executar uma simulação do movimento do braço robótico de uma forma mais realista.

5.1 - Dinâmica do Braço Robótico UR5

Para perceber como é executado o movimento do braço robótico e o impacto gerado pelos binários é utilizada a seguinte equação de movimento

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{5.1}$$

onde q é o vetor que representa o ângulo dos elos, $M(q)$ é a matriz de inércia, $C(q,\dot{q})$ contem termos associados à aceleração centrífuga e à pseudoforça de Coriolis, $g(q)$ é o vetor de gravidade e τ o vetor dos binários de entrada. Esta equação pode ser utilizada de duas formas diferentes, isto é, pode ser utilizada para obter o estado final $(q,\dot{q})$ através do binário de entrada conhecido, usando dinâmica direta, ou pode ser utilizada para obter o binário de entrada para determinado estado conhecido $(q,\dot{q})$, usando dinâmica inversa (Ragazzon, 2012). Existem várias formas para obter as matrizes da dinâmica de um robô. No simulador foram utilizadas as matrizes obtidas pelo *framework* desenvolvido por Høifødt (2011) que implementa o método computacional de Newton-Euler com base nos parâmetros de Denavit-Hartenberg (DH). O *framework* gera as matrizes $M(q)$, $C(q,\dot{q})$ e $g(q)$ e converte-as para um formato legível pelo *Matlab* ®.

5.1.1 - Método de Newton-Euler

O método de Newton-Euler permite obter as matrizes da dinâmica do robô calculando as forças e os binários para cada articulação, sendo que cada articulação é tratada separadamente. Este método divide-se em duas partes, inicialmente são calculadas a velocidade e a aceleração para cada articulação, sendo o cálculo efetuado da primeira articulação até a última por ordem. Numa

segunda fase são calculados os binários e as forças para cada articulação na ordem contrária à primeira fase (Ragazzon, 2012).

5.1.2 - Parâmetros de Denavit-Hartenberg (DH)

Os parâmetros DH definem como um sistema de referências é associado às articulações de um braço robótico, permitindo a criação de matrizes de transformação. A informação relativamente a cada articulação para o UR5, segundo o método DH, encontra-se na Tabela 5.1 e é fornecida pelo fabricante.

Tabela 5.1: *Parâmetros DH para o UR5 (Tavares, 2015).*

Articulação	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	d_1	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3
4	0	$\pi/2$	d_4	θ_4
5	0	$-\pi/2$	d_5	θ_5
6	0	0	d_6	θ_6

Os parâmetros são definidos da seguinte forma:

- a_i - Distância entre o eixo *i*-1 e a origem do referencial o_i.
- α_i - Ângulo formado entre o eixo *i*-1 e o eixo *i* no plano normal a x_i.
- d_i - Distância entre a origem do referencial *i*-1, o_{i-1} e o eixo x_i.
- θ_i - Ângulo formado entre o eixo x_{i-1} e o eixo x_i no plano normal a z_{i-1}.

A Figura 5.1 representa a distribuição do sistema de coordenadas para o braço robótico UR5. Notar que as dimensões das articulações não se encontram à escala.

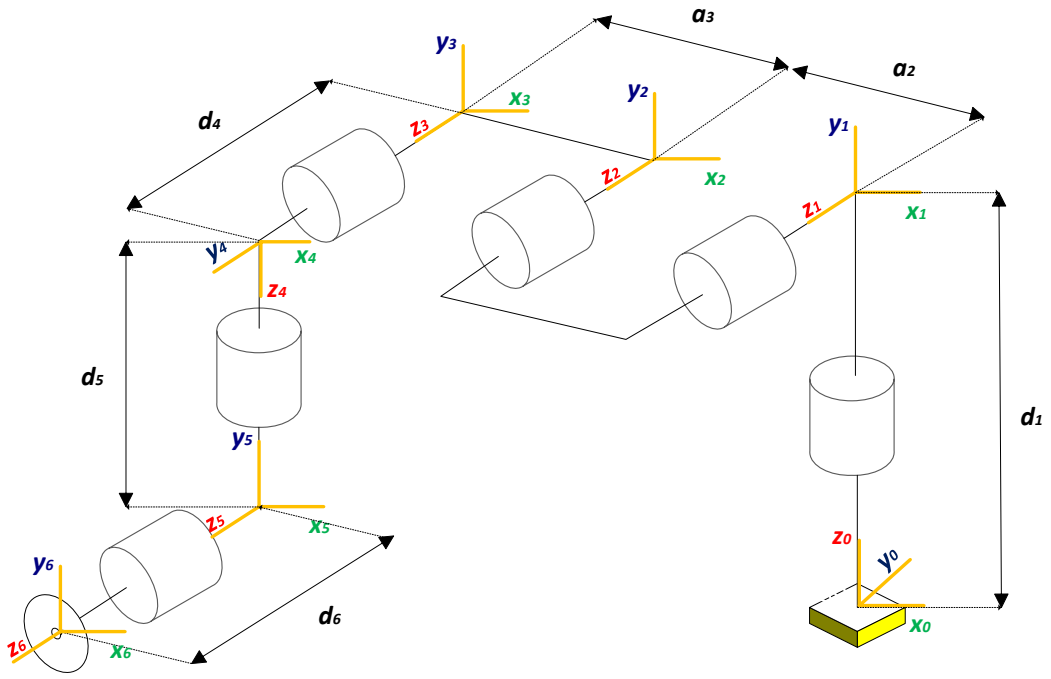

Figura 5.1: *Sistema de referências atribuído ao UR5.*

Através da análise da Figura 5.1 é possível obter os vetores que definem a translação entre referenciais, $r_{i-1,i}$.

$$r_{01} = [0 \quad d_1 \quad 0] \tag{5.2}$$

$$r_{12} = [-a_2 \quad 0 \quad 0] \tag{5.3}$$

$$r_{23} = [-a_3 \quad 0 \quad 0] \tag{5.4}$$

$$r_{34} = [0 \quad d_4 \quad 0] \tag{5.5}$$

$$r_{45} = [0 \quad -d_5 \quad 0] \tag{5.6}$$

$$r_{56} = [0 \quad 0 \quad d_6] \tag{5.7}$$

A matriz de rotação do referencial *i-1* para *i* (R_{i-1}^{i}) é dada por:

$$R_{i-1}^{i} = R_{z,\theta_i} R_{x,\alpha_i} \tag{5.8}$$

As matrizes R_{z,θ_i} e R_{x,α_i} representam, respetivamente, a rotação em torno do eixo *z* e *x*, sendo estas dadas por:

$$R_{z,\theta_i} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

$$R_{x,\alpha_i} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{5.10}$$

5.2 - Cinemática do Robô e Matriz Jacobiana

Para ser possível traçar o gráfico da deslocação do ponto central da ferramenta (TCP), o simulador contém um *script* que implementa a matriz de transformação para cinemática direta e a matriz Jacobiana. Estas matrizes são implementadas para obter a posição no espaço cartesiano, a velocidade e aceleração linear do TCP através de um vetor com as posições angulares das seis articulações. Para este trabalho apenas é necessário obter a posição *y* do TCP, visto que é utilizado um movimento linear, não sendo relevantes as posições *x* e *z*. Para a simulação as rotações *Rx*, *Ry* e *Rz* também não são consideradas.

5.2.1 - Cinemática Direta

O processo que permite obter a posição do robô (*x*, *y* e *z*) consoante o estado das suas articulações, que para o UR5 são as posições angulares visto que possui articulações rotacionais, é chamado de cinemática direta. As matrizes de transformação homogéneas são cruciais para a descrição da cinemática. A matriz de transformação homogénea que permite obter a posição e a orientação de $P_i x_i y_i z_i$ (coordenadas cartesianas de *i*) em relação a $P_{i-1} x_{i-1} y_{i-1} z_{i-1}$ é dada por (Kufieta, 2014):

$$A_i = \begin{bmatrix} R_i^{i-1} & P_i^{i-1} \\ 0 & 1 \end{bmatrix} \tag{5.11}$$

A matriz de transformação homogénea que permite obter a posição e a orientação de $P_j x_j y_j z_j$ em relação a $P_i x_i y_i z_i$ é dada por:

$$T_j^i = \begin{cases} A_{i+1}A_{i+2}...A_{j-1}A_j, & if\ i < j \\ I, & if\ i = j \\ (T_j^i)^{-1} & if\ i = j \end{cases} \tag{5.12}$$

A posição e a orientação do TCP são dados por:

$$T_n^0 = A_1(q_1)...A_n(q_n) = \begin{bmatrix} R_n^o & P_i^0 \\ 0 & 1 \end{bmatrix} \tag{5.13}$$

Tendo como base o sistema de coordenadas obtido através dos parâmetros DH como ilustrado na Figura 5.1, a matriz homogénea A_i é dada por (Kufieta, 2014; Tavares, 2015):

$$\begin{aligned} A_i &= Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &\quad \times \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \cos\theta_i & -\sin\theta_i & 0 \\ 0 & \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i \times \cos\alpha_i & \sin\theta_i \times \cos\alpha_i & a_i \times \cos\theta_i \\ \sin\theta_i & \cos\theta_i \times \cos\alpha_i & -\cos\theta_i \times \cos\alpha_i & a_i \times \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{5.14}$$

No simulador esta matriz está implementada num *script*, permitindo obter as posições de *y* ao longo do movimento simulado e assim traçar o gráfico da deslocação do TCP.

5.2.2 - Matriz Jacobiana

Com a implementação da matriz de transformação para cinemática direta no simulador é possível visualizar o movimento do braço robótico. No entanto, para tornar a simulação mais realística é necessário que haja um controlo da velocidade ($\dot{y}$) e da aceleração ($\ddot{y}$) linear do

movimento. Para obter $\dot{y}$ e $\ddot{y}$ é utilizada a matriz Jacobiana definida para manipuladores robóticos (Spong *et al*., 2006; Mathiassen *et al*., 2016)

$$J(q)=\begin{bmatrix} -P^0_{06,y} & -c_1C_{11} & -c_1C_{12} & -c_1C_{13} & -s_1s_5d_6-C_{cc}c_5d_6 & 0 \\ -P^0_{06,x} & -s_1C_{11} & -s_1C_{12} & -s_1C_{13} & c_1s_5d_6-C_{sc}c_5d_6 & 0 \\ 0 & C_{21} & C_{22} & C_{23} & -s_{234}c_5d_6 & 0 \\ 0 & s_1 & s_1 & s_1 & C_{cs} & -C_{cc}s_5+s_1c_5 \\ 0 & -c_1 & -c_1 & -c_1 & C_{ss} & -C_{sc}s_5+c_1c_5 \\ 1 & 0 & 0 & 0 & -c_{234} & -s_{234}s_5 \end{bmatrix} \tag{5.15}$$

onde

$$s_i=\sin(\theta_i) \qquad c_i=\cos(\theta_i) \tag{5.16a}$$

$$s_{abc}=\sin(\theta_a+\theta_b+\theta_c) \qquad c_{abc}=\cos(\theta_a+\theta_b+\theta_c) \tag{5.16b}$$

$$C_{11}=K_{11}+K_{12}+K_{13} \qquad C_{12}=K_{12}+K_{13} \qquad C_{13}=K_{13} \tag{5.17a}$$

$$C_{21}=K_{21}+K_{22}+K_{23} \qquad C_{22}=K_{22}+K_{23} \qquad C_{13}=K_{13} \tag{5.17b}$$

$$C_{ss}=s_1s_{234} \qquad C_{cs}=c_1s_{234} \tag{5.18a}$$

$$C_{sc}=s_1c_{234} \qquad C_{cc}=c_1c_{234} \tag{5.18b}$$

$$K_{11}=s_2a_2 \qquad K_{12}=s_{23}a_3 \qquad K_{13}=-c_{234}d_5-s_{234}s_5d_6 \tag{5.19a}$$

$$K_{21}=c_2a_2 \qquad K_{22}=c_{23}a_3 \qquad K_{23}=s_{234}d_5-c_{234}s_5d_6 \tag{5.19b}$$

A velocidade e a aceleração linear são obtidas através das seguintes equações (Kufieta, 2014):

$$\dot{y}=J(q)\dot{q} \tag{5.20}$$

$$\ddot{y}=J(q)\ddot{q}+\left(\frac{d}{dt}J(q)\right)\dot{q} \tag{5.21}$$

A posição angular (q), a velocidade angular ($\dot{q}$) e a aceleração angular ($\ddot{q}$), são valores conhecidos, visto que, são parâmetros enviados ao robô UR5 quando pretendemos que este se movimente. A derivada temporal da matriz Jacobiana é estimada através do método de Euler.

5.3 - Modelo do Pêndulo

Após calcular a aceleração linear ($\ddot{y}$) através da matriz Jacobiana é possível obter o ângulo do pêndulo (θ) ao longo do movimento, utilizando o seguinte modelo matemático não linear (Oliveira e Cunha, 2013)

$$L\ddot{\theta} + \ddot{y}\cos\theta + g\sin\theta + d\dot{\theta} = 0 \tag{5.22}$$

onde $\dot{\theta}$ representa a velocidade angular, $\ddot{\theta}$ representa a aceleração angular, L representa o comprimento do cabo do pêndulo, g representa a aceleração gravitacional constante e d representa o coeficiente de amortecimento. Com a obtenção do ângulo é possível traçar o gráfico com a variação do mesmo ao longo do movimento. O gráfico da variação da posição do extremo do pêndulo é obtido através da Equação 3.4.

5.4 - Dinâmica Inversa

O braço robótico UR5 utiliza um controlador interno para obter os binários de cada articulação consoante a posição do TCP pretendida. Como não é permitido o acesso ao controlador interno do robô e este não é exposto pelo fabricante, para gerar os binários na simulação é utilizado um controlador PD por dinâmica inversa.

Através da dinâmica inversa é possível obter o binário para cada articulação consoante o estado $(q, \dot{q})$ do robô. Considerando a equação de controlo (Ragazzon, 2012)

$$\tau = M(q)a_q + C(q,\dot{q})\dot{q} + g(q) \tag{5.23}$$

onde a_q é a variável a determinar. Como a matriz de inércia $M(q)$ é invertível, ao combinar a Equação 5.23 com a Equação 5.1 obtém-se

$$\ddot{q} = a_q \tag{5.24}$$

que representa um sistema de integrador duplo onde a_q é uma nova entrada que pode ser definida como um controlador PD com aceleração *feedforward*

$$a_q = \ddot{q}_d - K_0 \tilde{q} - K_1 \dot{\tilde{q}} \tag{5.25}$$

onde $\tilde{q} = q - q_d$ e $\dot{\tilde{q}} = \dot{q} - \dot{q}_d$. K_0 e K_1 são matrizes diagonais constituídas por ganhos de posição e velocidade, respetivamente.

As matrizes K_0 e K_1 podem ser definidas como

$$K_0 = \begin{bmatrix} w_1^2 & 0 & \dots & 0 \\ 0 & w_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_n^2 \end{bmatrix} \qquad K_1 = \begin{bmatrix} 2w_1 & 0 & \dots & 0 \\ 0 & 2w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 2w_n \end{bmatrix} \tag{5.26}$$

onde o vetor das frequências naturais é definido como $w = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}^T$. Com estas matrizes obtém-se um sistema de malha fechada desacoplado. A resposta para cada articulação é um sistema de segunda ordem linear com amortecimento crítico e com uma frequência natural w_i. Após definir as matrizes K_0 e K_1, ao aplicar a trajetória de referência $\left(q_d(t), \dot{q}_d(t), \ddot{q}_d(t) \right)$ à Equação 5.25 é obtido o valor de a_q e assim é possível obter o valor do binário de entrada através da Equação 5.23.

6 - Estudo de Caso

6.1 - Implementação do Controlo *Posicast* no UR5

Numa fase inicial da experimentação do UR5 foi feito um estudo sobre o funcionamento do braço robótico utilizando o *PolyScope* para fazer os primeiros testes de movimentos entre diferentes pontos. Os testes realizados tiveram como objetivo estabelecer a trajetória desejada e perceber qual a influência dos diferentes tipos de movimentos possíveis com a utilização do robô.

O UR5 permite a utilização de três tipos de movimentos diferentes (Universal Robots, 2017a):

- **Movimento J:** Este tipo de movimento implica que a ferramenta percorra um caminho curvilíneo entre dois pontos, pois não existem restrições para a deslocação entre os mesmos, o que resulta numa movimentação rápida. Os parâmetros associados a este movimento são a velocidade máxima e a aceleração da articulação.

- **Movimento L:** É utilizado quando se pretende um movimento linear entre diferentes pontos, implicando uma maior complexidade na movimentação de cada articulação para assegurar uma trajetória linear da ferramenta. Os parâmetros utilizados por este tipo de movimento são a velocidade e aceleração desejada.

- **Movimento P:** Este movimento utiliza uma velocidade constante e trajetórias lineares da ferramenta com uniões circulares e tem associado um raio de união que por defeito é compartilhado por todos os pontos. Quanto maior for o raio de união maior será a suavidade do caminho. A este tipo de movimento pode ser associado um movimento circular.

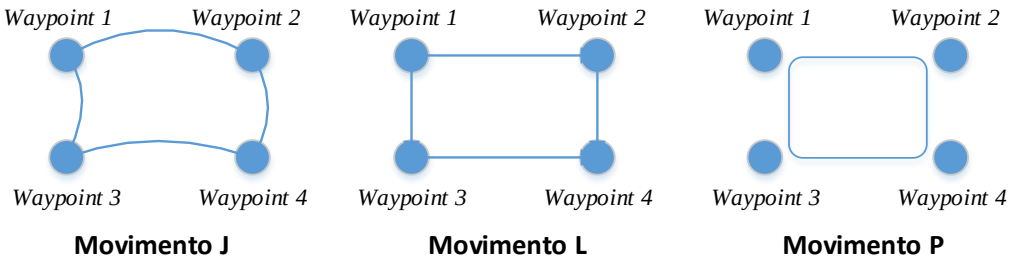

Figura 6.1: *Tipos de movimento.*

Analisando os diferentes tipos de movimentos apresentados na Figura 6.1 verifica-se que apenas o movimento *J* e o movimento *L* podem ser utilizados, visto que o movimento *P* é mais direcionado para movimentos circulares. Como é pretendido um movimento rápido para criar a oscilação necessária para o desenvolvimento do trabalho, o mais adequado seria o movimento *J*, mas como este produz um caminho curvilíneo é introduzida uma oscilação perpendicular à desejada. Com o movimento *L* é resolvido em parte o problema do movimento *J* pois introduz um menor erro na oscilação, apresentando no entanto um movimento mais lento. Assim, neste trabalho foi utilizado o movimento *L*, sendo que para compensar a velocidade reduzida do deslocamento com este movimento foi aumentado o valor da velocidade máxima nas configurações do robô. Por defeito o UR5 possui um valor de $1{,}5\,m/s$ que foi alterado para $5\ m/s$ (Universal Robots, 2017a).

Uma vez que o objetivo deste estudo é fazer o controlo do braço robótico através de um dispositivo externo, foi necessário estabelecer a conexão entre ambos. Para fazer a troca de dados foi utilizada uma comunicação via *socket*, sendo o código desenvolvido em *Python*. O Código 6.1 foi utilizado para estabelecer a comunicação.

Código 6.1: *Código para estabelecer uma comunicação via socket.*

```
Robot_IP = "192.168.10.202"
Port = 30002;
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((Robot_IP, Port))
```

Para estabelecer a conexão são utilizados dois parâmetros: o IP do robô e a porta para a troca de dados. Após estabelecer a ligação, o trabalho foi dividido em duas partes, consistindo no envio de comandos para o braço robótico e na aquisição dos dados provenientes do mesmo. Inicialmente, começou-se por implementar a trajetória estabelecida previamente com o

PolyScope através do envio de comandos em *Python*. Com o *PolyScope*, os pontos da trajetória foram escolhidos manualmente e guardados, não sendo necessário indicar a posição dos mesmos. No entanto, para enviar um comando através de um dispositivo externo é necessário saber previamente os valores associados aos diferentes pontos. Como o braço robótico é constituído por várias articulações é necessário saber o ângulo correspondente a cada uma para poder enviar um comando que permita fazer com que o robô execute um movimento linear para um determinado ponto. Através do Código 6.2 é enviado um comando que indica ao robô para se deslocar em direção à posição final com um movimento do tipo *L*.

Código 6.2: *Código para executar determinado movimento.*

```
#Código para movimentar o robô para a posição final
a=str(math.radians(-225.07))
b=str(math.radians(-90.23))
c=str(math.radians(90.32))
d=str(math.radians(-89.95))
e=str(math.radians(-90.35))
f=str(math.radians(46.87))

PosB = 'movel(['+a+','+b+','+c+','+d+','+e+','+f+'], a=4, v=5)'+'\n'

s.send (PosB.encode('utf-8'))
```

Para enviar um comando de movimentação linear o ângulo referente a cada articulação tem de ser enviado em radianos. Qualquer comando a enviar tem de ser codificado para que o robô o execute. Quando é enviada uma sequência de comandos ao robô é necessário definir intervalos de tempo entre eles, pois quando são enviados em simultâneo, o robô executa o último comando recebido e os restantes comandos são ignorados.

Depois de implementar o código que permite a execução da trajetória pretendida e de se perceber como funciona o envio de comandos através de um *script* em *Python*, foi feito um estudo sobre a aquisição de dados. Foram implementadas duas formas distintas para fazer a aquisição de dados. Inicialmente, foi feita a leitura dos dados retornados pela porta 30002 e isolado o valor da posição em que se encontra o robô. No entanto, este método não era muito adequado para o pretendido, pois pela porta 30002 os dados são transmitidos com uma frequência de 10 Hz, ou seja, os dados são enviados com um período de 0,1 segundo mas, como o braço robótico executa a trajetória de uma forma rápida num tempo aproximado a 1 segundo,

os dados obtidos não são suficientes para encontrar a posição intermédia necessária para uma fase posterior.

Como é necessário fazer a aquisição do maior número de pontos possível da posição do robô durante a sua movimentação para posteriormente ser possível encontrar o ponto intermédio mais próximo do calculado, foi utilizado um protocolo desenvolvido pela Universal Robots, *Real-Time Data Exchange* (RTDE) (Universal Robots, 2017c). O protocolo RTDE permite o sincronismo entre aplicações externas e o controlador UR sem interferir nas propriedades de tempo real do controlador. Este protocolo utiliza a porta 30004 com uma frequência de 125 Hz para a transmissão de dados, ou seja, os dados são enviados com um período de 0,008 segundo sendo assim possível adquirir um número muito mais elevado do que pelo método anterior. Para poder utilizar este protocolo o *software* do robô teve de ser atualizado para uma versão igual ou superior à versão 3.3.

Associado ao protocolo RTDE são apresentados alguns exemplos de aplicações desenvolvidas para *Python*, versão 2.7.11, com o objetivo de fornecer um ponto de partida mais fácil. Para este trabalho, dos exemplos fornecidos foram utilizados o *record.py* e o *example_ploting.py*. O *script record.py* permite guardar os dados de saída do robô num ficheiro de formato separado por vírgulas (CSV), sendo necessário definir o IP do robô, o número da porta e o número de dados que se pretende guardar. Caso não seja definido um número limite este, por defeito, guarda continuamente os dados até ser forçada a paragem da aquisição. O *script example_ploting.py* permite fazer a leitura e representação gráfica de determinados dados obtidos do ficheiro CSV gerado pelo *record.py* de uma forma simples.

Para ser possível a aquisição de dados durante o movimento do braço robótico, o *script* utilizado para o envio de comandos ao robô tem de correr em simultâneo com o *record.py*. Para isto ser possível, ambos os scripts foram colocados dentro de diferentes funções num só *script* e para executar as duas funções em paralelo foi utilizado um sistema de *threads* (Anexo A.1).

Depois de se executar o movimento desejado e ser possível fazer a recolha de dados, foram feitas as medições e cálculos necessários para o desenvolvimento do controlador *Posicast*. As posições inicial e final da extremidade do braço robótico na trajetória são apresentadas na Tabela 6.1.

Tabela 6.1: *Especificação das posições.*

Posição inicial (mm)	Posição final (mm)
x = 420,71	x = 420,73
y = 274,67	y = -265,62
z = 431,53	z = 431,57

Como se pode verificar na Tabela 6.1, o movimento ocorre paralelamente ao eixo dos YY no sentido negativo, mantendo-se os outros valores praticamente constantes. No entanto, ao longo desta fase, para efetuar os cálculos necessários para encontrar a posição intermédia de paragem, mantendo a coerência com as fórmulas referidas anteriormente, o eixo YY é denominado de XX .

O comprimento do pêndulo (L) é de $330{,}00\,mm$. Inicialmente, numa fase de testes, o ângulo máximo alcançado pelo pêndulo (θ) foi obtido utilizando uma câmara para filmar a trajetória e isolar a *frame* em que a extremidade do pêndulo se encontrava na posição mais avançada, obtendo-se um valor de aproximadamente 45°. Após efetuar os cálculos necessários, a posição intermédia de paragem obtida foi $x_p = -103{,}33mm$. No entanto, de forma a minimizar o erro introduzido pelo ângulo utilizado nos cálculos anteriores, foi implementado um código em *Matlab* (Anexo B) para traçar o gráfico da variação do ângulo ao longo do movimento, utilizando um *Arduino* para fazer a aquisição dos dados da tensão no potenciómetro acoplado ao pêndulo e converter esses dados para ângulos, sendo que, a tensão obtida quando o pêndulo se encontra na posição de repouso equivale a um ângulo de 0º. Na Figura 6.2 é apresentada a montagem do sistema que permite fazer a aquisição dos dados necessários em tempo de execução e são realçados o *Arduino* e o potenciómetro.

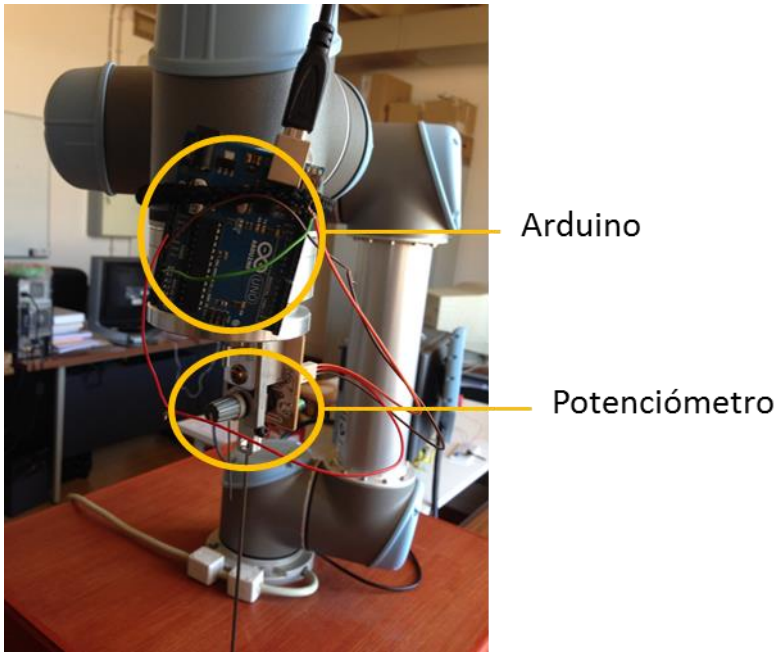

Figura 6.2: *Montagem do sistema.*

Após a execução do código (Anexo B) em paralelo com o movimento do braço robótico, foi obtida a seguinte figura.

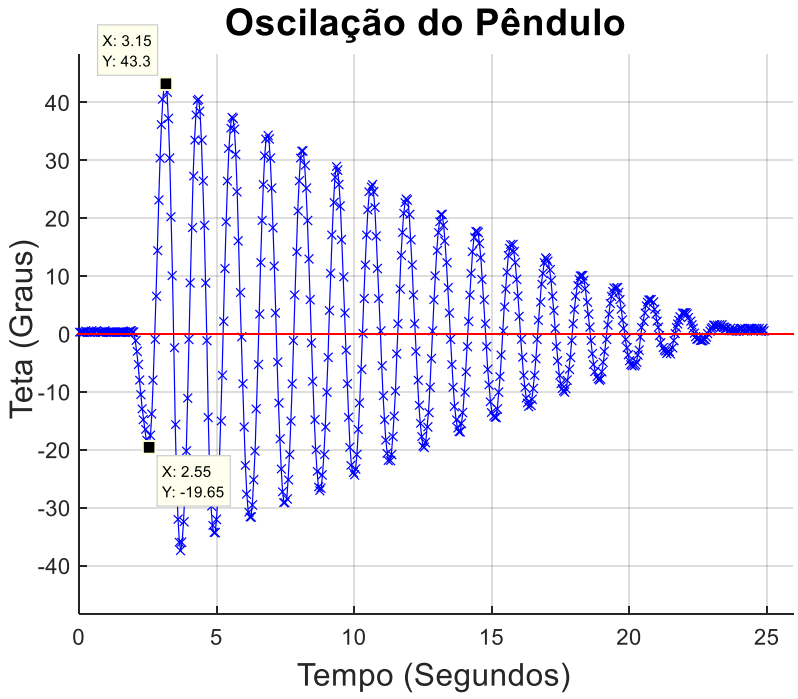

Figura 6.3: *Oscilação do pêndulo com um movimento simples.*

O ângulo máximo obtido por este método é de 43,3° (Figura 6.3), sendo este ligeiramente menor que o utilizado inicialmente. Foi então necessário repetir os cálculos com o novo ângulo obtido.

A distância percorrida pela extremidade do braço robótico e pelo pêndulo foram calculados por

$$x = x_i - x_f = 274,71 - (-265,62) = 540,29\,mm \tag{6.1}$$

$$x_{mf} = x + L\sin\theta = 540,29 + 0,33\sin 43,3 = 766,61\,mm \tag{6.2}$$

Para determinar a distância que o robô deve percorrer até atingir o ponto intermédio de paragem (x_p) foram executados os seguintes cálculos

$$\delta = e^{-\frac{\xi\pi}{\sqrt{1-\xi^2}}} = \frac{x_{mf} - x}{x} = \frac{766,61 - 540,29}{540,29} = 0,42 \tag{6.3}$$

$$A_1 = \frac{1}{1+\delta} = \frac{1}{1+0,42} = 0,7048 \tag{6.4}$$

$$A_2 = 1 - A_1 = 1 - 0,7048 = 0,2952 \tag{6.5}$$

Como $A_1 = 0,7048$, a distância a percorrer pelo braço robótico (Dx_p) até atingir o ponto intermédio de paragem (x_p) corresponde a $70,48\%$ da trajetória total

$$Dx_p = 540,29 \times 0,7048 = 380,80\,mm \tag{6.6}$$

Sabendo a distância a percorrer até atingir a posição x_p, facilmente se calcula a posição correspondente ao mesmo

$$x_p = x_i - Dx_p = 274,67 - 380,80 = -106,13\,mm \tag{6.7}$$

O tempo de paragem (T_p) na posição x_p foi obtido através da análise da Figura 6.3. O tempo necessário para que o pêndulo execute metade do período da primeira oscilação é de aproximadamente 0,6 s.

Após obter os valores de x_p e T_p implementou-se o código para o robô parar no ponto pretendido com o tempo desejado. Primeiro, para parar o robô no ponto intermédio, foram usados dois comandos de movimento *L*, sendo um para a posição de paragem calculada e outro para a posição final. Como os dois comandos não podem ser enviados em simultâneo, após enviar o primeiro foi utilizado um tempo de espera equivalente ao tempo necessário para que este atingisse o ponto x_p mais o tempo T_p. No entanto, por este método o ponto intermédio tem de ser enviado ao robô e o tempo em que este para não é muito preciso.

De forma a tornar o controlo do movimento do robô mais autónomo foi implementado outro método para forçar a paragem intermédia. O objetivo é fazer a leitura da posição do braço robótico ao longo do movimento, utilizando o *script record.py*, e quando este encontrar a posição pretendida seja enviado um comando para forçar a paragem. Como o *script record.py* apenas cria o ficheiro com os dados no final do movimento foi necessário editar o código proveniente do protocolo RTDE de forma a poder fazer a comparação das posições em tempo real. O protocolo RTDE possui um conjunto de classes, sendo que a classe *csv_writer.py* possui funções para receber os dados e guardar os mesmos num arquivo CSV. Para obter o resultado pretendido foram feitas alterações na função *writerow()* da classe *csv_writer.py* (Anexo A.2). Uma vez que os dados são adquiridos com um período de 0,008 segundo dificilmente é lido um valor exatamente igual ao calculado. Então, foi realizado um conjunto de leituras repetidas do movimento de forma a saber qual o erro máximo do valor mais próximo do calculado em relação ao valor pretendido, sendo depois estabelecido um intervalo para fazer a comparação.

Tabela 6.2: *Os três valores mais próximos obtidos na leitura dos dados pelo protocolo RTDE.*

1º Teste	-96,35	**-102,87**	-109,30
2º Teste	-96,40	**-102,96**	-109,34
3º Teste	-96,39	**-102,96**	-109,36

Após fazer uma análise aos valores obtidos e apresentados na Tabela 6.2, foi definido um intervalo entre -101 e -104 para fazer a comparação com os valores obtidos nas novas simulações. Com este intervalo obtém-se a posição mais próxima da calculada e garante-se que apenas um valor seja aceite como correto.

No *script* criado para executar o movimento com controlo *Posicast* em vez de enviar o comando para que o braço robótico se desloque para a posição intermédia, é enviado o comando para que se desloque diretamente para a posição final. Depois, o código entra num ciclo *while* infinito até que o robô atinja a posição desejada, utilizando para tal uma variável global para indicar a chegada. Quando é atingida a posição intermédia é enviado um comando para forçar a paragem do robô durante o tempo definido anteriormente e é enviado de seguida novamente o comando para movimentar o robô para a posição final (Anexo A.3). O código para forçar a paragem do movimento apenas durante o tempo desejado é apresentado no Código 6.3.

Código 6.3: *Código para forçar a paragem do robô.*

```
while True:

    from csv_writer import interrupt
    if interrupt != 0:
    break

stop = 'stopl(10)'+'\n'
s.send (stop.encode('utf-8'))
time.sleep(0.6)
```

O comando utilizado para indicar a paragem do movimento *L* do robô é o *stopl(a)*, que diminui a velocidade do movimento consoante o valor da aceleração, ou seja, quanto maior o valor da aceleração menor é o tempo que o robô necessita para parar por completo. No entanto, como a paragem do movimento não é instantânea, foi necessário ajustar a posição no qual se inicia a paragem do movimento. Foi escolhida uma aceleração de $10{,}5\, m/s^2$, para que a travagem seja rápida, mas não demasiado ao ponto de danificar o robô. Após fazer a simulação, foi traçado o gráfico da variação da posição da extremidade do braço robótico (em relação à coordenada y) durante o movimento com o *script example_plloting.py* e foi verificado que com a aceleração estabelecida para a paragem o robô finaliza a travagem com um desfasamento de 44,25 mm do ponto pretendido. Foi ajustado o intervalo estabelecido anteriormente para um intervalo entre -55 e -63 de forma a isolar o valor mais próximo de y = -61,87 e iniciar a paragem neste ponto, permitindo assim que o robô pare o mais próximo possível da posição pretendida.

Os gráficos da variação do TCP com o movimento simples e com o movimento com controlo *Posicast* encontram-se representadas na Figura 6.4 a) e b), respetivamente.

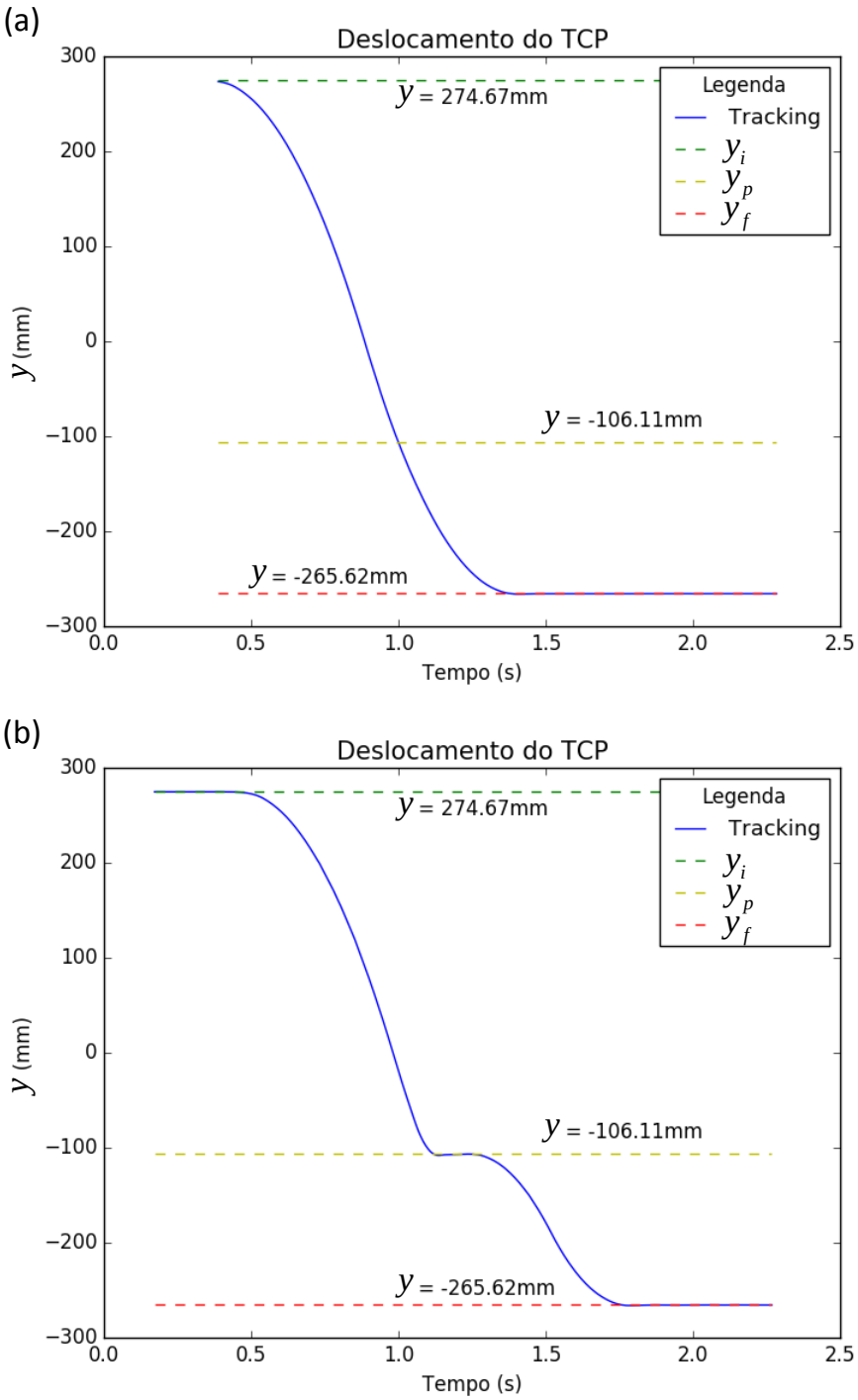

Figura 6.4: *Variação da posição da extremidade do braço robótico a) com um movimento simples e b) com um movimento com controlo Posicast.*

6.1.1 - Resultados

Após a implementação dos códigos necessários para executar os movimentos desejados, foram traçados os gráficos da oscilação e da variação da posição do extremo do pêndulo, para o movimento simples e para o movimento com controlo *Posicast*, de forma a permitir fazer a comparação entre eles e perceber qual a melhoria introduzida com o segundo movimento.

Com o movimento simples obteve-se a Figura 6.3, onde se pode verificar que o ângulo máximo atingido pelo pêndulo é 43,3° e o tempo necessário para que este estabilize na posição final é de aproximadamente 22 segundos. Os primeiros dois segundos não são considerados pois correspondem ao tempo entre a execução do código para gerar o gráfico e a execução do movimento do braço robótico.

Para o movimento com o controlo *Posicast* obteve-se o seguinte gráfico:

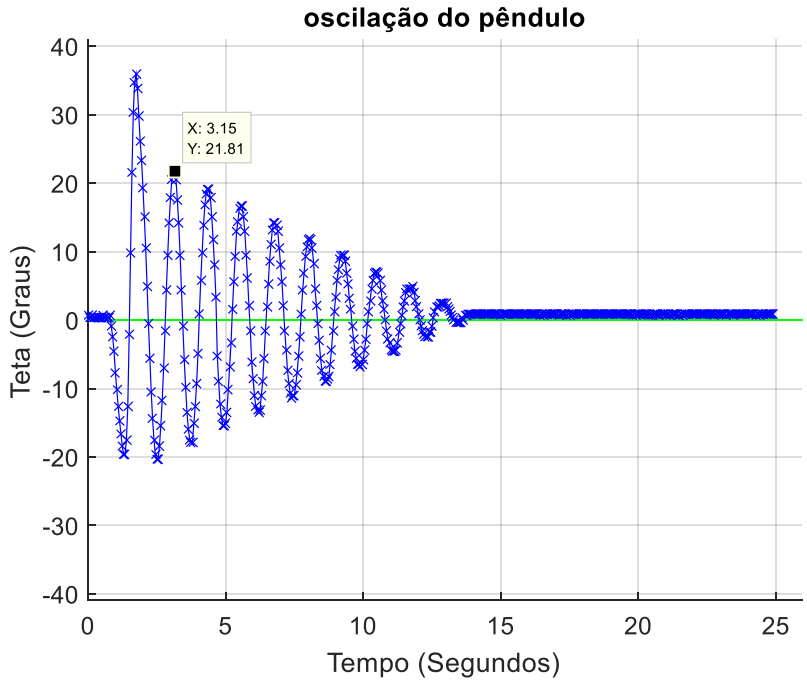

Figura 6.5: *Oscilação do pêndulo com controlo Posicast.*

Para poder fazer a comparação dos ângulos máximos atingidos na posição final nos dois movimentos é necessário fazer a leitura do segundo pico positivo para o movimento com

controlo *Posicast* como representado na Figura 6.5, pois o primeiro pico positivo ocorre no ponto intermédio. Pode-se então verificar que o ângulo máximo atingido na posição final é de 21,81° e o tempo necessário para que este estabilize na posição final é de aproximadamente 13 segundos. Com a implementação do controlo *Posicast* é verificada uma melhoria significativa, pois existe uma diminuição de 49,6% no ângulo máximo atingido na posição final, implicando uma diminuição de nove segundos para estabilizar a oscilação.

Para efetuar a comparação da variação da posição da extremidade do pêndulo com a oscilação foram criados novos *scripts* para traçar o gráfico apenas quando o robô atinge a posição final, permitindo assim analisar os desvios máximos alcançados pela extremidade do pêndulo. Para o movimento simples o gráfico só é traçado a partir da primeira variação positiva do ângulo, pois é essa a condição que garante que atingiu a posição final. O gráfico resultante do movimento simples está representado na Figura 6.6.

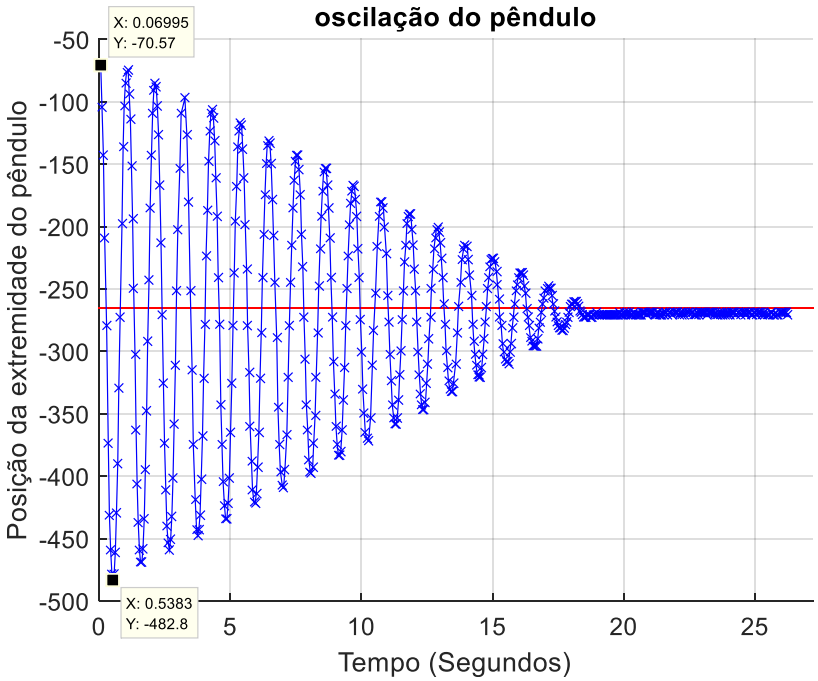

***Figura 6.6**: Variação da posição da extremidade do pêndulo com o movimento simples.*

Com este tipo de movimento a extremidade do pêndulo atinge a posição $y = -482{,}8\,mm$, afastando-se $217{,}18\,mm$ da posição final. Para o movimento com controlo *Posicast* o gráfico só é traçado a partir da segunda variação positiva do ângulo pois, contrariamente ao que acontece

no tipo de movimento anterior, com este movimento a primeira variação positiva ocorre quando é atingida a posição intermédia e a segunda quando é alcançada a posição final. O gráfico resultante do movimento com controlo *Posicast* está representado na Figura 6.7.

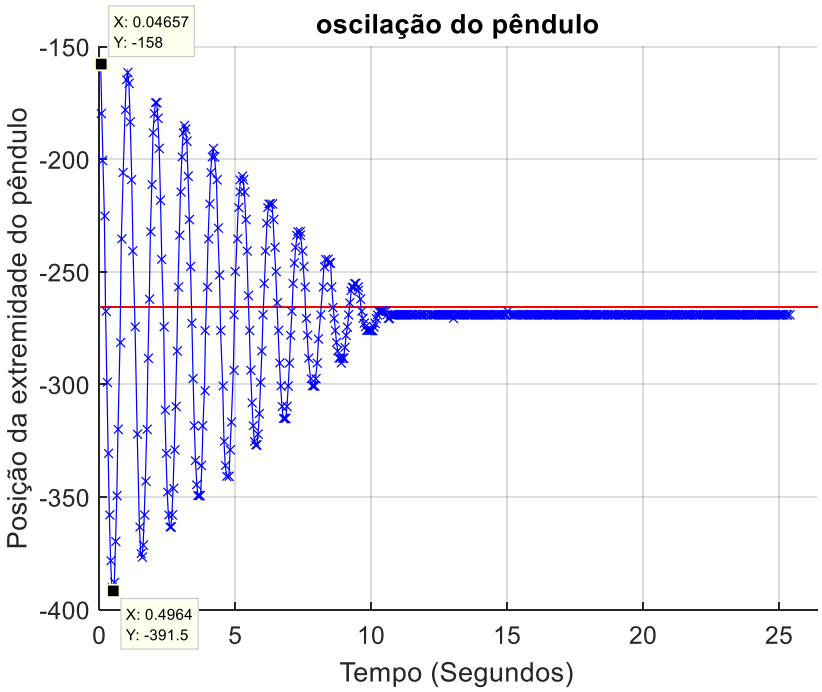

Figura 6.7: *Variação da posição da extremidade do pêndulo com um movimento com controlo Posicast.*

Com este tipo de movimento a extremidade do pêndulo atinge a posição $y = -391,5\,mm$, afastando-se $125,88\,mm$ da posição final. A introdução do controlo *Posicast* implica uma diminuição de 42,0% do afastamento do pêndulo relativamente à posição final. Embora o objetivo seja eliminar na totalidade a oscilação do pêndulo no final do movimento do braço robótico com a implementação do controlo *Posicast* não é possível obter uma diminuição de 100% da oscilação. É necessário ter em conta a própria dinâmica do robô UR5 que interfere com os valores calculados com base nos fundamentos teóricos do *Posicast*.

Os principais fatores responsáveis para que não sejam obtidos os resultados pretendidos são a posição intermédia (x_p) e o tempo de paragem (T_p) utilizados nessa posição. Como foi referido anteriormente, a posição em que o robô inicia a travagem para que o TCP pare na posição intermédia teve de ser ajustada consoante a aceleração de travagem utilizada e a distância percorrida para que interrompa por completo o movimento, o que implica que não haja precisão

da posição onde para o movimento. No entanto, a posição é sempre relativamente próxima da desejada sendo o erro introduzido baixo comparado com o que pode ser introduzido com a utilização de um tempo de paragem errado.

Teoricamente, com o controlo *Posicast*, o tempo de paragem para este caso seria de 0,6 segundo. No entanto esse valor foi ajustado para 0,15 segundo de forma a obter os resultados demonstrados. O tempo utilizado é bastante inferior ao estipulado porque a velocidade não é constante durante a execução de um movimento, ao contrário do que acontece com os "*carts*". No início do movimento a velocidade aumenta gradualmente até atingir o valor introduzido e diminui da mesma forma antes de parar, sendo o perfil de velocidades aproximado a um trapézio. Como a velocidade diminui antes de atingir a posição intermédia, o pêndulo começa a oscilar ligeiramente antes da posição desejada, sendo então T_p a soma do tempo que o robô demora para se movimentar da posição em que o pêndulo começa a oscilar até à posição intermédia com o tempo que se mantém parado nessa posição. Na Figura 6.8 está representado o perfil de velocidades para a execução de uma movimentação.

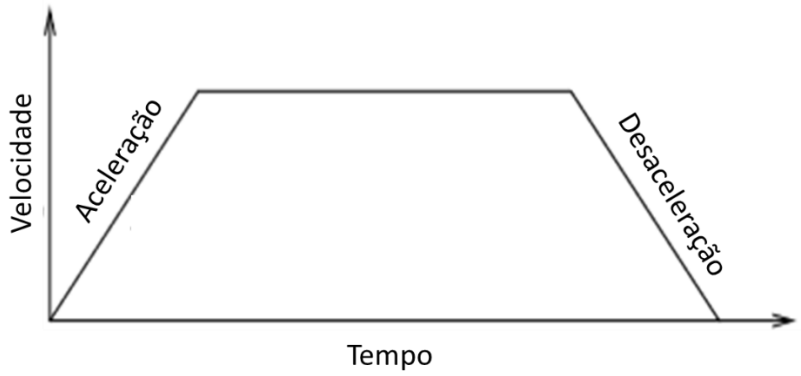

Figura 6.8 *Perfil de velocidades (Universal Robots, 2017a, adaptada).*

6.2 - Implementação do Controlo *Posicast* no Simulador

Para obter melhorias nos resultados, os parâmetros a ajustar são a posição intermédia (x_p) e o tempo de paragem (T_p) utilizados nessa posição, pois para o UR5 os valores utilizados na implementação não são ideais. De forma a tentar otimizar os parâmetros referidos foi utilizado um simulador desenvolvido em *Matlab*, que permite obter os gráficos correspondentes à deslocação da posição *y* do TCP, a variação da posição da massa suspensa e a variação do

ângulo ao longo do movimento simulado, para diferentes tempos de paragem com a posição intermédia fixa. Numa outra versão são traçados os mesmos gráficos, no entanto para diferentes posições intermédias e um tempo de paragem fixa. Para determinar o melhor tempo de paragem (ou posição na segunda versão) é utilizado o algoritmo PSO adaptado para o caso de estudo, desenvolvido também em *Matlab* ®.

6.2.1 - Resultados

O simulador pode ser utilizado de duas formas diferentes:

- Pode ser invocado com o valor do parâmetro desejado;
- Pode ser executado primeiro o algoritmo PSO que retorna o valor do parâmetro otimizado, invocando depois o simulador para testar esse valor.

Inicialmente, o simulador foi criado para testar diferentes tempos de paragem com uma posição de paragem fixa. Posteriormente foi criada uma nova versão adaptada para fazer o inverso, ou seja, para testar diferentes posições de paragem para um tempo de paragem fixo. Numa fase inicial foram utilizadas as duas versões do simulador sem a utilização do algoritmo PSO para perceber qual a influência da alteração dos parâmetros nos resultados:

- Para a primeira versão do simulador foram testados diferentes tempos de paragem entre 0 e 2 segundos com uma variação de 0,15 segundo entre eles e a posição de paragem igual á utilizada no robô. Com estes testes verificou-se que os resultados variaram significativamente com a alteração do tempo de paragem.
- Para a segunda versão do simulador foi utilizado um tempo de paragem de 0,15 segundo fixos e testadas várias posições de paragem diferentes. No entanto, verificou-se que com pequenas variações da posição de paragem não existe grande alteração nos resultados.

Para otimizar os resultados obtidos com o robô, o principal foco no decorrer deste trabalho foi a otimização do tempo de paragem, visto que este apresenta uma influência no resultado bastante superior do que a posição de paragem.

Os resultados obtidos com o simulador sem controlo *Posicast*, ou seja, com um tempo de paragem igual a 0 segundo estão apresentados na Figura 6.9.

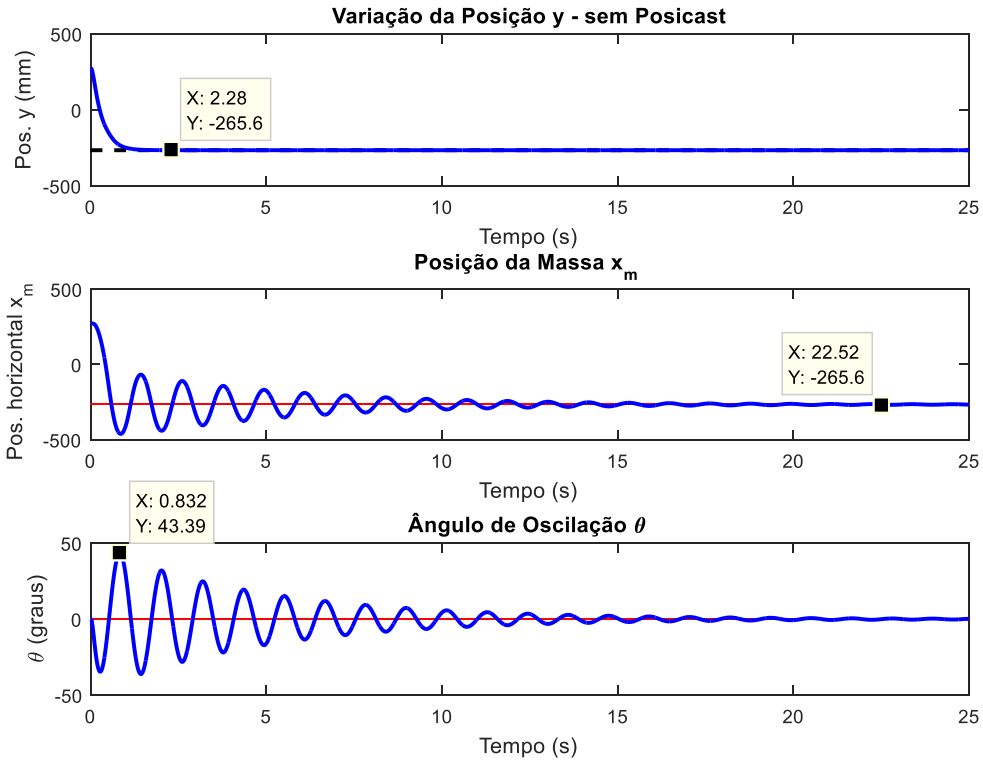

Figura 6.9: *Simulação do movimento simples.*

Para poder obter os resultados apresentados na Figura 6.9 foi necessário ajustar o fator de amortecimento e os ganhos K_0 e K_1, de forma a permitir uma simulação o mais realista possível. Inicialmente foi utilizado um fator de amortecimento igual a 0,3 e os ganhos sugeridos por Kufieta (2014),

$$Kd_0 = [69{,}0144\ 42{,}2086\ 23{,}9466\ 60{,}8311\ 93{,}1483\ 46{,}1087] \quad (6.8a)$$

$$Kd_1 = [69{,}6765\ 42{,}9010\ 24{,}3699\ 61{,}1925\ 94{,}1696\ 46{,}8422] \quad (6.8b)$$

onde Kd_0 e Kd_1 representam a diagonal principal das matrizes K_0 e K_1, respetivamente. No entanto, a utilização destes valores não apresentam os resultados desejados, sendo estes posteriormente alterados. O fator de amortecimento e os ganhos foram ajustados de forma a obter o ângulo máximo atingido pelo pêndulo na posição final e o tempo necessário para que o mesmo estabilize o mais próximo possível do obtido com o robô. O fator de amortecimento foi alterado para 0,17 e os ganhos K_0 e K_1 para os valores seguintes:

$$Kd_0 = \begin{bmatrix} 64 & 64 & 64 & 64 & 64 & 64 \end{bmatrix} \tag{6.9a}$$

$$Kd_1 = \begin{bmatrix} 20 & 20 & 20 & 20 & 20 & 20 \end{bmatrix} \tag{6.9b}$$

A simulação do movimento sem controlo *Posicast* apresenta um resultado bastante satisfatório, visto que o ângulo máximo atingido e o tempo necessário para que o pêndulo estabilize na simulação são idênticos aos obtidos com o robô. O ângulo máximo obtido com o robô foi, como já referido, 43,30º. Na simulação com controlo *Posicast* esse valor foi de 43,39º. Em relação ao tempo necessário para que o pêndulo estabilize, em ambos os casos rondam os 22 segundos, não sendo possível ler um valor exato através dos gráficos obtidos.

Para encontrar o tempo de paragem que permite obter a menor oscilação possível no final do movimento simulado e posteriormente comparar os resultados obtidos com os resultados alcançados com a utilização desse tempo no robô, foi utilizado o algoritmo PSO adaptado para este trabalho. A função objetivo utilizada é representada por:

$$ISE = \sum_{i=1}^{n} \theta_i^2 \tag{6.10}$$

onde θ_i é o ângulo do pêndulo para cada instante. A função ISE (Integral do erro quadrático) é uma medida de desempenho baseada na integração do erro quadrático ao longo do tempo. Para determinar o melhor tempo de paragem são inicializadas aleatoriamente 10 partículas num espaço de pesquisa limitado entre 0 e 2 segundos e o algoritmo PSO é executado ao longo de 50 iterações. Para cada iteração o algoritmo gera um gráfico com a distribuição das partículas no espaço de pesquisa, de forma a permitir ver e analisar a evolução das mesmas. Na Figura 6.10 estão representadas a inicialização e a distribuição aleatória do enxame (população) no espaço de pesquisa.

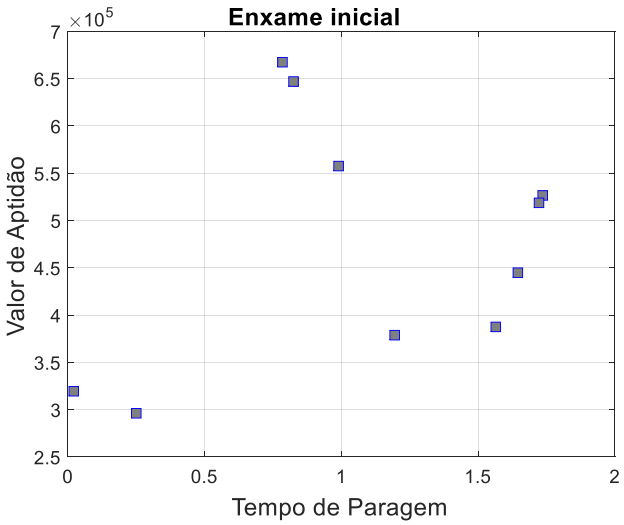

Figura 6.10: *Distribuição aleatória das partículas.*

Embora o algoritmo execute somente 50 iterações, foi possível verificar que logo a partir da iteração 8 é encontrado o valor ideal de 0,13 segundo, sendo que, nas iterações seguintes todas as partículas tendem a aproximarem-se dessa posição. Na Figura 6.11 está representada a distribuição das partículas após a execução de metade das iterações definidas.

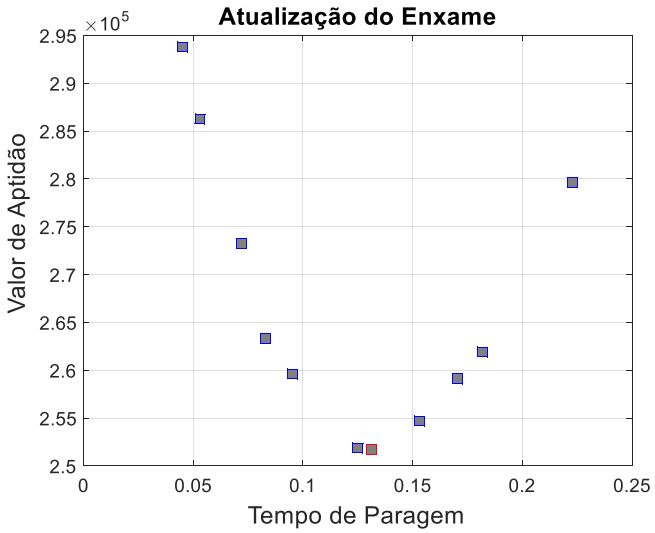

Figura 6.11: *Distribuição das partículas na iteração 25.*

Após a execução de 25 iterações é notória a evolução das partículas para a posição ideal. Na Figura 6.11 a partícula com melhor desempenho é representada com a cor vermelha e possui um valor para o tempo de paragem de 0,13 segundo. Ao contrário da distribuição da população inicial, na Figura 6.11 é possível verificar que as partículas já se encontram muito mais

próximas umas das outras, estando estas distribuídas num espaço de pesquisa entre 0 e 0,25 segundo. A aproximação das partículas aumenta com cada iteração até que todas encontrem a mesma posição. Na Figura 6.12 está representada a distribuição das partículas após as 50 iterações definidas inicialmente.

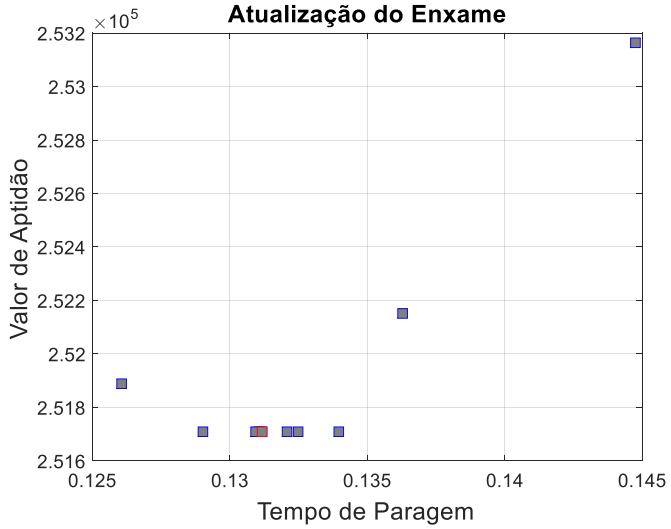

Figura 6.12: *Distribuição das partículas no final da execução das 50 iterações.*

No final da execução de todas as iterações é possível verificar que todas as partículas se encontram num espaço de pesquisa entre 0,12 e 0,15 segundo. De notar que, 7 das 10 partículas encontram-se aproximadamente no 0,13 segundo, sendo este valor considerado o valor ideal para o tempo de paragem. Se fosse executado um maior número de iterações seria esperado que todas as partículas atingissem a mesma posição. Para o caso em estudo, as 50 iterações definidas são suficientes para perceber qual o valor ótimo visto que desde da iteração 8 o valor ótimo não sofreu qualquer alteração.

Os resultados até aqui descritos foram obtidos utilizando apenas 10 partículas uma vez que só é feita a otimização de um parâmetro. A utilização de um número maior de partículas implica uma alocação mais elevada de recursos computacionais e um maior tempo de execução do algoritmo. Neste caso a utilização de um número maior de partículas é desnecessário, pois verificou-se que os resultados obtidos com um maior número de partículas são os mesmos que com apenas 10 partículas. Na Figura 6.13 estão representadas a evolução do melhor valor de aptidão encontrado e a média dos valores de aptidão das partículas ao longo de cada iteração, sendo apresentadas com uma linha contínua e uma linha tracejada, respetivamente.

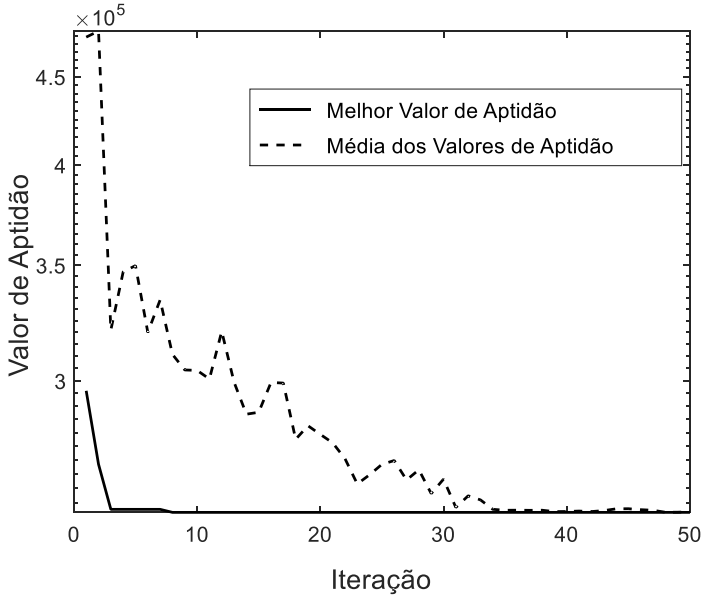

Figura 6.13: *Evolução do desempenho das partículas.*

Na Figura 6.13 é possível verificar que a partir da iteração 8 o valor ótimo para o tempo de paragem estabiliza, ou seja, a partícula com melhor valor de aptidão estabiliza numa posição no espaço de pesquisa. É também possível verificar que a partir da iteração 35 todas as partículas encontram-se próximas do valor ideal, pois a média dos valores de aptidão estabiliza no melhor valor de aptidão encontrado.

Após obter o valor considerado ótimo para o tempo de paragem (0,13 segundo) é executado o simulador com esse tempo. Quando o simulador executa uma simulação de um movimento com controlo *Posicast* retorna dois resultados sobrepostos, o resultado do movimento simples e o resultado do movimento com controlo *Posicast* para permitir uma melhor comparação entre eles. O resultado obtido com um tempo de paragem 0,13 segundo está apresentado na Figura 6.14.

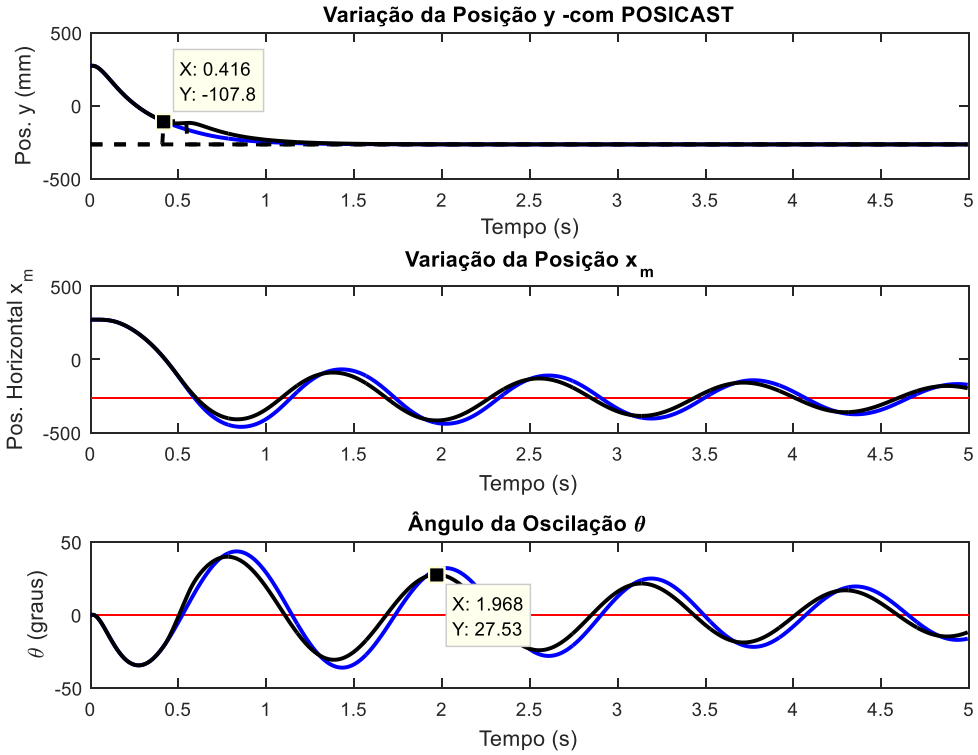

Figura 6.14: *Simulação do movimento com controlo Posicast.*

Na Figura 6.14 são apresentados apenas os primeiros 5 segundos do resultado obtido para permitir uma melhor análise dos resultados obtidos pelo simulador com um tempo de paragem de 0,13 segundo. A linha azul representa o resultado obtido com um movimento simples e a linha preta representa o resultado obtido com um movimento com controlo *Posicast*. Na Figura 6.14 é também possível verificar que quando é atingida a posição de paragem, que neste caso é em Y= -107,80 *mm*, o movimento para durante 0,13 segundo como desejado. Para poder fazer a análise da melhoria implementada com o controlo *Posicast* no simulador é necessário comparar o ângulo máximo atingido na posição final do movimento com os dois tipos de movimentos. Como já foi referido anteriormente, o ângulo máximo atingido como a simulação do movimento sem controlo *Posicast* é de 43,39º. Para a simulação do movimento com controlo *Posicast* é obtido um ângulo máximo de 27,53º, que corresponde ao segundo pico positivo do gráfico do ângulo de oscilação, pois o primeiro pico positivo corresponde ao ângulo máximo atingido na posição intermédia de paragem. Para ser mais percetível a diminuição do ângulo máximo atingido com o movimento com controlo *Posicast*, na Figura 6.15 são apresentados apenas os primeiros dois picos positivos da variação do ângulo para os dois tipos de movimentos e os ângulos referidos anteriormente.

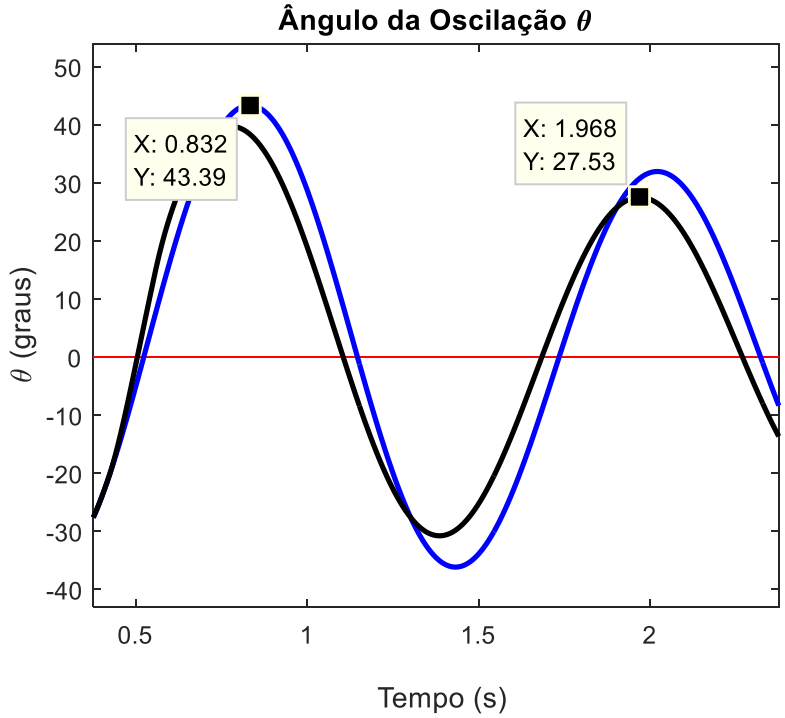

Figura 6.15: *Ângulos máximos atingidos com os dois tipos de movimentos.*

A introdução de um controlo Posicast no simulador implica uma diminuição da oscilação na posição final do movimento de 36,6%. Como foi utilizado o algoritmo PSO para encontrar o melhor tempo de paragem, esta é a diminuição máxima da oscilação na posição final com o simulador.

7 - Análise e Comparação dos Resultados

Com a utilização do algoritmo PSO é possível obter o tempo de paragem que apresenta melhores resultados no simulador, no entanto é preciso verificar se esse tempo também é o ideal para utilizar no robô. Inicialmente, no braço robótico foi utilizado um tempo de paragem igual a 0,6 segundo que posteriormente foi adaptado para 0,15 segundo por tentativa erro, não havendo garantias de que seja o melhor valor a usar. Com o valor de 0,15 segundo foi atingido um ângulo máximo de 21,81º na posição final e obteve-se uma diminuição de 49,6% da oscilação do pêndulo no final do movimento em relação à oscilação obtida com o movimento simples. Após a utilização do algoritmo PSO, para as mesmas condições, verificou-se que 0,13 segundo é o tempo de paragem que apresenta melhores resultados no simulador, sendo esse valor bastante próximo do utilizado com o braço robótico. Para avaliar se o valor obtido com o PSO é também o valor ideal na realidade, foi testado o tempo de paragem de 0,13 segundo na execução do movimento com controlo *Posicast* do robô UR5 e verificou-se que o ângulo máximo atingido na posição final diminuiu para 20,89º e a diminuição da oscilação no final do movimento passou a ser de 51,8%. Isto permite concluir que o valor obtido através do algoritmo PSO apresenta melhores resultados comparado com o valor utilizado anteriormente.

Todos os testes apresentados anteriormente foram executados utilizando um pêndulo com um comprimento de 0,33 m e uma massa de 0,260 kg. Embora para estas condições o valor obtido pelo algoritmo PSO apresente melhorias nos resultados, foi necessário testar o algoritmo para diferentes casos de forma a validar a eficácia do mesmo. Assim os testes foram replicados com condições diferentes. Para analisar a influência da massa nos resultados foram colocadas gradualmente três massas diferentes no pêndulo e para analisar a influência do comprimento do pêndulo foi testado outro pêndulo com um comprimento de 0,44 m e com uma massa de 0,544 kg, no qual também foram colocadas gradualmente três massas diferentes. Nas Tabelas 7.1 e 7.2 são apresentados os resultados referentes aos testes efetuados com os pêndulos de 0,33 m e 0,44 m, respetivamente.

Tabela 7.1: *Resultados com o pêndulo com comprimento de 0,33m.*

Massa	T_p (PSO)	Ângulo máximo atingido no final do movimento (Simulador)		Ângulo máximo atingido no final do movimento (UR5)		T_p (-) (UR5)	T_p (+) (UR5)
0,260 Kg	0,13 s	Mov. Simples 43,39º	Mov. Posicast 27,53º	Mov. Simples 43,30º	Mov. Posicast 20,89º	0,08 s Ângulo Máx.=22,12º Melhoria=48,9%	0,15 s Ângulo Máx.=21,81º Melhoria=49,6%
		Melhoria=36,6%		Melhoria=**51,8%**			
0,310 Kg	0,13 s	Mov. Simples 44,91º	Mov. Posicast 30,45º	Mov. Simples 43,61º	Mov. Posicast 23,34º	0,08 s Ângulo Máx.=23,65º Melhoria=45,8%	0,18 s Ângulo Máx.=25,49º Melhoria=41,6%
		Melhoria=32,2%		Melhoria=**46,5%**			
0,360 Kg	0,13 s	Mov. Simples 46,05º	Mov. Posicast 32,74º	Mov. Simples 43,92º	Mov. Posicast 24,26º	0,08 s Ângulo Máx.=25,49º Melhoria=42%	0,18 s Ângulo Máx.=26,72º Melhoria=39,2%
		Melhoria=28,9%		Melhoria=**44,8%**			
0,410 Kg	0,14 s	Mov. Simples 46,97º	Mov. Posicast 34,64º	Mov. Simples 44,84º	Mov. Posicast 25,49º	0,09 s Ângulo Máx.=26,11º Melhoria=41,8%	0,19 s Ângulo Máx.=29,18º Melhoria=34,9%
		Melhoria=26,3%		Melhoria=**43,2%**			

Tabela 7.2: *Resultados com o pêndulo com comprimento de 0,44m.*

Massa	T_p (PSO)	Ângulo máximo atingido no final do movimento (Simulador)		Ângulo máximo atingido no final do movimento (UR5)		T_p (-) (UR5)	T_p (+) (UR5)
0,544 Kg	0,21 s	Mov. Simples 39,37º	Mov. Posicast 27,27º	Mov. Simples 39,92º	Mov. Posicast 18,12º	0,15 s Ângulo Máx.=18,70º Melhoria=53,2%	0,25 s Ângulo Máx.=22,10º Melhoria=44,6%
		Melhoria=30,7%		Melhoria=**54,6%**			
0,594 Kg	0,21 s	Mov. Simples 39,76º	Mov. Posicast 28,11º	Mov. Simples 41,15º	Mov. Posicast 20,58º	0,16 s Ângulo Máx.=21,50º Melhoria=47,8%	0,26 s Ângulo Máx.=24,57º Melhoria=40,3%
		Melhoria=29,3%		Melhoria=**50,0%**			
0,644 Kg	0,21 s	Mov. Simples 40,01º	Mov. Posicast 28,80º	Mov. Simples 41,46º	Mov. Posicast 19,97º	0,16 s Ângulo Máx.=19,66º Melhoria=**52,6%**	0,26 s Ângulo Máx.=24,88º Melhoria=40,0%
		Melhoria=28,2%		Melhoria=51,8%			
0,694 Kg	0,22 s	Mov. Simples 40,39º	Mov. Posicast 29,43º	Mov. Simples 41,77º	Mov. Posicast 20,89º	0,17 s Ângulo Máx.=19,66º Melhoria=**52,9%**	0,27 s Ângulo Máx.=25,19º Melhoria=39,7%
		Melhoria=27,1%		Melhoria=50,0%			

Nas Tabelas 7.1 e 7.2 são apresentados, para cada massa, o melhor tempo de paragem obtido pelo algoritmo PSO, o ângulo máximo atingido no final do movimento simples e do movimento com controlo *Posicast*, tanto com o simulador como com o robô. É também apresentada a melhoria obtida em ambos os casos. Para poder verificar se o tempo de paragem obtido pelo algoritmo PSO é o ideal são também apresentados os resultados obtidos com o robô para um tempo ligeiramente inferior e outro superior ao obtido pelo algoritmo PSO.

Analisando os dados obtidos é possível verificar que a variação do comprimento do pêndulo tem uma maior influência no tempo de paragem do que a variação da massa, pois foi obtido um tempo de paragem quase constante para cada pêndulo, sendo apenas verificada uma variação de 0,01 segundo em ambos os casos com a variação da massa.

Para o pêndulo com um comprimento de 0,33 m é possível verificar que com o aumento da massa os ângulos máximos atingidos com o movimento simples e com o movimento com controlo *Posicast* aumentam também como esperado, tanto no simulador como no robô. No entanto, a melhoria que a implementação do controlo *Posicast* apresenta diminui com o aumento da massa nos dois casos. Para todos os testes executados com este pêndulo verificou-se que o melhor resultado é obtido com a utilização do tempo de paragem encontrado com o algoritmo PSO, visto que com um tempo ligeiramente inferior ou superior os resultados apresentam melhorias mais baixas.

Para o pêndulo com um comprimento de 0,44 m todos os resultados obtidos com o simulador têm o mesmo comportamento do que com o pêndulo anterior. Embora para este pêndulo tenham sido usadas massas superiores aos testes anteriores e os ângulos máximos obtidos tenham aumentado com o aumento da massa, foram obtidos ângulos inferiores em relação aos obtidos no caso anterior, o que permite concluir que com o aumento do comprimento do pêndulo, para além de aumentar o tempo de paragem, diminui o ângulo máximo atingindo na posição final. Os resultados obtidos com os testes realizados no robô com este pêndulo não são em todos os casos os esperados pois, embora para as duas primeiras massas testadas os resultados estarem coerentes com o comportamento verificado com o pêndulo anterior, nos últimos dois testes a variação dos dados deixa de ser previsível e o tempo de paragem que apresenta melhores resultados não corresponde ao obtido com o algoritmo PSO.

Todos os testes realizados foram executados mais que uma vez pois, como os dados dos ângulos são obtidos através do *Arduino*, nem sempre é captado o valor exato do ângulo máximo. A repetição dos testes nas mesmas condições permitiu perceber que com o pêndulo com comprimento de 0,44 m, o aumento da massa do mesmo provoca o aumento da incerteza do valor lido, principalmente nos últimos dois casos em que foi visível a introdução de uma vibração indesejada quando atingida a posição de paragem intermédia devido ao impacto da travagem, o que implica uma influência nos resultados. Embora o tempo de paragem obtido pelo algoritmo PSO não seja o ideal para os últimos dois testes executados, os resultados obtidos em geral foram bastante satisfatórios, pois mesmo nesses dois testes o tempo de paragem obtido é bastante próximo do ideal, sendo o erro de aproximadamente 0,05 segundo.

Nos resultados apresentados é ainda visível que os ângulos obtidos com o simulador não coincidem com os obtidos com o braço robótico. O simulador foi preparado para obter resultados idênticos ao robô com o movimento simples, no entanto quando é feita a simulação de um movimento com controlo *Posicast* os resultados desviam-se da realidade. Um dos principais fatores para que não sejam obtidos resultados idênticos entre o simulador e o braço robótico com o movimento com controlo *Posicast* é o facto de no braço robótico ser utilizado um comando para forçar a paragem na posição intermédia com o tempo desejado (*stopl(a)*), havendo um controlo da aceleração com que é feita a paragem e no simulador não ser tido em conta esse comando, sendo utilizada uma aceleração constante, igual para o arranque e para a paragem. Outro fator que influencia a diferença de resultados obtidos entre o simulador e o braço robótico é o facto de no simulador ser utilizado um perfil de velocidade igual para todas as articulações, o que não se verifica na utilização do braço robótico. Na Figura 7.1 está representada a variação do ângulo (em radianos) para cada articulação ao longo do movimento do braço robótico com controlo *Posicast*, sendo que para a obtenção da mesma foi desenvolvido um *script* em *Python* que utiliza os dados recolhidos através do protocolo RTDE.

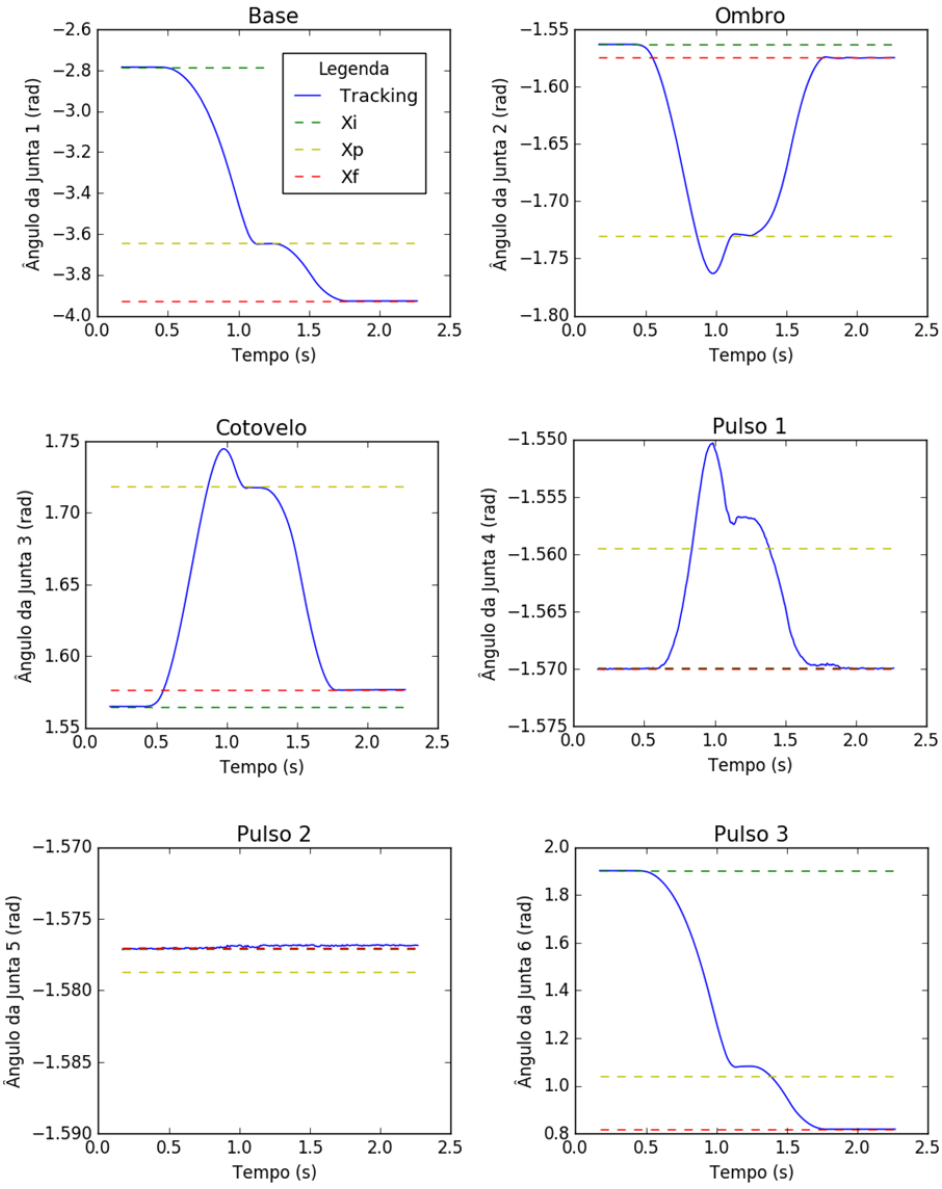

Figura 7.1: *Variação do ângulo de cada articulação com um movimento com controlo Posicast.*

Na Figura 7.1 é possível verificar que a variação do ângulo do pulso 1 e do pulso 2 é quase desprezável para a execução do movimento, sendo que a velocidade utilizada nos mesmos é baixa comparada com a utilizada nas outras articulações. Assim, verifica-se que as velocidades utilizadas em cada articulação não são sempre iguais, ao contrário do que é imposto no simulador. Para além destes últimos dois fatores referidos, existe ainda o facto de no simulador não ser feita a distinção dos tipos de movimentos utilizados, sendo que com a utilização de diferentes tipos de movimentos no braço robótico são obtidos resultados diferentes, pois as velocidades de execução alteram e existe um erro introduzido na oscilação menor ou maior

consoante o movimento, como já foi referido anteriormente. Com o UR5 foi utilizado um movimento do tipo *L*, o que implica um movimento mais complexo para manter a trajetória linear. Por outro lado, no simulador é utilizado um movimento do tipo *J* para executar a transição de um ponto para outro, pois não existe um controlo da movimentação das articulações individualmente para poder fazer restrições de forma a obter um movimento linear e isso pode introduzir erros nos resultados referentes aos ângulos obtidos.

Embora o simulador não apresente resultados exatamente iguais aos resultados obtidos com o robô, devido à complexidade que este apresenta, o simulador permite perceber a influência do movimento do UR5 para diferentes pêndulos acoplados à extremidade e prever o comportamento (se os ângulos melhoram ou pioram) com a alteração dos parâmetros utilizados no controlo *Posicast* no braço robótico. A utilização do simulador em conjunto com o algoritmo PSO permite perceber quais as condições que apresentam melhores resultados de uma forma mais rápida, pois sem a utilização nas mesmas seria necessário executar numerosos testes com o robô até encontrar as condições ideais.

8 - Conclusão

Neste trabalho foi feito o planeamento de trajetória para o braço robótico UR5, ao qual foi acoplado um pêndulo na extremidade, de forma a obter uma trajetória linear (ao longo de um dos eixos) que o mesmo executa com uma velocidade elevada criando o mínimo de oscilação possível do pêndulo na posição final. Para poder fazer o controlo do movimento do robô foi feita uma comunicação via *socket* para enviar os comandos e foi utilizado o protocolo RTDE para poder adquirir os dados relativos ao robô durante a sua movimentação.

Para diminuir o ângulo máximo atingido pelo pêndulo foi implementado um controlador *Posicast* de meio ciclo. O controlador teve de ser adaptado de forma a ter em consideração as características do robô. Na execução do movimento do braço robótico com o controlador *Posicast* obteve-se uma diminuição significativa do ângulo máximo da oscilação na posição final (49,6%), em relação ao movimento simples. Como os parâmetros do controlador *Posicast* tiveram de ser adaptados ao robô, os valores usados podem não ser os ideais e daí a necessidade de os otimizar permitindo assim uma melhoria dos resultados obtidos. Para otimizar os parâmetros do controlador *Posicast* foi utilizado o algoritmo de otimização por enxame de partículas em conjunto com um simulador que permite analisar a influência na variação dos seus parâmetros.

Com o simulador foi possível verificar que a variação do tempo de paragem tem uma influência nos resultados significativamente maior do que a variação da posição de paragem. Assim, foi utilizado o algoritmo PSO para otimizar o tempo de paragem assumindo que a posição de paragem utilizada é a ideal, pois permite a simplificação do algoritmo uma vez que apenas otimiza um parâmetro. Antes de ser feita a otimização do tempo de paragem foi utilizado um tempo que foi obtido por tentativa erro. Com o algoritmo PSO este valor foi otimizado sendo que, ao implementar esse valor otimizado no robô, o ângulo máximo atingido pelo pêndulo na posição final passa a ser significativamente menor e corresponde a uma diminuição de 51,8% em relação ao obtido com o movimento simples.

Para validar o valor obtido com o PSO foram executados diversos testes com comprimento do elo do pêndulo e massas suspensas diferentes, tanto no simulador como no robô. Foi possível constatar que o valor obtido pelo algoritmo PSO é na maioria dos casos testados o valor ideal a

usar. No entanto, quando o pêndulo apresenta características que provocam uma elevada vibração indesejada na execução do movimento, o valor obtido pelo PSO deixa de ser fiável uma vez que no algoritmo utilizado não foi considerado o erro introduzido pela vibração.

Com este trabalho foi desenvolvida uma metodologia que permite fazer o transporte de uma carga suspensa de uma forma rápida com o UR5, sendo obtida uma oscilação consideravelmente menor do que com um movimento direto da posição inicial para a final. Com a utilização do algoritmo PSO deixa de ser necessária a execução de numerosos testes para determinar o tempo de paragem, pois sem o PSO esse valor teria de ser encontrado por tentativa erro.

Como possibilidades de trabalho futuro propõem-se as seguintes linhas:

- Testar outros tipos de controladores;
- Integrar o sistema pêndulo-UR5 no simulador Gazebo (Gazebo, 2017), o qual corre sobre o ROS (*Robot Operating System*), de modo a simular o comportamento em modelo 3D.

Referências bibliográficas

Alavandar, S., Jain, T. e Nigam, M. J. (2009). Particle swarm optimized hybrid fuzzy precompensated trajectory control of rigid-flexible manipulator. *International Journal of Knowledge-based and Intelligent Engineering System*, 13, pp: 155-167.

Dhillon, B. S. e Yang, N. (1996). Availability analysis of a robot with safety system. *Microelectronics Reliability*, 36(2), pp: 169-177.

Ducatelle, F., Di Caro, G. A. e Gambardella, L. M. (2010). Principles and Applications of Swarm Intelligence for Adaptive Routing in Telecommunications Networks. *Swarm Intelligence*, 4(3), pp: 173-198.

Gazebo (2017). Consultado em 7 de Julho de 2017, disponível em: http://gazebosim.org/

Høifødt, H. (2011). Dynamic Modeling and Simulation of Robot Manipulators The Newton-Euler Formulation. Department of Engineering Cybernetics Norwegian University of Science and Technology.

Huang, H. C., Chen, C. P. e Wang, P. R. (2012). Particle Swarm Optimization for Solving the Inverse Kinematics of 7-DOF Robotic Manipulators. *IEEE International Conference on Systems, Man, and Cybernetics*, Seoul (KOR), Oct. 14-17, 2012.

Huey, J. R., Sorensen, K. L. e Singhose, W. P. (2008). Useful applications of closed-loop signal shaping controllers, *Control Engineering Practice*, 16(7), pp: 836-846.

Hung, H. Y. (2007). Posicast Control Past and Present. *IEEE Multidisciplinary Engineering Education Magazine*, 2(1), pp: 7-11.

Kennedy, J. e Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of the IEEE International Conference on Neural Networks*, 4, pp: 1942-1948.

Koller, H. (2015). Colaboração entre Homem e robô. Consultado em 14 de Outubro de 2016, disponível em: http://www.onoffre.com/artigos/2015/04/30/colaboracao-homens-robos

Kucera, V. e Hromcik, M. (2011). Delay-based input shapers in feedback interconnections, *Preprints of the 18th IFAC World Congress*, pp: 7577-7582.

Kufieta, K. (2014). Force Estimation in Robotic Manipulators: Modeling, Simulation and Experiments. Department of Engineering Cybernetics NTNU Norwegian University of Science and Technology.

Lei, Q. e Wisse, M. (2014). Fast grasping of unknown objects using force balance optimization. *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), Chicago (USA), Sept. 14-18, 2014.

Mathiassen, K., Fjellin, J. E., Glette, K., Hol, P. K. e Elle, O. J. (2016). An Ultrasound Robotic System using the Commercial Robot UR5. *Frontiers in Robotics and AI*, (3)1, pp: 1-2.

Medeiros, J. A. C. C. (2005). Enxame de Partículas Como Ferramenta de Otimização em Problemas Complexos de Engenharia Nuclear. Universidade Federal do Rio de Janeiro.

Montes, L. (2015). Robots colaborativos para las fábricas de PSA, BMW y Volkswagen. *El Mundo*. Consultado em 14 de Outubro de 2016, disponível em: http://www.elmundo.es/economia/2015/03/26/551306d9ca4741c0368b4570.html

Müller, R., Vette, M. e Scholer, M. (2014). Inspector Robot – A new collaborative testing system designed for the automotive final assembly line. *CIRP Conference on Assembly Technologies and Systems*, pp: 59-64.

Myhre, T. A. e England, O. (2016). Tracking a Swinging Target with a Robot Manipulator using Visual Sensing. *Modeling, Identification and Control*, 37(1), pp: 53–62.

Oliveira, P. B. M., Pires, E. J. S. e Cunha, J. B. (2011). Particle Swarm Optimization for Gantry Control: A Teaching Experiment, *EPIA'2011, XV Portuguese Conference on Artificial*

Intelligence, Lisboa (POR), Oct. 2011. *Lecture Notes in Artificial Intelligence, LNAI7026, Springer-Verlag*, pp: 196-207.

Oliveira, P. B. M. e Vrančić, D. (2012). Underdamped Second-Order Systems Overshoot Control. *IFAC Conference on Advances in PID Control - PID'12,* Brescia (ITA), March 28-30, 2012.

Oliveira, P. B. M. e Cunha, J. B. (2013). Cantry Crane Control: a Simulation Case Study. *2013 2nd Experiment@ International Conference (exp.at'13),* Coimbra (POR), Sept. 18-20, 2013.

Oliveira, P. B. M., Vrančić D. e Cunha, J. B. (2014). Posicast PID Control of Oscillatory Systems, *Controlo 2014-Procedings of the 11th Portuguese Conference in Automatic Control*, Porto (POR).

Oliveira, J., Oliveira, P. B. M., Pinho, T. M. e Cunha, J. B. (2017). Swarm-based Auto-tuning of PID Posicast Control for Uncertain Systems, *MED 2017 - 25th Mediterranean Conference on Control and Automation (MED),* Valletta (MLT), July 3-6, 2017, pp: 1299-1303.

Ostergaard, E. (2012). Lightweight Robot for Everybody. *IEEE Robotics & Automation Magazine,* 19(4), p: 17.

Parsopoulos, K. E. e Vrahatis, M. N. (2010). Particle Swarm Optimization and Intelligence: Advances and Applications. Information Science Reference - Imprint of: IGI Publishing Hershey, PA ©2010.

Ragazzon, M. R. P. (2012). Robot Manipulator Collision Handling in Unknown Environment without using External Sensors. Department of Engineering Cybernetics Norwegian University of Science and Technology.

Reynolds, C. W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*, 21(4), pp: 25-34.

Rini, D. P., Shamsuddin, S. M. e Yuhaniz, S. S. (2011). Particle Swarm Optimization: Technique, System and Challenges. *International Journal of Computer Applications*, 14(1), pp: 19-27.

Rocha, C. D. C. (2016). Sistema de Bancada Laboratorial para Tarefas Repetitivas. Dissertação de Mestrado Integrado em Bioengenharia. Faculdade de Engenharia do Porto.

Selvi, V. e Umarani, R. (2010). Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques. *International Journal of Computer Applications*, 5(4), pp:1-6.

Senol, M. A., Gözde, H., Taplamacioglu, M. C. e Arı, M. (2016). Design of Swarm Intelligence Based Optimal Controller to Direct Matrix Converter Used in Renewable Energy System. *IEEE International Conference on Renewable Energy Research and Applications (ICRERA)*, Brimingham (UK), Nov. 20-23, 2016.

Serapião, A. B. S. (2009). Fundamentos de Otimização por Inteligência de Enxames: Uma visão geral. *Revista Controle & Automação*, 20(3), pp: 271-304.

Siciliano, B. e Khatib, O. (2016). Springer Handbook of Robotics. Spring, p: 5.

Smith, O. J. M. (1957). Posicast Control of Damped Oscillatory Systems, *Proc. IRE*, 45(9), pp: 1249-1255.

Spong, M., Hutchinson, S. e Vidyasagar, M. (2006). Robot modeling and control. John Wiley & Sons, Inc.

Šuligoj, F., Jerbić, B., Švaco, M., Šekoranja, B., Mihalinec, D. e Vidaković, J. (2015). Medical applicability of a low-cost industrial robot arm guided with an optical tracking system. *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), Hamburgo (GER), Oct. 2, 2015.

Tavares, P. M. S. (2015). Planeamento de trajetória sem Manipuladores em ambientes industriais. Dissertação de Mestrado Integrado em Engenharia Eletrotécnica e de Computadores. Faculdade de Engenharia da Universidade do Porto.

Tsai, C. C., Hung, C. C. e Chang, C. F. (2014). Trajectory Planning and Control of a 7-DOF Robotic Manipulator. *International Conference on Advanced Robotics and Intelligent Systems*, Taipei (TWN), June 6-8, 2014.

Universal Robots (2017a). *User Manual UR5 (3).*

Universal Robots (2017b). Consultado em Março 15, 2017 em: https://www.universal-robots.com

Universal Robots (2017c). Consultado em Fevereiro 13, 2017 em: https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/real-time-data-exchange-rtde-guide-22229/

Vrančić, D. e Oliveira, P. B. M. (2012). Design of feedback control for underdamped systems. *IFAC Conference on Advances in PID Control. - PID'12,* Brescia (ITA), March 28-30, 2012.

Waintraub, M. (2009). Algoritmos Paralelos de Otimização por Enxame de Partículas em Problemas Nucleares. Tese de doutoramento em Engenharia Nuclear. Universidade Federal do Rio de Janeiro.

Waltl, H. (2015). Nova cooperação entre homens e robôs nos processos de produção da Audi. Audi, Media Info, p: 1.

Wang, X., Song, R., Liu, X., Li, Q., Cheng, T. e Xue, Y. (2015). System Design For Orthognathic Aided Robot. *IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems,* Shenyang (CH), June 8-12, 2015.

Wu, Q., Cole, C. e McSweeney, T. (2016). Applications of particle swarm optimization in the railway domain. *International Journal of Rail Transportation*, 4(3), pp: 167-190.

Anexos

A - Códigos desenvolvidos em *Python* para executar o movimento do braço robótico e fazer a aquisição dos dados

Os códigos anexados nesta secção apenas funcionam se estivem na mesma pasta em conjunto com os restantes códigos do protocolo RTDE disponibilizados pela Universal Robots (2017c).

A.1 - Movimento simples

```
import thread
import socket
import time
import struct
import math
import argparse
import logging
import sys
import rtde as rtde
import rtde_config as rtde_config
import csv_writer as csv_writer

from csv_writer import interrupt
ciclo = 0

#---------- Função para enviar os comandos de deslocação ao robô ----------------
def mov():

  #------------- Estabelecer ligação-----------------------------------------------------

      Robot_IP = "192.168.10.202" # Ip do robô.
      Port = 30003; # Porta utilizada pelo robô.
      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      s.connect((Robot_IP, Port))

  #-----------------------Definir trajetória-------------------------------------------
      ciclo = 0
      if ciclo == 0:
         try:
            # -----------------Deslocar para o ponto final----------------------
            a=str(math.radians(-225.07))
            b=str(math.radians(-90.23))
            c=str(math.radians(90.32))
            d=str(math.radians(-89.95))
```

```
            e=str(math.radians(-90.35))
            f=str(math.radians(46.87))

            PosB = 'movel(['+a+','+b+','+c+','+d+','+e+','+f+'], a=2, v=5)'+'\n'
            s.send (PosB.encode('utf-8'))
            ciclo = 1
        except socket.error as socketerror:
            print('Error: ', socketerror)
            s.close()
```

#---------- Função *record.py* do protocolo RTDE (cópia)----------

```
def read():

    #parameters
    parser = argparse.ArgumentParser()
    parser.add_argument('--host', default='192.168.10.202', help='name of host to connect to
(192.168.10.202)')
    parser.add_argument('--port', type=int, default=30004, help='port number (30004)')
    parser.add_argument('--samples', type=int, default=0, help='number of samples to
record')
    parser.add_argument('--config', default='record_configuration.xml', help='data
configuration file to use (record_configuration.xml)')
    parser.add_argument('--output', default='robot_data.csv', help='data output file to write to
(robot_data.csv)')
    parser.add_argument("--verbose", help="increase output verbosity", action="store_true")
    args = parser.parse_args()

    if args.verbose:
        logging.basicConfig(level=logging.INFO)

    RTDE_PROTOCOL_VERSION = 1

    conf = rtde_config.ConfigFile(args.config)
    output_names, output_types = conf.get_recipe('out')

    con = rtde.RTDE(args.host, args.port)
    con.connect()

    # get controller version
    con.get_controller_version()

    # set protocol version
    if not con.negotiate_protocol_version(RTDE_PROTOCOL_VERSION):
        logging.error('Unable to negotiate protocol version')
        sys.exit()

    # setup recipes
    if not con.send_output_setup(output_names, output_types):
```

```
            logging.error('Unable to configure output')
            sys.exit()

        #start data synchronization
        if not con.send_start():
            logging.error('Unable to start synchronization')
            sys.exit()

        with open(args.output, 'w') as csvfile:
            writer = csv_writer.CSVWriter(csvfile, output_names, output_types)
            writer.writeheader()

            i = 1
            keep_running = True

            while keep_running:

                if i%125 == 0:
                    if args.samples > 0:
                        sys.stdout.write("\r")
                        sys.stdout.write("{:.2%} done.".format(float(i)/float(args.samples)))
                        sys.stdout.flush()
                    else:
                        sys.stdout.write("\r")
                        sys.stdout.write("{:3d} samples.".format(i))
                        sys.stdout.flush()
                if args.samples > 0 and i >= args.samples:
                    keep_running = False
                try:
                    state = con.receive()
                    if state is not None:
                        writer.writerow(state)
                    else:
                        sys.exit()
                except KeyboardInterrupt:
                    keep_running = False
                i += 1

        sys.stdout.write("\rComplete!          \n")

        con.send_pause()
        con.disconnect()

#------- Utilização de trheads para correr as funções anteriores em "simultâneo" ----------
try:
   thread.start_new_thread(mov,())
   thread.start_new_thread(read,())
except:
   print "Error: unable to start thread"
```

A.2 - Função *writerow()* da classe *csv_writer.py* editada para fazer a comparação de dados em tempo real

```
import csv

import serialize

interrupt = 0

class CSVWriter(object):

(…)

    def writerow(self, data_object):

        global interrupt # Variável para indicar o ponto de paragem

        data = []
        for i in range(len(self.__names)):
            size = serialize.get_item_size(self.__types[i])
            value = data_object.__dict__[self.__names[i]]

            #----------------------Parte editada-------------------------------

            if  i == 10:
                y = value.pop(1) # Retira o primeiro elemento da lista value (valor da pos Y)
                y=y*1000          # Passa o valor para mm

                if y < -55 and y > -63:
# Filtra o valor mais próximo do ponto intermédio calculado contando com a distância
#percorrida na paragem
                    interrupt = y # interrupt assume o valor do ponto intermédio a utilizar

                y=y/1000             # Para pôr como foi retirado da lista
                value.insert( 1, y) # Volta a inserir o elemento na lista

            #-----------------------------------------------------------------

            if size > 1:
                data.extend(value)

            else:
                data.append(value)
        self.__writer.writerow(data)
```

A.3 - Movimento com controlo *Posicast*

O código para executar o movimento com controlo *Posicast* é igual ao código para executar o movimento simples (Anexo A.1), diferindo apenas na secção onde é definida a trajetória.

(…)

```
ciclo = 0

if ciclo == 0:
    try:

        #------------Definir trajetória--------------

        # -----------------Deslocar para o ponto final-----------------------

        a=str(math.radians(-225.07))
        b=str(math.radians(-90.23))
        c=str(math.radians(90.32))
        d=str(math.radians(-89.95))
        e=str(math.radians(-90.35))
        f=str(math.radians(46.87))

        PosB = 'movel(['+a+','+b+','+c+','+d+','+e+','+f+'], a=2, v=5)'+'\n'
        s.send (PosB.encode('utf-8'))

        # -----Código para fazer a paragem no ponto intermédio----------

        while True:
            from csv_writer import interrupt # Acede à variável interrupt do script csv_writer
            if interrupt !=  0:
# Se o valor for diferente de 0 quer dizer que já chegou a posição intermédia
                break

        stop = 'stopl(10.5)'+'\n'
# Comando para parar o movimento L com a aceleração de paragem associada
        s.send (stop.encode('utf-8'))
        time.sleep(0.13)

         # ---------Deslocar para o ponto final após paragem--------------------

        a=str(math.radians(-225.07))
        b=str(math.radians(-90.23))
        c=str(math.radians(90.32))
        d=str(math.radians(-89.95))
        e=str(math.radians(-90.35))
        f=str(math.radians(46.87))
```

```
            PosB = 'movel(['+a+','+b+','+c+','+d+','+e+','+f+'], a=2.2, v=5)'+'\n'
#Aceleração maior porque percorre uma distância mais curta
            s.send (PosB.encode('utf-8'))

            ciclo = 1  # Para sair do if

        except socket.error as socketerror:
            print('Error: ', socketerror)
            s.close()
```

(…)

B - Código desenvolvido em *Matlab* para obter o gráfico da oscilação

```
%%---------------------- Inicialização das variáveis-------------------------

a = arduino('COM3');   % Criar um objeto arduino
count = 1;             % Iniciar um contador
amostras = 501;        % Número de amostras pretendidas
tempo=0;

%%------------ Aquisição dos valores da tensão e calculo do Teta-------

  while count < amostras
    ti=tic;                          % Inicia um temporizador
    voltage = readVoltage(a, 'A0'); % Leitura da tensão
    tf=toc(ti);                      % Mede o tempo que demorou para adquirir a tensão

    disp(['Tensao: ',num2str(voltage)])

    Teta = (voltage-1.289)/0.0159;
% Calculo do angulo do pêndulo (considerando a posição de repouso 0º)
    disp(['Teta: ',num2str(Teta)])
    fprintf('\n')

    y(count)= Teta; % Para o teta corresponder ao valor no eixo dos YY
    VetorY(count) = [y(count)];
% Guarda todos os dados medidos para no fim fazer o gráfico contínuo

    drawnow

    tempo = tempo + tf;
    Tempo(count)=[tempo];
    count = count + 1;
  end
```

```
TetaMax = max(VetorY);
disp(['Angulo maximo: ', num2str(TetaMax)])

%%----------------- Configuração da janela para o gráfico-----------------

title('Oscilação do pêndulo');
xlabel('Tempo (Segundos)');
ylabel('Teta (Graus)');
grid on;
hold on;

plot([0 amostras], [0 0], 'r')

ylim([-TetaMax-5 TetaMax+5]) % Limite imposto para a variação do ângulo máximo
xlim([0 tempo+1]); % Limite para o tempo máximo necessário para a aquisição dos dados
plot(Tempo,VetorY, 'X-b') % Gráfico da oscilação do pêndulo ao longo do movimento
```