Faculdade de Engenharia da Universidade do Porto

Concepção e Implementação de uma Estratégia de Controlo Aplicada a um Sistema Robótico do tipo *Master/Slave*

Mário Jorge Moreira Cardoso

Licenciado em Engenharia Electrotécnica – Electrónica e Computadores pelo Instituto Superior de Engenharia do Porto

Dissertação submetida para a satisfação parcial
dos requisitos exigidos para a obtenção do grau de mestre em Automação, Instrumentação e Controlo

Porto, Janeiro de 2009

Dissertação realizada sob a orientação do
Professor Doutor António Mendes Lopes,
do Departamento de Engenharia Mecânica e Gestão Industrial
da Faculdade de Engenharia da Universidade do Porto

Agradecimentos

Ao Prof. António Mendes Lopes, meu orientador, pela sua disponibilidade e ajuda no desenvolvimento desta dissertação.

Aos meus pais e irmã, pelo apoio e compreensão que demonstraram durante a execução desta tese.

Aos meus amigos em geral, destacando o Miguel e o Diego, pela sua colaboração. À Helena pelas correcções na redacção do texto.

Resumo

Os sistemas de telemanipulação surgiram em aplicações como a construção no espaço e a desactivação de instalações nucleares, para permitir o controlo do movimento de um manipulador robótico localizado num ambiente remoto.

Os sistemas de telemanipulação possuem importantes vantagens na execução de tarefas em ambientes desconhecidos, impossíveis ou difíceis de realizar por um operador humano, e onde é impossível a utilização de sistemas robóticos autónomos devido às suas limitações na área da inteligência artificial, difícil interpretação de dados dos sensores, e difícil modelação do ambiente.

Um sistema robótico de telemanipulação é composto por dois subsistemas: *master* e *slave*. Tal sistema permite a um operador, agindo localmente sobre o robô *master*, interagir com ambientes remotos, por intermédio do robô *slave*. Tarefas que podem beneficiar de um tal sistema são, por exemplo, a manipulação de ambientes hostis e/ou de difícil acesso: espaço, meio subaquático, instalações nucleares e químicas, construção, tarefas de busca e salvamento, combate a incêndios, entretenimento e a cirurgia remota (telemedicina).

A presente dissertação descreve o desenvolvimento de um sistema robótico de telemanipulação. Numa primeira fase do trabalho foi necessário desenvolver e implementar uma arquitectura de controlo, de forma a explorar as potencialidades do robô *slave* (robô cartesiano). A segunda fase do trabalho foi dedicada ao estudo, desenvolvimento e implementação de uma estratégia de comando e interligação dos dois sistemas *master* e *slave*.

Abstract

Telemanipulation systems were firstly applied in nuclear decommissioning and constructions in space, giving the user direct motion control of a robotic manipulator located in a remote environment.

Telemanipulation can be used to accomplish a great number of tasks that are impossible or hazardous for direct human manipulation. It offers particular advantages in unknown environments where is not possible to use completely autonomous robotic systems due to limitation on artificial intelligence, sensor-data interpretation, and environment modeling.

A telemanipulation robotic system comprises two robotic subsystems: a master and slave. This type of systems allows the operator to interact with remotely environments, through a slave robot, by manipulating a master one. Telemanipulation has potential benefits in diverse applications, such as constructions in space; hazardous; undersea; nuclear; chemical and biological environments; in construction; military; archaeology; entertainment; medical (e.g. Minimally Invasive Surgery and telemedicine) areas; warehousing; fire fighting and lifesaving tasks.

The present dissertation describes the development of a telemanipulation robotic system. In a first stage of the work it was necessary to develop and implement a control architecture that explores all potentialities of the slave robot (cartesian robot). The second stage of the work was the study, development and implementation of a command strategy and interconnection of the master and the slave robotic systems.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

Capítulo 1

Introdução

Neste capítulo é efectuada uma introdução aos sistemas de telemanipulação. É feita uma breve referência a sistemas robóticos de telemanipulação existentes, com o intuito de divulgar as tecnologias empregues e as potenciais aplicações deste tipo de sistemas, revelando as suas características típicas e demonstrando as principais vantagens da sua utilização nas mais diversas áreas. Posteriormente é efectuada uma breve descrição do sistema robótico do tipo *master/slave* utilizado na realização desta tese, dando especial relevo às características principais deste, como sendo o tipo de estrutura utilizada, a sua configuração e a interligação entre os diversos elementos.

Neste capítulo são também enumerados os objectivos desta tese e apresentada a estrutura organizativa da mesma.

1.1 Sistemas Robóticos de Telemanipulação

A telemanipulação, por definição, corresponde à capacidade de realizar manipulação remota sobre ambientes e/ou objectos, o que consiste no controlo directo de um operador humano sobre um manipulador robótico, sendo que ambos estão situados em locais distintos. Um sistema robótico de telemanipulação é composto por dois subsistemas, *master* e *slave*, que estão separados fisicamente mas que interagem um com o outro através de um canal de comunicação.

Num sistema robótico de telemanipulação o nível de interacção entre o operador e o ambiente remoto pode ser diverso. O subsistema local *master* pode, simplesmente, comandar o subsistema remoto *slave*, não havendo *feedback* sensorial para o operador, ou quando muito, existindo apenas *feedback* visual. No caso de um sistema de telemanipulação unilateral, o operador movimenta o subsistema *master*, sendo que o *slave* deve cumprir as trajectórias de posição e de velocidade de referência provenientes do *master*, com ou sem factor de escala. Se existir contacto do manipulador remoto com o ambiente (acidental ou não), o operador não se apercebe do nível de interacção, sendo que as forças em jogo podem causar danos no ambiente ou no manipulador. Um sistema de telemanipulação bilateral, com *feedback* de força para o operador (com ou sem factor de escala) (Thompson, *et al.*, 2001; Sitti and Hashimoto, 2003), é assim fundamental, não só por questões de segurança, mas porque a tarefa a executar assim o exige (é essencial, por exemplo, na cirurgia remota) (Reiley, *et al.*, 2008).

O estudo dos sistemas de telemanipulação teve início nos anos 40, sendo que as primeiras aplicações foram desenvolvidas para o manuseamento de material radioactivo. Nos finais dos anos 40 e nos anos 50 um grupo de investigadores do *ANL* (*Argonne National Laboratory*) de Chicago, do qual se destaca o nome de *Raymond Geortz*, desenvolveu o primeiro sistema servo-eléctrico do tipo *master/slave* (figura 1.1) (Nuño Ortega e Basañez, 2006).

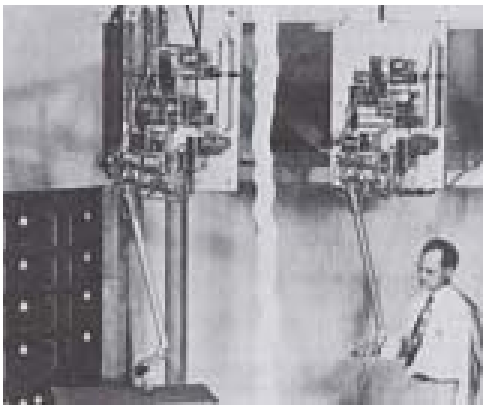
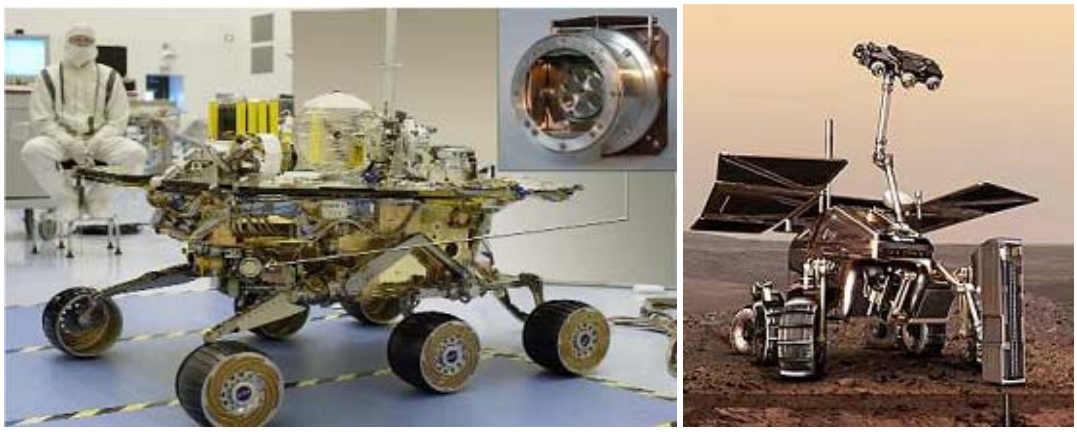


Figura 1. 1 – Primeiro sistema servo-eléctrico do tipo *master/slave* (Raymond Geortz)

Até ao presente, muitas foram as áreas que beneficiaram do uso de sistemas de telemanipulação, entre as quais se podem destacar a área da construção, militar, aeroespacial (Hirzinger, *et al.*, 1997), medicina (cirurgia remota) (Baumann and Clavel, 1998), trabalhos aquáticos, protecção civil (combate a incêndios, busca e salvamento), arqueologia, entretenimento, entre outros.

Os sistemas robóticos de telemanipulação têm sido de enorme importância na exploração e execução de trabalhos no espaço. São utilizados na exploração de planetas, como é o caso do Mars Exploration Rovers (MER) da NASA e o Exomars da ESA (figura 1.2), ambos desenvolvidos para a exploração do planeta Marte.



a) **b)**

Figura 1. 2 – a) MER da NASA; b) Exomars da ESA (NASA e ESA Copyrigth)

Outras aplicações dos sistemas robóticos de telemanipulação no espaço correspondem a trabalhos de inspecção, manutenção e reparação: de estações espaciais, satélites e *space shuttles* (figuras 1.3 e 1.4).

a) **b)**

Figura 1. 3 – a) Inspecção de um *Space Shuttle*; b) Braço Robótico aplicado a um Satélite (NASA Copyrigth)

Figura 1. 4 – Braço Robótico aplicado numa Estação Espacial (ESA Copyrigth)

Na área da medicina, as aplicações de sistemas robóticos de telemanipulação têm vindo a aumentar, especialmente na tele-cirurgia. Alguns casos mais conhecidos são os sistemas robóticos comerciais "Zeus" (figura 1.5) e "Da Vinci" (figura 1.6), que facilitam o trabalho dos cirurgiões.

Figura 1. 5 – Sistema Robótico de Telemanipulação "Zeus" (www.space.com)

Figura 1. 6 – Sistema Robótico de Telemanipulação "Da Vinci" (www.nextgenmd.org)

Estes sistemas robóticos comerciais, embora permitam movimentos de elevada precisão com um factor de escala em relação ao movimento manual do operador humano (Da Vinci – 5:1 e Zeus – 10:1), ainda não dispõem de *feedback* háptico (sensação ao toque) essencial para o cirurgião (Tavakoli, *et al.*, 2008). Nesse sentido vários são os trabalhos de investigação em curso na área da cirurgia assistida por sistemas robóticos, mais concretamente dos sistemas com *feedback* háptico.

Em Okamura (2004) pode verificar-se que foram efectuadas algumas experiências com dispositivos hápticos, no âmbito da cirurgia assistida por sistemas robóticos. Neste caso, nas experiências efectuadas, foram utilizados dois dispositivos hápticos *Phantom*, da empresa Sensable Inc. (EUA) (figura 1.7).

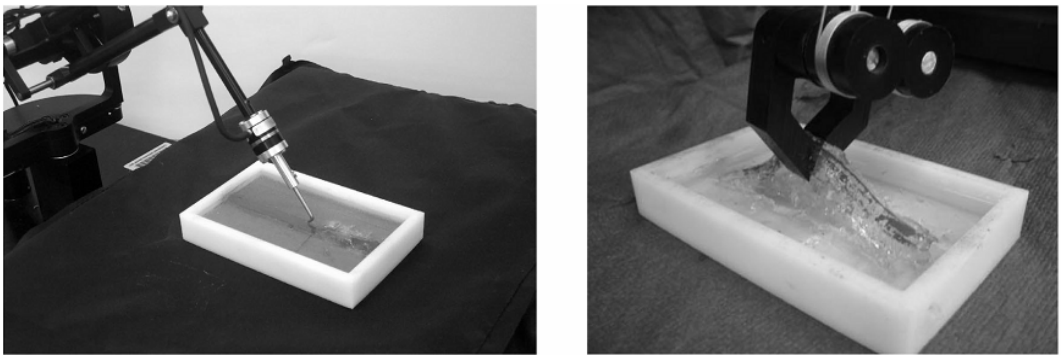


Figura 1. 7 – Exemplo de uma Experiência com Dispositivos Hápticos associados a Operações Cirúrgicas (Okamura, 2004)

Os trabalhos aquáticos são também uma das áreas onde se podem encontrar exemplos de aplicação de sistemas robóticos de telemanipulação. Um desses exemplos corresponde ao sistema comercial "Predator" (figura 1.8), que pode ser aplicado a sistemas de locomoção aquática (ROV – figura 1.9), facilitando o trabalho em zonas de difícil acesso, de elevada profundidade ou de elevado tempo de execução. O "Predator" consiste num braço robótico de actuação hidráulica, que dispõe de *feedback* de força, sendo controlado por um sistema *Human-Machine Interface* (*HMI*) (figura 1.8). A *HMI* (mini-master®) utilizada para controlar o "Predator" corresponde a uma inovação tecnológica, uma vez que disponibiliza ao operador o *feedback* de força associado à interacção entre o manipulador robótico e o espaço de trabalho.

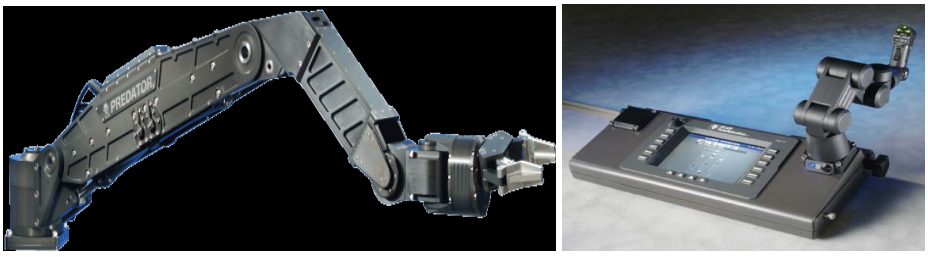


Figura 1. 8 – Sistema de Telemanipulação utilizando o Manipulador Robótico "Predator" e a *HMI* "mini-master®" (http://krafttelerobotics.com/products/predator.htm)

Figura 1. 9 – Exemplo de uma Aplicação do Manipulador Robótico "Predator" (http://krafttelerobotics.com/products/predator.ht)

Existem também vários trabalhos na área dos sistemas de telemanipulação no âmbito de projectos de investigação. De seguida, são apresentados dois exemplos com características próprias que vale a pena referir.

Em Nuño Ortega e Basañez (2006), é descrito um sistema de telemanipulação háptico com um controlo de impedância. O sistema robótico de telemanipulação (figura 1.10) é composto por um *Phantom 1.5 6 DOF* (*master*) e um robô industrial *TX-90 Stäubli* (*slave*) com um controlador *CS80-C Stäubli* e um sensor de força *JR3*. O sistema de telemanipulação dispõe de *feedback* de força (sensor de força *JR3*) e de *feedback* visual (duas câmaras de vídeo *CANON VC-C5*). Uma das características do sistema é o controlo de força implementado (controlo de impedância adaptado), uma vez que o algoritmo de controlo está desenvolvido de forma a adaptar-se às características do canal de comunicação.

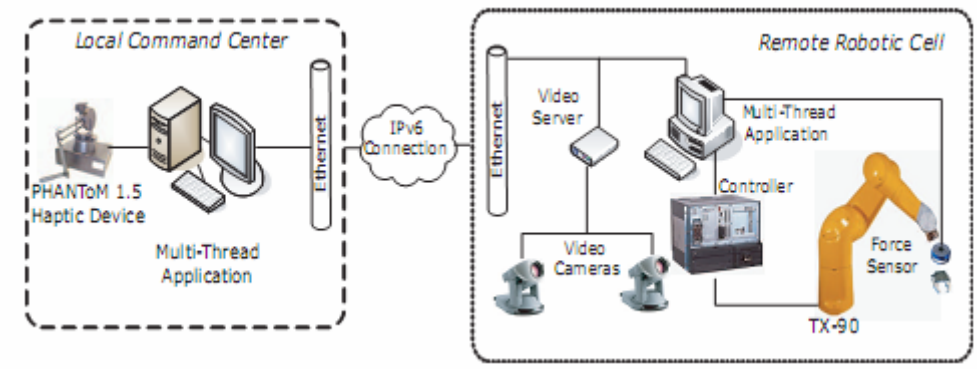


Figura 1. 10 – Exemplo de uma Arquitectura de um Sistema Robótico de Telemanipulação (Nuño Ortega e Basañez, 2006)

Conforme se pode verificar na figura anterior, foi utilizado um canal de comunicação para interligar ambos os sistemas robóticos. Esse canal de comunicação está inserido

numa *Local Area Network (LAN)* com uma estrutura do tipo cliente/servidor, que foi implementada através de *sockets*, utilizando os protocolos *TCP/UDP* e *IPv6*.

Em Filippi (2007), foi desenvolvido um sistema de telemanipulação que utiliza o *Wiimote*, *Nintendo*, como dispositivo *master*, sendo que o robô *slave* corresponde a um braço robótico *Lynx 6* (figura 1.11). A principal característica deste sistema corresponde ao canal de comunicação entre os dois sistemas, efectuado através da tecnologia *Wireless*, utilizando o protocolo de comunicação *Bluetooth*.

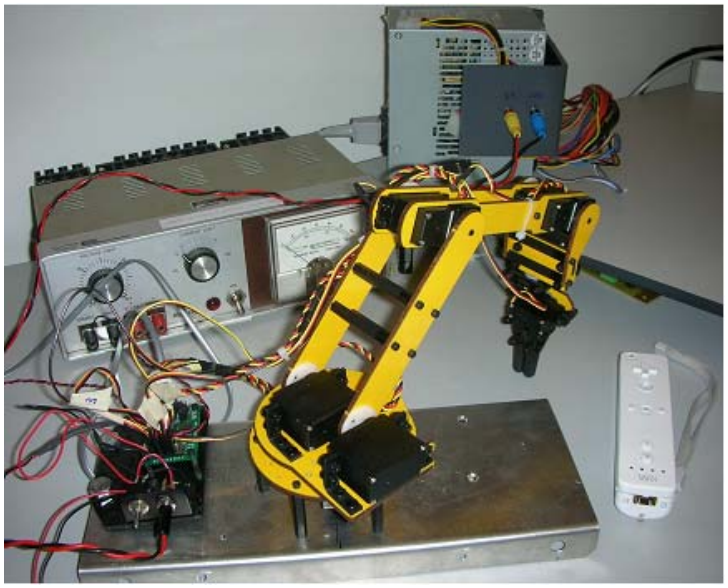


Figura 1. 11 – Exemplo de um Sistema de Telemanipulação via *Wireless* (Filippi, 2007)

Utilizando a *API Wiim*, que corresponde a um conjunto de classes C^{++}, foi possível criar uma *DLL*, que permite a integração dessa *API* em *Matlab*. Deste modo, é possível aceder aos parâmetros e comandos do dispositivo *Wiimote*.

O robô *slave* (*Lynx 6*) efectua a réplica dos movimentos do dispositivo *Wiimote*. Para isso foi implementado um controlo de seguimento de posição (controlo unilateral). O sistema de controlo corresponde a um *PC* onde está instalado o *Matlab*, que obtém as coordenadas do dispositivo *Wiimote*, sendo que estas são enviadas para os controladores de eixo (placa de circuito impresso com um microcontrolador) do robô *slave* via porta série.

1.2 Descrição do Sistema Robótico do Tipo *Master/Slave*

O sistema robótico de telemanipulação (figura 1.12) utilizado na realização desta tese é composto por dois subsistemas: *master* e *slave*. Tal sistema permite a um operador, agindo localmente sobre o robô *master* (*Phantom Haptic Device*), interagir com ambientes remotos, por intermédio do robô *slave* (Robô Cartesiano).

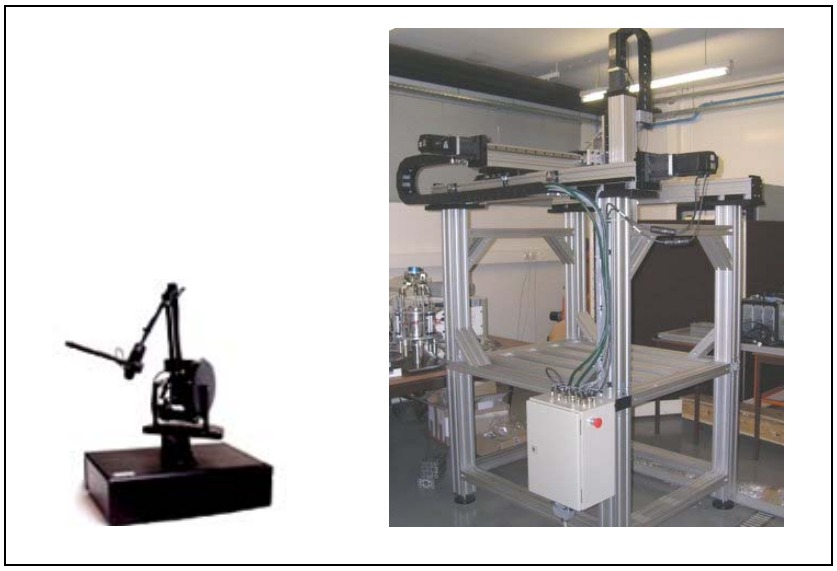


Figura 1. 12 – Sistema robótico do tipo *master/slave*

O robô *master* é um *Phantom Haptic Device* (figura 1.13), que possui uma estrutura mecânica muito particular, com 3 graus de liberdade e *feedback* de força. O controlador dedicado apresenta uma arquitectura *suficientemente* aberta, podendo ser acedido por *device drivers* disponíveis para o efeito. Na realização desta tese, este dispositivo é parte integrante do sistema robótico *master/slave*, mas a sua utilização também pode ser explorada na área da interacção com ambientes virtuais.

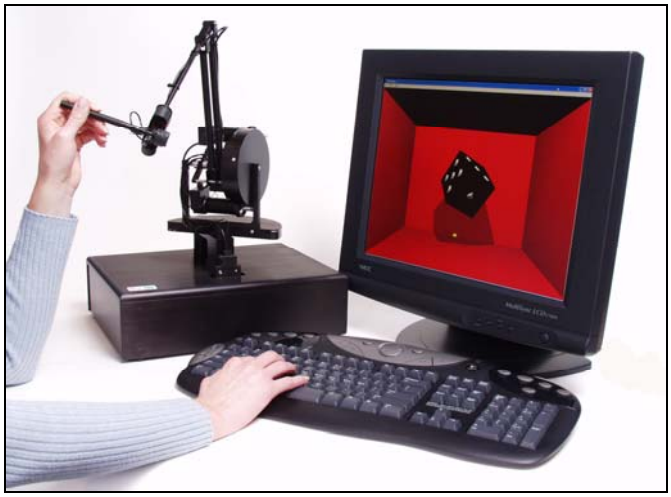


Figura 1. 13 – *Phantom Premium 1.5 High Force (master)*

O robô *slave* apresenta uma estrutura cartesiana (figura 1.14), com elevado espaço de trabalho e capacidade de carga. O seu controlador é baseado em *PC*, com uma arquitectura totalmente aberta, tendo sido desenvolvido no âmbito desta tese. Este robô pode operar autonomamente ou inserido no sistema robótico do tipo *master/slave*.

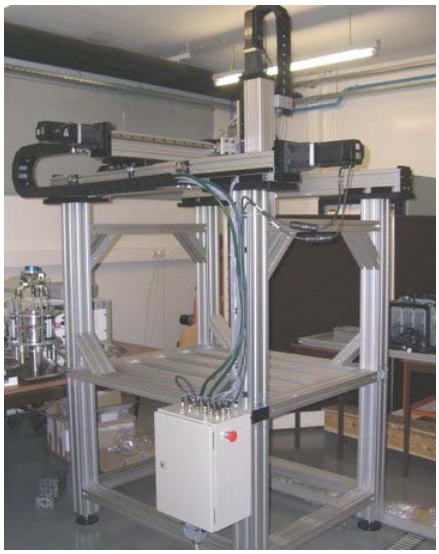


Figura 1. 14 – Robô Cartesiano de 3 Eixos (*slave*)

A arquitectura de controlo do sistema robótico baseia-se numa configuração *Host/Target*. O *Host PC* funciona como plataforma de desenvolvimento, bem como centro de supervisão e comando do *master*. O *Target PC* desempenha o papel de controlador do sistema robótico *slave*. O *Host PC* comunica com o *Target PC* através de uma ligação dedicada (cabo *crossover*).

1.3 Objectivos da Dissertação

Na primeira fase do trabalho pretendia-se desenvolver uma arquitectura de controlo (aplicação de controlo e interface gráfica de comando), que permitisse a operação do robô cartesiano de forma autónoma, sendo que este deveria apresentar as principais funcionalidades de um robô industrial.

Deste modo, em primeiro lugar efectuou-se o teste do hardware e o desenvolvimento de controladores de eixo robustos para o robô cartesiano, sendo necessário a simulação dinâmica do robô cartesiano e o desenvolvimento de algoritmos de controlo e sua implementação, cumprindo as seguintes etapas:

- Modelação do sistema físico e sua validação;
- Desenvolvimento de controladores robustos;
- Concepção de uma interface gráfica de comando;
- Testes, implementação e avaliação de desempenho.

Na segunda fase do trabalho pretendia-se implementar um sistema de telemanipulação constituído por um robô *master*, disponível comercialmente, e por um robô *slave* (robô cartesiano de 3 eixos existente).

Para o efeito foi necessário efectuar a interligação dos controladores dos dois subsistemas (*master* e *slave*). Posteriormente, foi desenvolvida uma estrutura de controlo para o sistema de telemanipulação. Desta forma, esta segunda fase do trabalho compreendeu as seguintes etapas:

- Implementação da ligação física;
- Concepção da interface gráfica de comando;
- Concepção da arquitectura de controlo;
- Implementação da arquitectura de controlo;
- Avaliação experimental.

1.4 Estrutura da Dissertação

Tendo em vista os objectivos enunciados na secção anterior, o trabalho desenvolvido é reportado neste documento, que se encontra organizado em sete capítulos e dois anexos.

No capítulo 2 são apresentadas as principais características do robô cartesiano (*slave*). É apresentado o modelo do sistema físico, descrita a síntese dos controladores de eixo e apresentados os algoritmos utilizados pelos geradores de trajectória. Neste capítulo são também apresentadas as principais funcionalidades de operação, em modo autónomo, do robô cartesiano e descrita a arquitectura de controlo implementada: é apresentada a interligação entre as várias componentes do sistema de controlo do robô, dando-se especial importância à descrição da aplicação de controlo desenvolvida.

No capítulo 3 descreve-se a interface gráfica de comando do robô cartesiano em modo autónomo. Esta possibilita que o utilizador efectue operações de definição e execução de trajectórias entre dois ou mais pontos, com a possibilidade de escolha de diferentes tipos de movimento. Sendo que, após a conclusão das trajectórias é possível a visualização de gráficos que possuem informação relativa aos movimentos efectuados. A interface gráfica permite também o movimento manual de cada um dos eixos do robô.

No capítulo 4 são descritos os principais requisitos de funcionamento de um sistema *master/slave*. É também apresentado e caracterizado o dispositivo *Phantom Haptic Device*, que desempenha as funções de *master* no sistema de telemanipulação. Por último é descrita a arquitectura de *hardware* do sistema robótico do tipo *master/slave*.

A arquitectura do *software* de controlo do sistema robótico de telemanipulação é apresentada no capítulo 5. São descritas as várias aplicações desenvolvidas, nomeadamente a aplicação gráfica de comando (*Visual Studio*) e a aplicação de controlo (*Simulink*).

No capítulo 6 são descritas as várias componentes que compõem a interface gráfica de comando do sistema de telemanipulação, bem como todas as suas funcionalidades.

No capítulo 7 apresentam-se as principais conclusões e possíveis desenvolvimentos futuros.

Finalmente, os anexos apresentam muito sucintamente os *manuais de operação* do sistema desenvolvido.

1.5 Conclusões

Neste capítulo foi efectuada uma introdução aos sistemas de telemanipulação, descrevendo-se a sua origem, o seu significado, as áreas de aplicação e as tecnologias utilizadas. Foram apontados alguns exemplos de trabalhos relacionados com a área dos sistemas robóticos de telemanipulação, mais concretamente exemplos de trabalhos similares aos desenvolvidos nesta tese.

Do estudo efectuado dos sistemas de telemanipulação verificou-se que existem muitas áreas de aplicação, tirando proveito das vantagens desses sistemas para a realização de tarefas repetitivas, perigosas ou volumosas, impossíveis de realizar por operadores humanos. Existem ainda tarefas que devido à elevada distância entre o operador e o sistema a controlar requerem a utilização de sistemas robóticos de telemanipulação.

Neste capítulo foi também resumidamente apresentado o sistema robótico do tipo *master/slave* utilizado para a realização deste trabalho. Deste modo, foram descritos sucintamente os robôs *slave* (cartesiano) e o *master* (*Phantom Haptic Device*). Por último foram enunciados os objectivos e descrita a organização desta tese.

Capítulo 2

Descrição e Estratégia de Controlo do Robô Cartesiano

2.1 Introdução

Neste capítulo será efectuada a descrição das principais características do robô cartesiano (*slave*). O seu espaço de trabalho, capacidade de carga, estrutura e transmissão mecânica, tipo de actuadores, sensores e detectores, bem como o *hardware* e o *software* de controlo utilizados.

É efectuada uma descrição da modelação do sistema físico e da síntese dos controladores de eixo, bem como o desenvolvimento dos algoritmos utilizados para implementar os geradores de trajectória. Neste contexto, são apresentados e devidamente comentados alguns resultados experimentais, que comprovam o bom desempenho dos algoritmos de controlo desenvolvidos.

Neste capítulo são também apresentadas as principais funcionalidades de operação, em modo autónomo, do robô cartesiano. De salientar que a possibilidade de funcionamento do robô cartesiano em modo autónomo, independentemente da utilização deste como *slave* no sistema de telemanipulação, tem como objectivo explorar as funcionalidades típicas de um robô industrial. Possibilitando desta forma o desenvolvimento, teste e

validação de vários tipos de controladores, tornando-se assim uma excelente plataforma de ensino no desenvolvimento de estratégias de controlo robustas.

Por último, é efectuada uma descrição da arquitectura de controlo implementada, onde é apresentada a interligação entre as várias componentes do sistema de controlo do robô cartesiano, dando-se especial importância à descrição da aplicação de controlo desenvolvida.

2.2 Robô Cartesiano

O robô cartesiano é composto por três eixos lineares de fusos de esferas da *Rexroth*, montados como mostra a figura 2.1.

Figura 2. 1 – Robô Cartesiano

Os modelos dos fusos de esferas de cada eixo são respectivamente:

- Eixo X: *Rexroth* MKK 25-110 com 1400 mm de comprimento;
- Eixo Y: *Rexroth* MKK 25-210 com 1200 mm de comprimento;
- Eixo Z: *Rexroth* MKK 25-210 com 800 mm de comprimento.

O movimento no eixo X é realizado através de dois eixos paralelos, sendo apenas um deles um eixo motor, enquanto que o outro serve apenas para guiamento. Estes dois eixos estão fixos na estrutura e os seus patins suportam o eixo Y que por sua vez movimenta o eixo Z através de uma união entre os patins dos eixos Y e Z.

Esta estrutura proporciona uma movimentação no espaço tridimensional com um volume de trabalho de 1080x1250x720 mm e um atravancamento de 1750x1550x2900 mm. No entanto, por uma questão de segurança, o espaço de trabalho definido para este projecto foi de 630x520x300 mm. Sendo que em cada extremidade dos três eixos estão instalados batentes metálicos de protecção, para evitar que a movimentação dos eixos ultrapasse os limites de segurança.

Os passos associados a cada fuso de esferas são de 20 mm para os eixos X e Y e 10 mm para o eixo Z. Quanto às velocidades nominais, sendo os motores de 3000 rpm, são de 1 m/s em X e Y e 0.5 m/s em Z. Por questões de segurança optou-se por limitar as velocidades máximas para cada eixo nos 0.2 m/s, uma vez que nos testes experimentais efectuados verificou-se que este valor de velocidade correspondia a movimentos com rapidez considerável.

Os elementos actuadores dos eixos lineares de fuso de esferas são servomotores *brushless AC* da *Omron* (*R88MW75030*) com 750 W de potência e 3000 rpm de velocidade nominal. Os motores são idênticos para os três eixos tendo o motor do eixo vertical um travão electromagnético. A estes motores estão associados *drivers* da *Omron* (*R88DWT08HH*) alimentados a 230V AC, podendo ser comandados em velocidade ou binário através de uma entrada analógica (-10/+10V DC). Na realização deste trabalho optou-se pelo comando em binário. Cada motor tem um *encoder* (*Omron*) incremental *standard* para *feedback* de posição, que se liga com o *driver* através de uma comunicação série definida pelo fabricante. Este equipamento gera 2048 impulsos por rotação e funciona a uma velocidade máxima de 5000 rpm.

O comando do robô cartesiano é efectuado por computador através de cartas de interface A/D, D/A, I/O digital e *Encoders* (*PCI-QUAD04*), da *MeasurementComputing*. É através da carta D/A (*PCIM-DDA 06/16*) que são enviadas referências de tensão aos *drivers* que as converte em referências de binário. Através do I/O digital desta carta é também possível monitorizar e controlar os sinais digitais presentes no armário de controlo do robô. A secção de I/O digital da carta A/D (*PCI-DAS 1602/16*) é responsável pela aquisição de alguns dos sinais presentes no armário de controlo (estado dos *drivers* – figura 2.2), sendo eles:

- *Driver On – Drivers* de potência alimentados;
- *Emergency* – Sinal de emergência activo (botoneiras);
- *Alarm* – Alarme genérico do sistema;
- *Servo Ready – Drivers* de potência preparados para funcionamento;
- *Run – Drivers* de potência activos;
- *Overtravel* – Sinalização de actuação de um dos fins-de-curso mecânicos;
- *Brake* – Sinal de travão do eixo Z actuado.

O I/O digital da carta A/D é também responsável pela aquisição dos sinais dos fins-de-curso mecânicos (2 por cada eixo, colocados nas extremidades dos mesmos) e dos detectores indutivos (1 por cada eixo, colocados aproximadamente a meio do espaço de trabalho definido). As entradas analógicas da carta são usadas para adquirir os sinais dos acelerómetros montados nos eixos do robô.

Os fins-de-curso mecânicos são de posição ajustável e foram colocados perto das extremidades de cada eixo. Assim quando estes são actuados enviam um sinal ao *driver* de potência do respectivo eixo, sucedendo-se a paragem do mesmo. Desta forma os fins-de-curso mecânicos têm funções de segurança, ou seja impedem que durante as movimentações dos eixos do robô, estes ultrapassem o espaço de trabalho definido e colidam com os batentes metálicos de protecção.

Os detectores indutivos estão localizados aproximadamente a meio do curso de cada eixo e são utilizados para definir a origem do referencial da estrutura cartesiana (efectuar o zero dos *encoders* incrementais).

Figura 2. 2 – Armário Eléctrico de Controlo

Para o desempenho das funções de controlador do robô cartesiano optou-se pela escolha de um *PC Desktop*, mas podia utilizar-se um *Laptop* ou um *PC* industrial (*PC-104*), entre outros. Neste *PC* estão instaladas as cartas de aquisição de dados, que possuem uma interface para barramento *PCI*. Estas cartas de aquisição são responsáveis pela interface entre o computador (*Target PC*) e o sistema físico a controlar (robô cartesiano), pois é através destas que é possível a monitorização e o controlo de variáveis analógicas e sinais digitais.

Assim, a aplicação de controlo do sistema corre num *PC Pentium IV* @ 1.71 *GHz* (*Target PC*), enquanto que a interface de comando do operador está instalada num *PC Pentium IV* @ 2.8 *GHz* (*Host PC*). No *Target PC* foi instalado o módulo de tempo-real *xPC Target* do *Matlab/Simulink*, este módulo corresponde a um *kernel* de tempo-real. Desta forma os recursos do *Target PC* são unicamente utilizados para a realização das tarefas do controlador, sendo que essas tarefas são executadas com um tempo de ciclo de 1 ms (1 kHz). No *Host PC* instalou-se o sistema operativo *Windows XP Professional*, o *software* de desenvolvimento do controlador do sistema *Matlab/Simulink* e o *software* de desenvolvimento de ambientes gráficos *Microsoft Visual Studio*. Os dois computadores comunicam através de um canal de comunicação dedicado.

Depois de desenvolvida e simulada no *Host PC*, a aplicação de controlo é enviada para o *Target PC* para ser executada em tempo-real. No *Host PC* o utilizador possui uma interface que pode utilizar para enviar comandos e observar o estado do robô.

2.3 Modelação do Sistema Físico e Síntese de Controladores

Nesta secção será apresentado o modelo do sistema físico, com a obtenção da respectiva função de transferência. Será ainda descrito o processo de síntese dos controladores de velocidade e de posição dos eixos do robô cartesiano. Por último serão apresentados alguns resultados experimentais que demonstram o desempenho dos controladores implementados.

Cada eixo do robô possui um sistema de transmissão mecânica fuso/fêmea de esferas com uma relação de transmissão elevada. Em geral, a utilização de transmissões mecânicas nos eixos dos robôs, leva a que as variações inércias (causadas por alterações da configuração da estrutura ou da carga manipulada), quando referidas aos motores, surjam divididas pelo quadrado do factor de redução. O efeito dessas variações, para factores de redução elevados, é, assim, desprezável, abrindo caminho à utilização de controladores de ganhos fixos, lineares e descentralizados (ex. PID's). Cada eixo pode possuir o seu próprio subsistema de controlo, onde eventuais acoplamentos dinâmicos são tratados como perturbações.

2.3.1 Modelação do Sistema Físico

O modelo físico simplificado de cada eixo, corresponde a um "Sistema de Parafuso" (figura 2.3), uma vez que em sistemas de controlo de movimento é frequente a necessidade de converter movimento rotacional em movimento de translação (Ogata, 1997). As equações que descrevem a dinâmica deste sistema são as seguintes:

$$\begin{cases} \dfrac{\theta}{2\pi} = \dfrac{x}{h} & 2.1 \\ Fx = T\theta & 2.2 \\ F = M\,\ddot{x} + B\dot{x} & 2.3 \end{cases}$$

Símbolo	Legenda (unidades S.I.)
θ	Posição angular (rad/s)
x	Posição linear (m)
h	Passo (m)
F	Força (N)
T	Binário (N.m)
M	Massa (kg)
B	Atrito dinâmico (N/m)

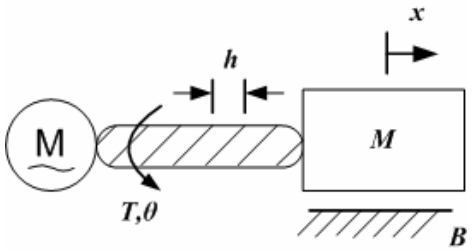


Figura 2. 3 – Sistema de Parafuso

Resultando:

$$T = M\left(\frac{h}{2\pi}\right)^2 \ddot{\theta} + B\left(\frac{h}{2\pi}\right)^2 \dot{\theta} \Leftrightarrow T = J\ddot{\theta} + D\dot{\theta} \qquad 2.4$$

Símbolo	Legenda (unidades S.I.)
J	Momento de inércia ($kg.m^2$)
D	Atrito dinâmico (rotação) (N.m.s/rad)

Partindo deste exemplo, passa-se então à modelação de um dos eixos lineares, que é caracterizado por um sistema de transmissão de fuso de esferas, com um acoplamento composto por uma componente de elasticidade, tal como sugere a figura 2.4:

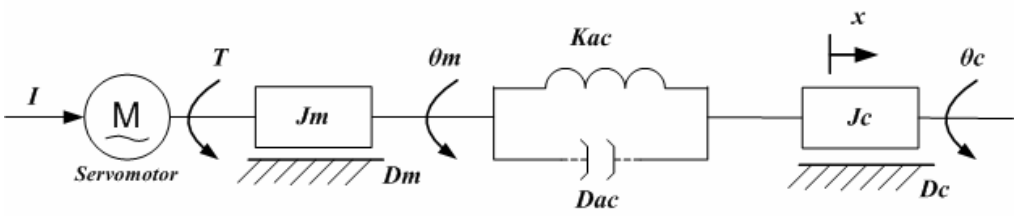


Figura 2. 4 – Modelo de um Sistema de Transmissão com Elasticidade

Assim as equações que traduzem a dinâmica do sistema são:

$$\begin{cases} T = Kt\, I & 2.5 \\ T = Jm\,\ddot{\theta}m + Dm\,\dot{\theta}m + Dac\left(\dot{\theta}m - \dot{\theta}c\right) + Kac\left(\theta m - \theta c\right) & 2.6 \\ Jc\,\ddot{\theta}c + Dc\,\dot{\theta}c + Dac\left(\dot{\theta}c - \dot{\theta}m\right) + Kac\left(\theta c - \theta m\right) = 0 & 2.7 \end{cases}$$

Símbolo	Legenda (unidades S.I.)
Kt	Constante eléctrica do motor (N.m/A)
I	Corrente eléctrica (A)
θm	Posição angular do motor (rad/s)
θc	Posição angular da carga (rad/s)
Jm	Inércia do motor ($kg.m^2$)
Jc	Inércia da carga ($kg.m^2$)
Dm	Atrito dinâmico no motor (N.m.s/rad)
Dc	Atrito dinâmico na carga (N.m.s/rad)
Dac	Atrito dinâmico no acoplamento (N.m.s/rad)
Kac	Rigidez do acoplamento (N.m/rad)

A partir das equações (2.6) e (2.7) consegue chegar-se à função de transferência do sistema, aplicando a Transformada de *Laplace* da seguinte forma:

$$\begin{cases} \left[Jm\,s^2 + (Dm + Dac)s + Kac\right]\theta m(s) - (Dac\,s + Kac)\theta c(s) = T(s) & 2.8 \\ \left[Jc\,s^2 + (Dc + Dac)s + Kac\right]\theta c(s) - (Dac\,s + Kac)\theta m(s) = 0 & 2.9 \end{cases}$$

A partir da equação (2.9), obtém-se:

$$\frac{\theta c(s)}{\theta m(s)} = \frac{Dac\,s + Kac}{Jc\,s^2 + (Dc + Dac)s + Kac} \qquad 2.10$$

Da manipulação das equações (2.8) e (2.9), resulta:

$$\begin{aligned} \frac{\theta m(s)}{T(s)} = \frac{Jc\,s^2 + (Dc + Dac)s + Kac}{s^4(Jc\,Jm) + s^3\left[Jc\,Dm + Jm\,Dc + (Jm + Jc)Dac\right] + \\ + s^2\left[Dm\,Dc + Dac(Dm + Dc) + Kac(Jm + Jc)\right] + sKac(Dm + Dc)} \end{aligned} \qquad 2.11$$

A função de transferência final, $\frac{\theta c(s)}{T(s)} = \frac{\theta c(s)}{\theta m(s)}\frac{\theta m(s)}{T(s)}$, que se pode obter a partir das equações (2.10) e (2.11), resulta em:

$$\begin{aligned} \frac{\theta c(s)}{T(s)} = \frac{Dac\,s + Kac}{s^4(Jc\,Jm) + s^3\left[Jc\,Dm + Jm\,Dc + (Jm + Jc)Dac\right] + \\ + s^2\left[Dm\,Dc + Dac(Dm + Dc) + Kac(Jm + Jc)\right] + sKac(Dm + Dc)} \end{aligned} \qquad 2.12$$

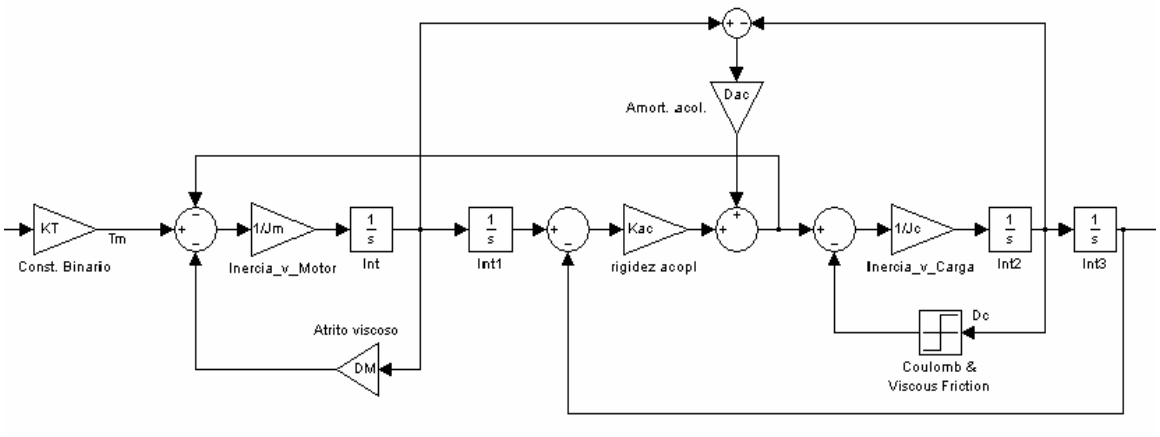


Figura 2. 5 – Modelo em *Simulink* de um eixo linear

Da equação (2.12) pode concluir-se que, no caso de um acoplamento rígido (ideal) em que $(Kac \to \infty)$ e $(Dac = 0)$, resulta:

$$\frac{\theta m(s)}{T(s)} = \frac{\theta c(s)}{T(s)} = \frac{1}{s^2(Jc + Jm) + s(Dc + Dm)} \qquad 2.13$$

A figura 2.5 mostra o modelo *Simulink* que implementa as equações referentes à dinâmica dos eixos do robô cartesiano. De seguida, são apresentados os valores dos parâmetros que constituem o modelo:

Parâmetro	**Valor (unidades S.I.)**
h	eixo X e Y – 20 mm; eixo Z – 10 mm (fornecido pelo fabricante)
Kt	0.590 N.m/A (fornecido pelo fabricante)
Jm	1.672E-4 $kg.m^2$ (calculado a partir de valores fornecidos pelo fabricante)
Jc	eixo X – 0.0024 $kg.m^2$; eixo Y – 0.0016 $kg.m^2$; eixo Z – 8.695E-4 $kg.m^2$ (calculado a partir de valores fornecidos pelo fabricante)
Dm	0.001 N.m.s/rad (fornecido pelo fabricante)
Dc	0.001 N.m.s/rad (10% do binário do motor à velocidade máxima)
Dac	4E-4 N.m.s/rad (valor estimado)
Kac	12E3 N.m/rad (fornecido pelo fabricante)

2.3.2 Síntese de Controladores

Nesta secção será abordado o desenvolvimento, a implementação e a validação dos controladores usados para o controlo de velocidade e de posição dos eixos. Serão apresentados os resultados obtidos em simulação e nos testes experimentais efectuados, sendo apresentada a análise desses resultados.

Controlador de Velocidade

A utilização de codificadores incrementais como transdutores de deslocamento, obriga a que, sempre que é iniciada uma sessão, o primeiro procedimento a efectuar pelo robô cartesiano corresponda ao posicionamento dos eixos em pontos referenciados e à inicialização dos codificadores (zero do robô cartesiano).

Para efectuar o procedimento de zero foi projectado um controlador de velocidade PI (Proporcional + Integral). Para controlo de velocidade, o sistema a controlar pode ser visto como um sistema do tipo zero. Pelo que, a acção integral garantirá erro nulo para uma entrada em velocidade. Em simulação este controlador revelou um excelente desempenho.

Controlador PI (Proporcional + Integral)

A equação 2.14 traduz a dinâmica de um controlador PI, existindo algumas variantes a esta equação geral.

$$u(t) = Kp\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau \right) \qquad 2.14$$

Símbolo	Legenda (unidades S.I.)
u	Acção de controlo
Kp	Ganho proporcional
e	Erro
Ti	Constante de tempo integral

Para simplificação do cálculo dos ganhos do controlador é utilizado o modelo simplificado do sistema, ou seja, considerando o acoplamento rígido, equação (2.13), e desprezando a modelação do atrito estático da carga.

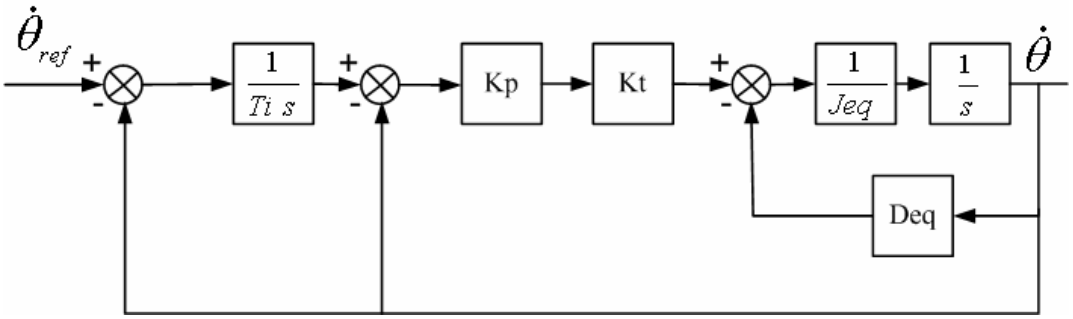


Figura 2. 6 – Diagrama de Blocos do Controlador PI de Velocidade

A partir do diagrama de blocos do sistema da figura 2.6 resulta a seguinte função de transferência em malha fechada:

$$\frac{\dot{\theta}}{\dot{\theta}_{ref}} = \frac{\frac{1}{Ti\ s}\frac{Kp\ Kt}{Jeq\ s + Deq + Kp\ Kt}}{1 + \frac{1}{Ti\ s}\frac{Kp\ Kt}{Jeq\ s + Deq + Kp\ Kt}} \qquad 2.15$$

em que $Jeq = Jm + Jc$ e $Deq = Dm + Dc$.

Símbolo	**Legenda (unidades S.I.)**
Jeq	Momento de inércia equivalente ($kg.m^2$)
Deq	Atrito dinâmico equivalente (N.m.s/rad)

Sabendo que a função de transferência de um sistema de segunda ordem é:

$$\frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \qquad 2.16$$

Símbolo	Legenda (unidades)
ω_n	Frequência natural
ξ	Coeficiente de amortecimento

Manipulando a equação (2.15), de forma a obter uma equação na forma da função acima apresentada (2.16), resulta:

$$\frac{\dot{\theta}}{\dot{\theta}_{ref}} = \frac{\frac{Kp\ Kt}{Jeq\ Ti}}{s^2 + s\frac{(Deq + Kp\ Kt)}{Jeq} + \frac{Kp\ Kt}{Jeq\ Ti}} \qquad 2.17$$

Da conjugação das equações (2.16) e (2.17), obtêm-se:

$$\omega_n^2 = \frac{Kp\ Kt}{Jeq\ Ti} \qquad 2.18$$

e

$$2\xi\omega_n = \frac{Deq + Kp\ Kt}{Jeq} \qquad 2.19$$

Uma vez que os valores de *Kt*, *Deq* e *Jeq* são conhecidos, escolhendo $\omega_n = 80 rad/s$ e $\xi = 1$, das equações (2.18) e (2.19) retiram-se os valores dos parâmetros do controlador, neste caso:

Kpx=0.694 e *Tix=0.025*;
Kpy=0.490 e *Tiy=0.025*;
Kpz=0.279 e *Tiz=0.025*.

Uma vez obtidos os parâmetros do controlador PI, passa-se então à fase de validação (simulação em *Simulink*) dos mesmos. Para isso utiliza-se o modelo completo do sistema (que inclui o acoplamento), aplicando um controlador com a configuração que é apresentada na figura 2.7:

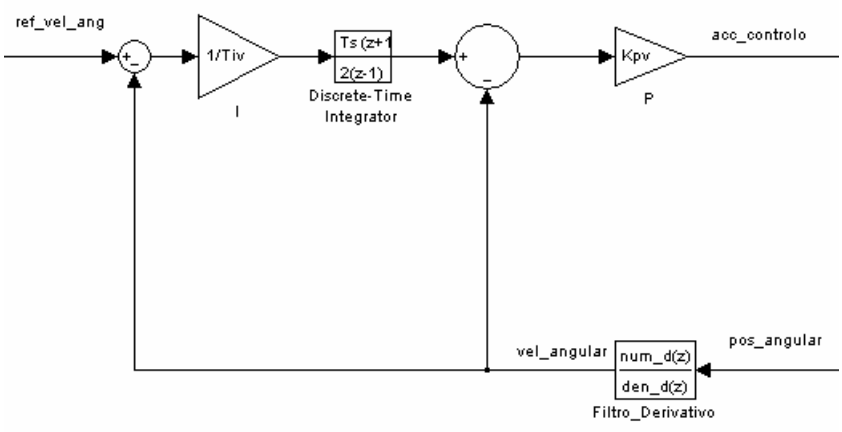


Figura 2. 7 – Modelo em *Simulink* do Controlador PI de Velocidade

Como se pode verificar na figura 2.7, trata-se de uma implementação digital do controlador PI projectado, utilizando um integrador discretizado pelo método de *Tustin* (*Discrete-Time Integrator*), com o período de amostragem de *Ts* =1ms.

Uma vez que não se dispõe directamente do sinal de velocidade, aplica-se ao sinal de posição um filtro derivativo de primeira ordem (discretizado), com frequência de corte de 100 Hz, cuja função de transferência é:

$$H(s) = \frac{As}{s + A} \qquad 2.20$$

com $A = 2\pi \times 100$.

Aplicando uma referência de velocidade que evolui no tempo de 0 a 50 mm/s segundo um perfil sinusoidal, obtém-se um arranque suave do eixo, tal como se pode verificar nos resultados obtidos por simulação.

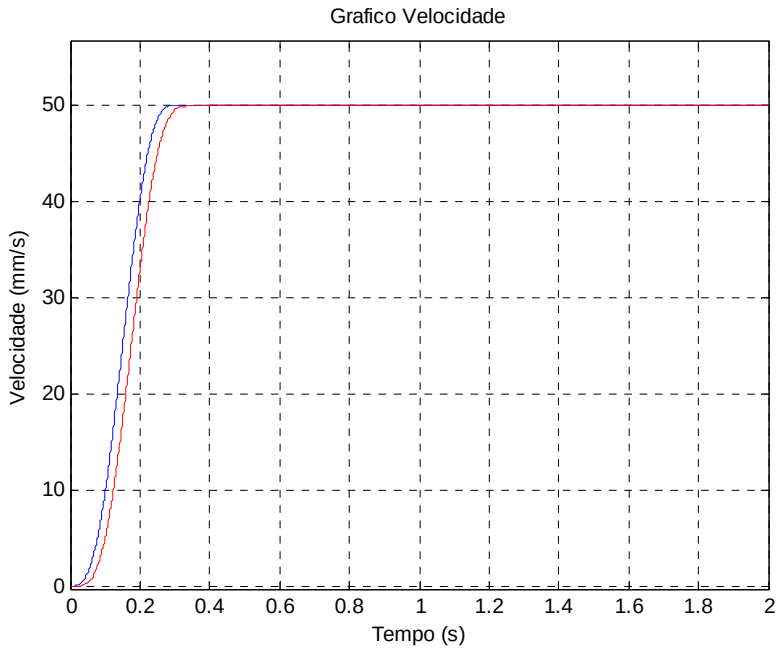


Figura 2. 8 – Resposta do Sistema a um Perfil Sinusoidal de Velocidade sem Atrito Estático; azul: Referência de Velocidade; vermelho: Reposta do Sistema

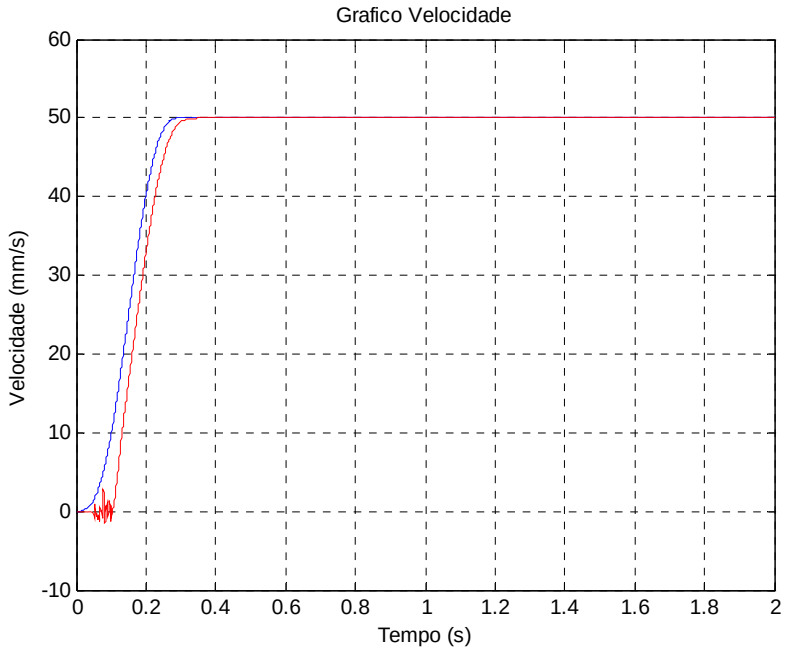


Figura 2. 9 – Resposta do Sistema a um Perfil Sinusoidal de Velocidade com Atrito Estático; azul: Referência de Velocidade; vermelho: Reposta do Sistema

Da análise das figuras 2.8 e 2.9 pode verificar-se que os resultados de simulação são bastante satisfatórios. De notar que no caso da figura 2.9 foi considerado o efeito do atrito estático. Por essa razão verifica-se uma vibração inicial na carga (*stick-slip*), o que

não acontece no resultado da figura 2.8, dado que essa "*não-linearidade*" não foi tida em conta.

Por razões práticas não é funcional a implementação do controlador PI de velocidade. Este controlador necessita de informação de velocidade e do seu integral (posição), sendo que a informação de posição só estará disponível após o procedimento de inicialização do robô. Assim sendo, foi projectado um controlador proporcional, que apesar de apresentar desempenho inferior ao controlador PI, satisfaz os requisitos que se impõem para este tipo de procedimento, ou seja, efectuar um movimento suave a uma velocidade razoável (aproximadamente constante), não sendo necessário garantir erro estático nulo.

Controlador Proporcional

Como já foi referido não foi possível a utilização do controlador PI descrito anteriormente. Assim, ao invés foi utilizado um controlador proporcional de velocidade, que como se irá verificar nesta secção apresenta resultados inferiores ao controlador anterior, mas que permite executar correctamente o procedimento de inicialização do robô (notar que o controlador de velocidade só é executado no procedimento de inicialização).

Em seguida é exposta a forma como este controlador foi projectado, apresentando os passos inerentes à síntese do mesmo, bem como os resultados obtidos tanto em simulação como nos testes experimentais efectuados.

Assim, este controlador pode ser representado pela seguinte expressão:

$$u(t) = Kp\, e(t) \qquad 2.21$$

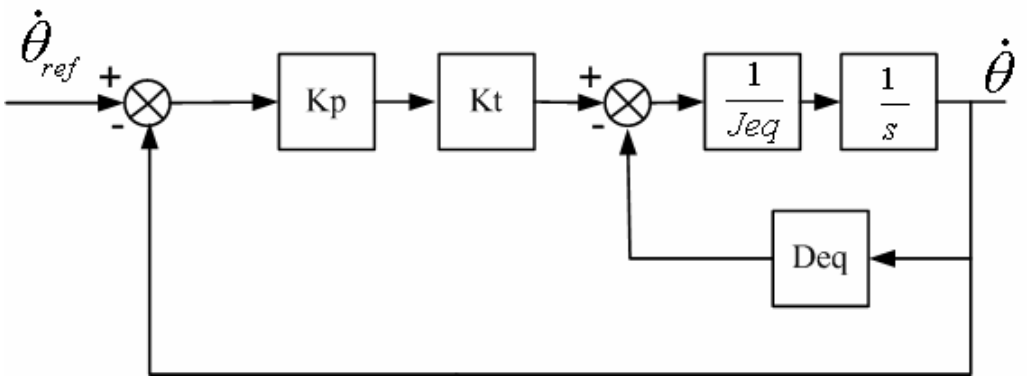


Figura 2. 10 – Diagrama de Blocos do Controlador Proporcional de Velocidade

A partir do diagrama de blocos da figura 2.10 chega-se à função de transferência do sistema em malha fechada:

$$\frac{\dot{\theta}}{\dot{\theta}_{ref}} = \frac{Kp\ Kt}{Jeq\ s + Kp\ Kt + Deq} \qquad 2.22$$

Com o ganho proporcional obtido (*Kpx=0.694; Kpy=1; Kpz=0.4;* ajustados experimentalmente), implementa-se um controlador proporcional em *Simulink*, tal como sugere a figura 2.11:

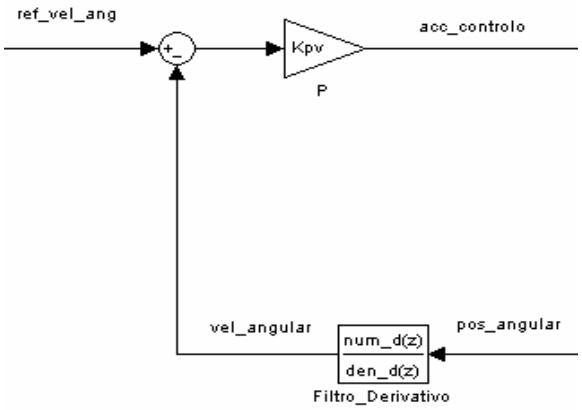


Figura 2. 11 – Modelo em *Simulink* do Controlador Proporcional de Velocidade

Em seguida são apresentados os resultados obtidos em simulação, relativos ao desempenho do controlador proporcional, tendo como base o modelo completo da figura 2.5, tal como aconteceu com o controlador PI.

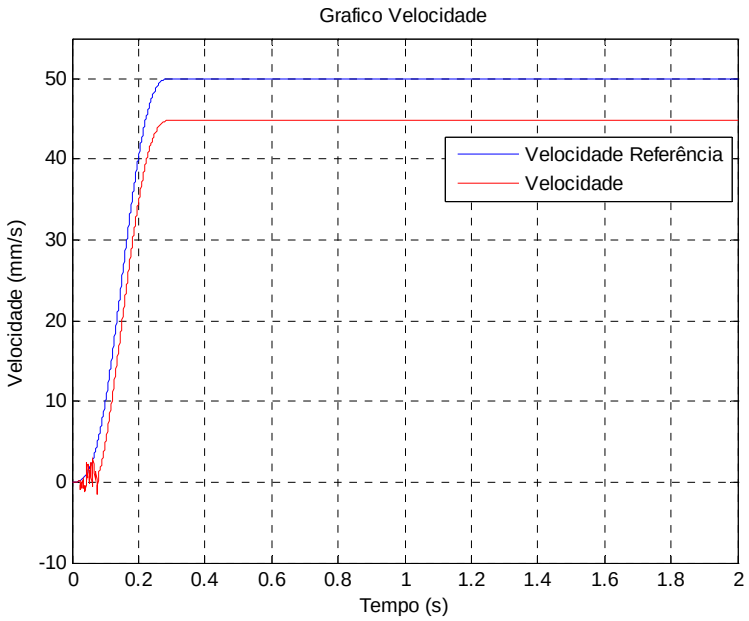


Figura 2. 12 – Resposta do Sistema a um Perfil Sinusoidal de Velocidade com Atrito Estático

Pode concluir-se que apesar deste controlador não apresentar os resultados obtidos pelo controlador PI, verifica-se que a menos de um erro em regime permanente a velocidade se mantém constante (figura 2.12), sendo este o principal objectivo da implementação deste controlador.

Controlador de Posição (PID)

Este controlador foi projectado de modo a que o robô pudesse fazer seguimento de trajectória entre dois ou mais pontos. As trajectórias são definidas especificando as coordenadas dos pontos no espaço cartesiano e a velocidade média desejada. Assim, foram desenvolvidos para cada um dos eixos controladores PID independentes.

Pretendia-se com a implementação de controladores PID a obtenção de bons resultados no que diz respeito à capacidade de posicionamento. Isso foi conseguido, como se pode constatar pelos resultados apresentados nesta secção, quer na fase de simulação quer na fase de testes experimentais efectuados.

O controlador PID resulta da combinação dos modos proporcional, integral e derivativo (figura 2.13). Pode afirmar-se que é um compromisso entre as vantagens e desvantagens de um controlador PI e PD. A acção integral está directamente ligada à exactidão do

sistema, sendo responsável pelo erro nulo em regime permanente. O efeito perturbante do controlador PI é contrabalançado pela acção derivativa que tende a aumentar a estabilidade relativa do sistema ao mesmo tempo que torna a resposta mais rápida devido ao seu efeito de antecipação.

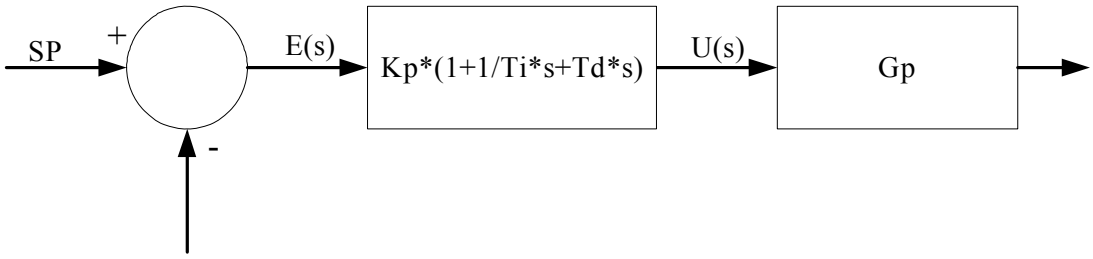


Figura 2. 13 – Diagrama de Blocos de um Controlador Proporcional + Integral + Derivativo (PID)

A saída do controlador PID é dada por:

$$u(t) = Kp\left(e(t) + \frac{1}{Ti}\int_0^{\tau} e(\tau)d\tau + Td\,\frac{de(t)}{dt} \right) \xrightarrow{L} U(s) = Kp\left(1 + \frac{1}{Ti\,s} + Td\,s \right) E(s) \quad 2.23$$

Símbolo	**Legenda (unidades S.I.)**
U	Acção de controlo
E	Erro
Td	Constante de tempo derivativa

Neste tipo de controlador, o modo integral é usado para eliminar o erro em regime estacionário causado por grandes variações de carga. O modo derivativo, com o seu efeito estabilizador, permite um aumento do ganho e reduz a tendência para as oscilações, o que conduz a uma velocidade de resposta superior quando comparado com um controlador P ou PI. No entanto, estas propriedades assumem um carácter geral, pelo que podem existir excepções em determinados sistemas.

A função de transferência do controlador PID é dada por:

$$G_{pid} = \frac{u(s)}{r(s)} = K\left(1 + \frac{1}{T_i\, s} + \frac{s\, p\, T_d}{s + p}\right) = \frac{K\left(s^2 + \frac{1 + T_d T_i}{T_i} s + \frac{p + T_i\, p}{T_i}\right)}{s(s + p)} \qquad 2.24$$

Na equação (2.24) a componente derivativa é filtrada por um filtro passa-baixo para minimização do ruído. É importante referir que as equações anteriormente apresentadas constituem a versão clássica do controlador PID. Existem outras versões e variantes, mas a filosofia de funcionamento, a partir da combinação dos efeitos das três acções básicas, é a mesma.

Assim, o controlador PID implementado é uma dessas variantes, tal como sugere a configuração da figura 2.14, equação (2.24) discretizada:

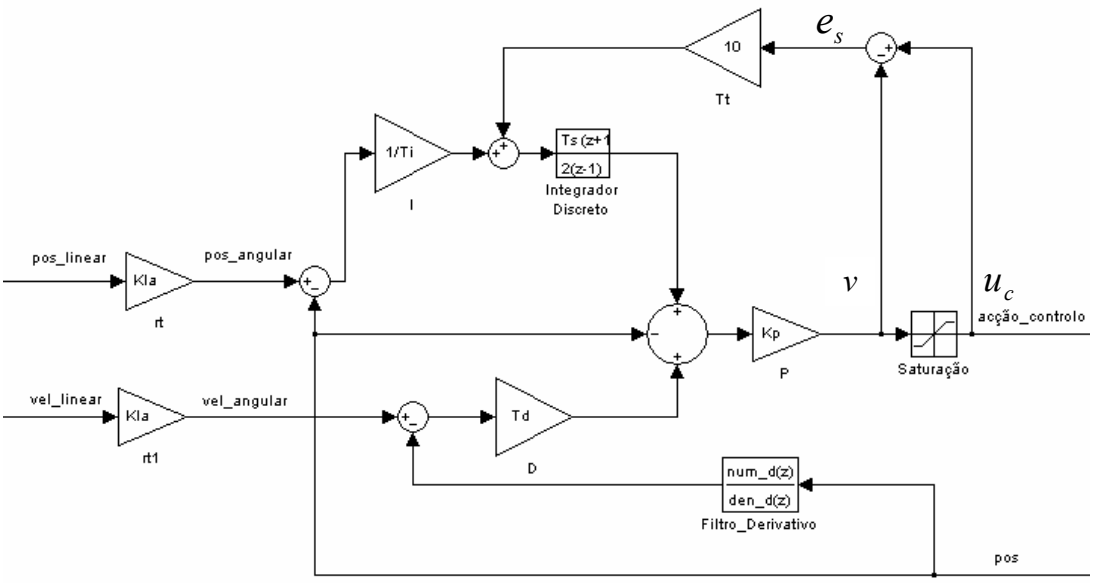


Figura 2. 14 – Modelo em *Simulink* do Controlador PID de Posição

Como se pode verificar na figura 2.14 o controlador PID possui uma malha de anti-saturação (*Anti-Windup*), para evitar os problemas que o *Windup* do integrador apresenta. O *Windup* do integrador é um efeito que é causado pelo aumento do valor do erro. Se o valor do erro aumentar ao ponto de o integrador saturar a acção de controlo, a malha de realimentação fica inoperante, uma vez que a acção de controlo permanecerá saturada mesmo que o erro diminua. Deste modo, a acção do integrador torna o sistema

instável, uma vez que pode efectuar a integração de valores muito elevados. Quando o valor do erro começa a reduzir, o valor do integrador pode ser tão alto ao ponto de demorar um tempo considerável até atingir um valor normal.

Para compensar este efeito podem ser utilizados vários métodos *Anti-Windup*. Um deles corresponde a impedir que o integrador efectue a integração quando o actuador está saturado. Outro método, que foi o escolhido, corresponde a efectuar uma realimentação do sinal de erro, e_s, (figura 2.14) correspondente à diferença entre o sinal de saída para o actuador u_c e a saída do controlador, *v*, enviado esse sinal de erro para o integrador atenuado por um ganho $1/T_t$ (T_t - constante tempo que, por regra $T_t = T_i$, mas em certos casos $T_t < T_i$ melhora as performances do controlador (Rundqwist, 1991)). Assim, o valor do erro e_s é zero enquanto o actuador não estiver saturado. Quando o actuador está saturado a realimentação tenta fazer com que o erro e_s seja zero, isto significa efectuar o *"reset"* do integrador, para que a saída do controlador se mantenha no limite da saturação (Astrom e Wittenmark, 1997).

Para a validação deste controlador foi utilizado o modelo completo da figura 2.5. Deste modo, a partir do diagrama da figura 2.15 chegou-se à função de transferência do sistema.

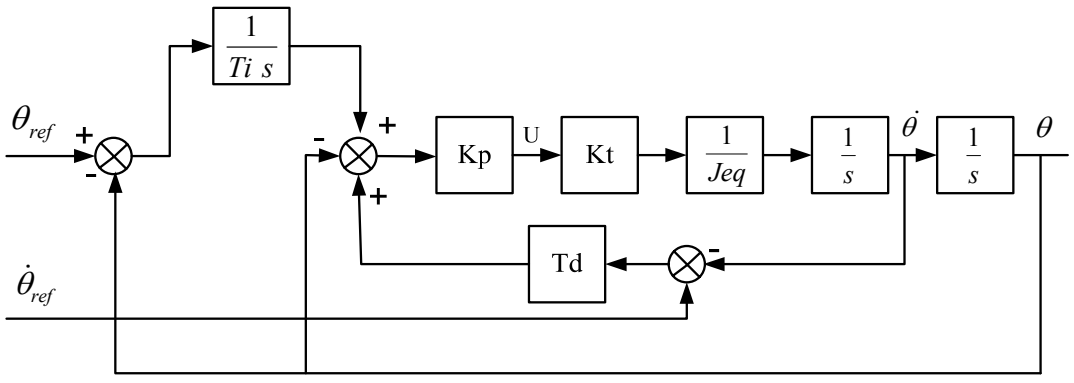


Figura 2. 15 - Diagrama de Blocos do Controlador PID de Posição e do Sistema Físico

$$U = \left[-\theta + (\theta_{ref} - \theta) Td\, s\, \theta + (\theta_{ref} - \theta) \frac{1}{Ti\, s} \right] Kp \qquad 2.25$$

$$\theta = \left[-\theta + (\theta_{ref} - \theta) Td\ s\ \theta + \frac{(\theta_{ref} - \theta)}{Ti\ s} \right] \frac{Kp\ Kt}{Jeq\ s^2} \tag{2.26}$$

$$\theta\ Jeq\ s^2 = -\theta\ Kp\ Kt + (\theta_{ref} - \theta) \frac{Kp\ Kt}{Ti\ s} + (\theta_{ref} - \theta) Td\ Kp\ Kt\ s \tag{2.27}$$

$$\theta\ Jeq\ s^2\ Ti\ s = -\theta\ Kp\ Kt\ Ti\ s + (\theta_{ref} - \theta) Kp\ Kt + (\theta_{ref} - \theta) s\ Td\ Kp\ Kt\ Ti\ s \tag{2.28}$$

$$\frac{\theta}{\theta_{ref}} = \frac{s^2\ Ti\ Td\ Kp\ Kt + Kp\ Kt}{s^3 Jeq\ Ti + s^2\ Ti\ Td\ Kp\ Kt + s\ Ti\ Kp\ Kt + Kp\ Kt} \tag{2.29}$$

Para efeitos de cálculo da função de transferência foram desprezados tanto os atritos do sistema como o acoplamento, de forma a tornar mais simples a obtenção da mesma.

Depois de obtida a função de transferência, equação (2.29), utilizando o *Protótipo de Bessel* (Ogata, 1997) chegou-se aos valores dos parâmetros do controlador da seguinte forma:

$$B_K(s) = (s + 5.0093)\left[(s + 3.9668)^2 + 3.7845^2\right] \tag{2.30}$$

$$\phi_d(s) = \left(s + \frac{5.0093}{ts} \right) \left[\left(s + \frac{3.9668}{ts} \right)^2 + \left(\frac{3.7845}{ts} \right)^2 \right] \tag{2.31}$$

$$\phi_d(s) = s^3 + \frac{12.9429\ s^2}{ts} + \frac{69.7997\ s}{ts^2} + \frac{150.5693}{ts^3} \tag{2.32}$$

$$\frac{\theta}{\theta_{ref}} = \frac{s^2\ Ti\ Td\ Kp\ Kt + Kp\ Kt}{s^3 + \frac{s^2\ Td\ Kp\ Kt}{Jeq} + \frac{s\ Kp\ Kt}{Jeq} + \frac{Kp\ Kt}{Jeq\ Ti}} \tag{2.33}$$

$$\begin{cases} \dfrac{Td\ Kp\ Kt}{Jeq} = \dfrac{12.9429}{ts} \\ \dfrac{Kp\ Kt}{Jeq} = \dfrac{69.7997}{ts^2} \\ \dfrac{Kp\ Kt}{Jeq\ Ti} = \dfrac{150.5693}{ts^3} \end{cases}$$

De seguida são apresentados os ganhos de cada controlador de eixo obtidos a partir do sistema de equações anterior:

Ganho	**Valor**
Kpx	30.387
Tix	0.046
Tdx	0.018
Kpy	21.480
Tiy	0.046
Tdy	0.018
Kpz	12.264
Tiz	0.046
Tdz	0.018

Com os valores dos ganhos do controlador obtidos a partir do sistema de equações anterior e através do modelo em *Simulink*, simulou-se a reposta do sistema a um degrau utilizando o modelo da figura 2.5, o qual já contempla os efeitos dos atritos e do acoplamento que tinham sido desprezados no cálculo dos parâmetros do controlador, obtendo o seguinte desempenho na resposta ao degrau:

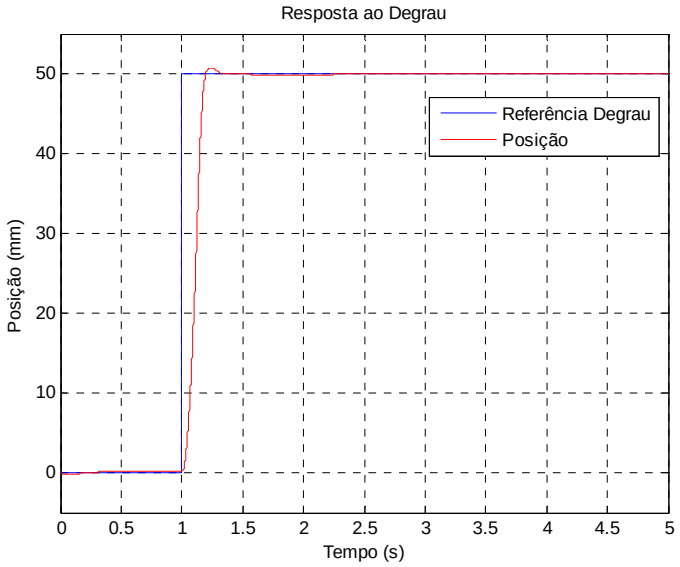


Figura 2. 16 – Resposta do Sistema ao Degrau

Da análise do gráfico da figura anterior pode verificar-se que o sistema apresenta um ligeiro *overshoot* (na ordem dos 0.84%), apresentando também um erro em regime permanente (na ordem dos 0.5E-3 mm). Perante os resultados obtidos, conclui-se que o controlador implementado satisfaz em grande parte as necessidades do sistema, nomeadamente a exactidão de posicionamento.

Depois de analisada a resposta do sistema a um degrau, testou-se a sua resposta para uma trajectória linear com um perfil sinusoidal de velocidade, uma vez que um dos objectivos finais consistia em implementar um algoritmo de posicionamento entre dois pontos, utilizando para o efeito um gerador de trajectória com este tipo de perfil. Este tema, *Geração de Trajectórias*, será abordado com mais pormenor na secção seguinte.

Desta forma, para uma trajectória linear com um perfil sinusoidal de velocidade, obteve-se a seguinte resposta (resultado de simulação):

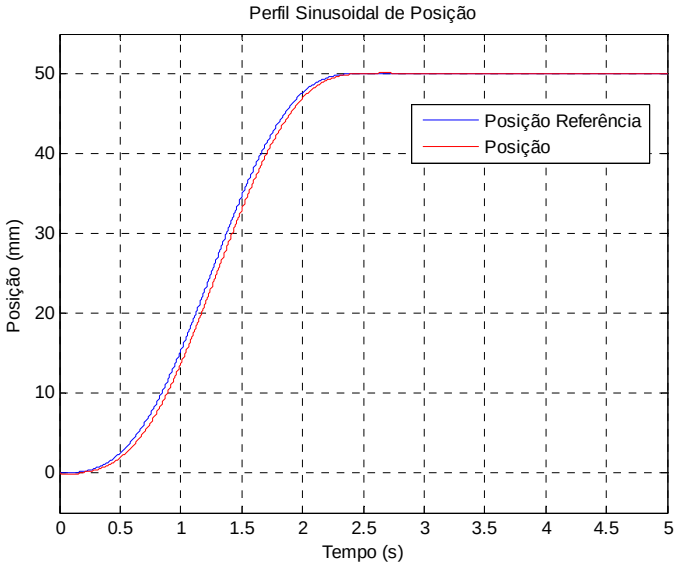


Figura 2. 17 – Resposta do Sistema a um Perfil Sinusoidal com Referência de Velocidade

Na figura 2.17 é possível analisar os bons resultados da resposta do controlador a trajectórias com perfil sinusoidal de velocidade, em que são utilizadas as referências de posição e velocidade, tal como sugere o modelo do sistema da figura 2.15.

Quando a referência de velocidade do gerador de trajectórias não é utilizada, verifica-se a existência de uma ligeira perturbação em regime permanente, tal como se pode visualizar na figura 2.18.

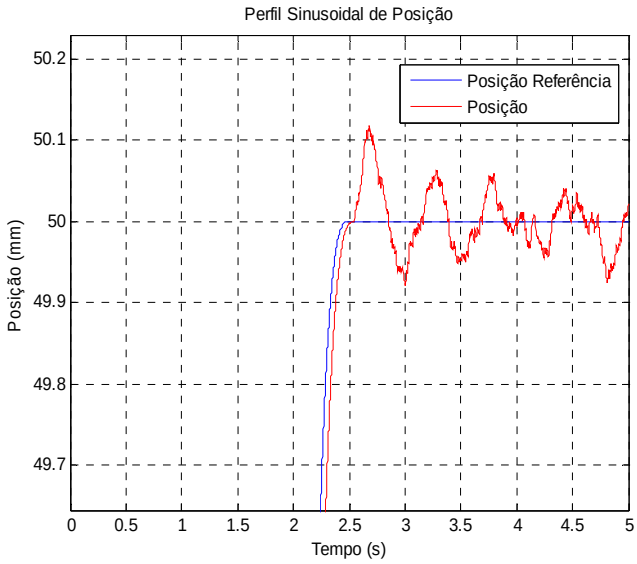


Figura 2. 18 – Resposta do Sistema a um Perfil Sinusoidal de Posição sem Referência de Velocidade

A perturbação verificada na figura 2.18, deve-se essencialmente ao efeito que o atrito estático provoca no sistema, tal situação não se verifica quando este é desprezado, ou se, além do uso da referência de posição também se utilizar a referência de velocidade. Desta forma, obtém-se uma melhor resposta tal como se pode verificar na figura 2.19, o que não elimina um ligeiro erro em regime permanente (na ordem dos 0.5E-3 mm), um valor insignificante uma vez que a resolução do *encoder* é de 0.77E-3 mm.

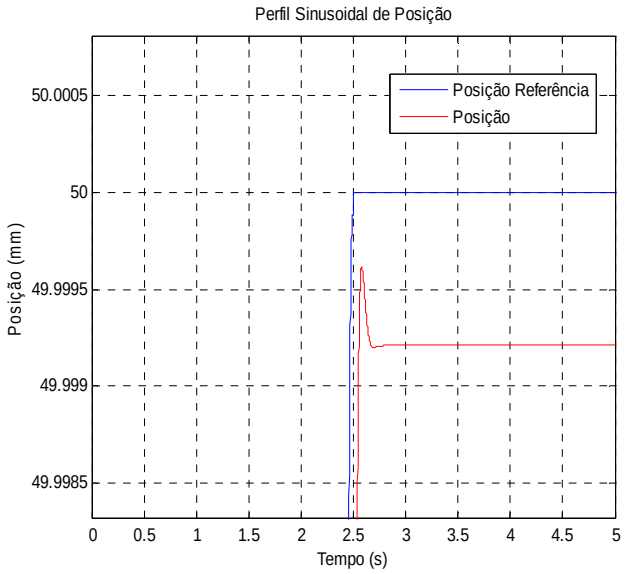


Figura 2. 19 – Resposta do Sistema a um Perfil Sinusoidal de Posição com Referência de Velocidade (ampliação)

Um outro aspecto que se verifica da análise do comportamento do sistema, é o facto de existir um pequeno atraso na reposta. Tal efeito não interfere significativamente no desempenho desejado do controlador implementado, mas poderá ser suprimido com a utilização de outro tipos de controladores, como é o caso dos controladores de ordem fraccionária.

2.3.3 Avaliação Experimental dos Controladores

Nesta secção serão apresentados alguns resultados experimentais, por forma a ser possível a análise do comportamento dos controladores de eixo para diferentes tipos de trajectórias.

Os controladores de eixo foram implementados com base nas simulações efectuadas através do *Matlab/Simulink*, sendo que a validação e os ajustes dos parâmetros de cada controlador foram efectuados através dos resultados experimentais.

Da análise dos resultados experimentais podem retirar-se algumas conclusões relativas ao desempenho dos controladores de eixo, mediante a menor ou maior amplitude de movimento, bem como a menor ou maior velocidade de execução das trajectórias.

Deste modo, o primeiro passo para a validação e ajuste dos parâmetros de cada controlador de eixo, foi a execução de uma trajectória entre dois pontos do espaço cartesiano com diferentes amplitudes de movimento e diferentes velocidades de execução da trajectória. Os resultados apresentados de seguida são correspondentes ao eixo X, uma vez que o procedimento para os restantes eixos é idêntico.

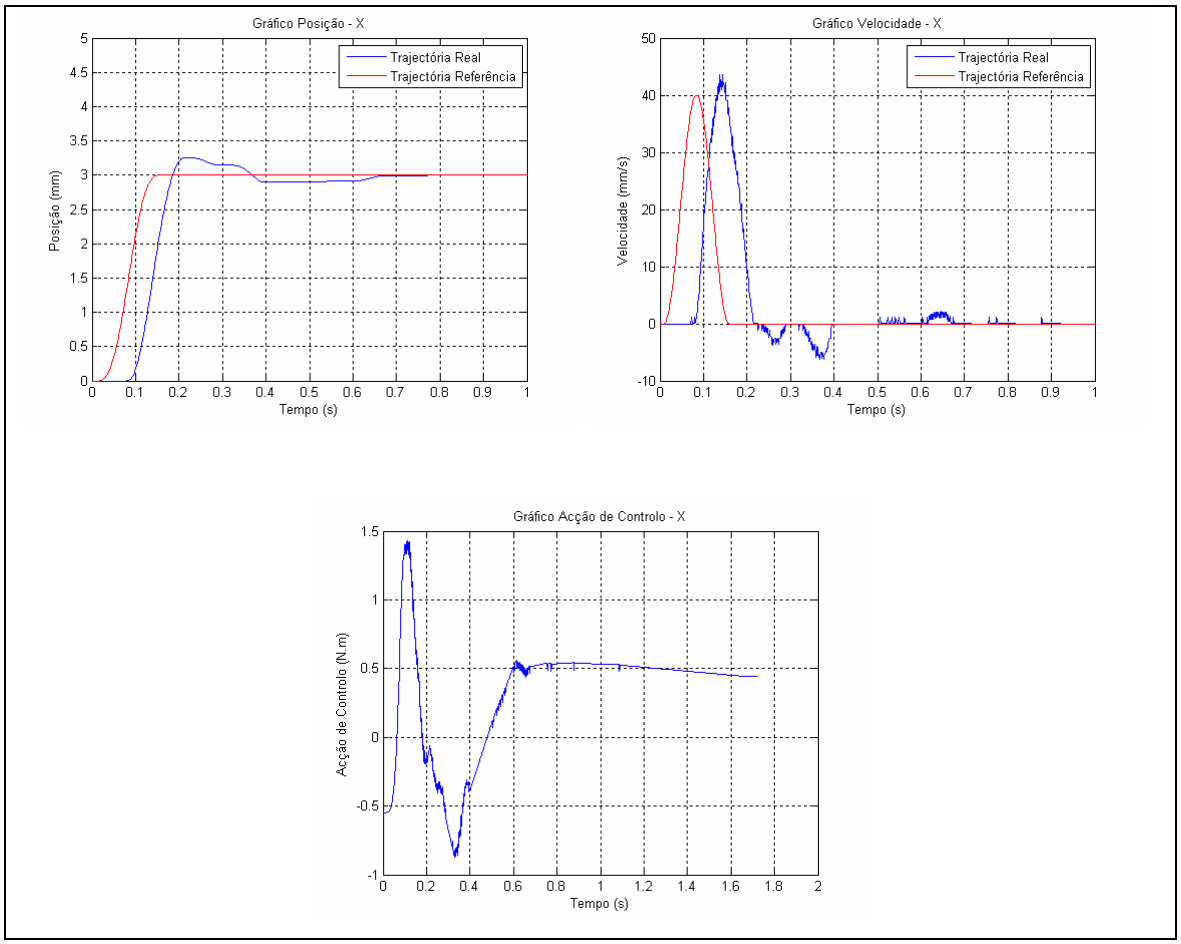


Figura 2. 20 – Trajectória do Eixo X (0 – 3 mm; 20 mm/s)

Da análise da figura 2.20 pode verificar-se, que para movimentos de baixa amplitude e baixa velocidade, a trajectória real difere ligeiramente das referências de posição e velocidade. Pode também constatar-se que a resposta do sistema apresenta um atraso e um ligeiro *overshoot*. A acção de controlo não chega a atingir a saturação, mas como se pode observar toma valores negativos durante a execução da trajectória, o que dá origem ao ligeiro *overshoot*.

O erro em regime permanente é praticamente nulo, como se pode observar na figura 2.20, o que demonstra que o controlador cumpre um dos requisitos da sua implementação, que corresponde a exactidão no posicionamento.

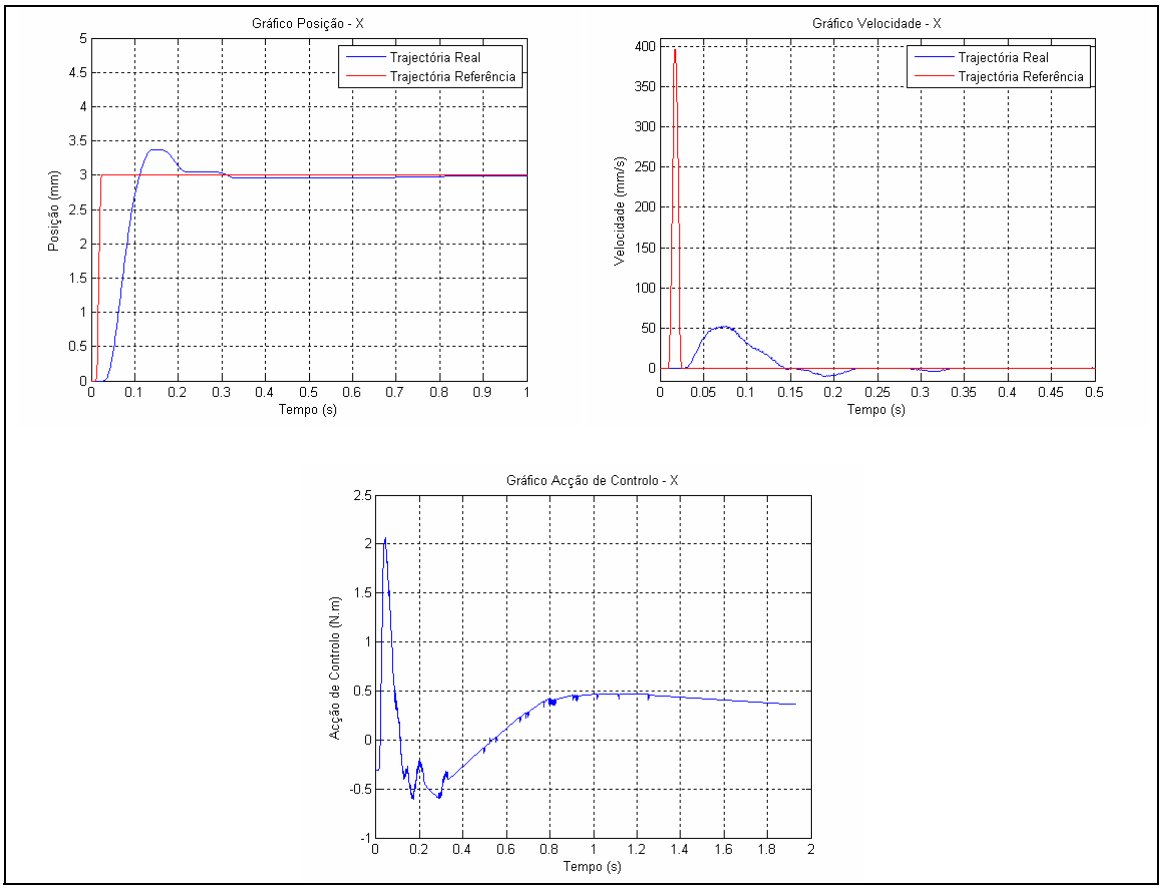


Figura 2. 21 – Trajectória do Eixo X (0 – 3 mm; 200 mm/s)

Neste caso (figura 2.21), com o aumento da velocidade verifica-se um aumento no atraso da resposta do controlador em relação às referências de posição e velocidade. Pode também constatar-se que a acção de controlo aumentou em relação ao resultado anterior (figura 2.20), mas verifica-se que a mesma não atingiu a saturação. Tal como aconteceu para a velocidade de 20 mm/s, para os 200 mm/s ocorreu um ligeiro *overshoot*, devido à acção de controlo tomar valores negativos durante a execução da trajectória. O valor do *overshoot* aumentou com o aumento de velocidade.

Em relação ao erro em regime permanente, constata-se que o seu valor aumenta com o aumento de velocidade, embora permaneça muito baixo.

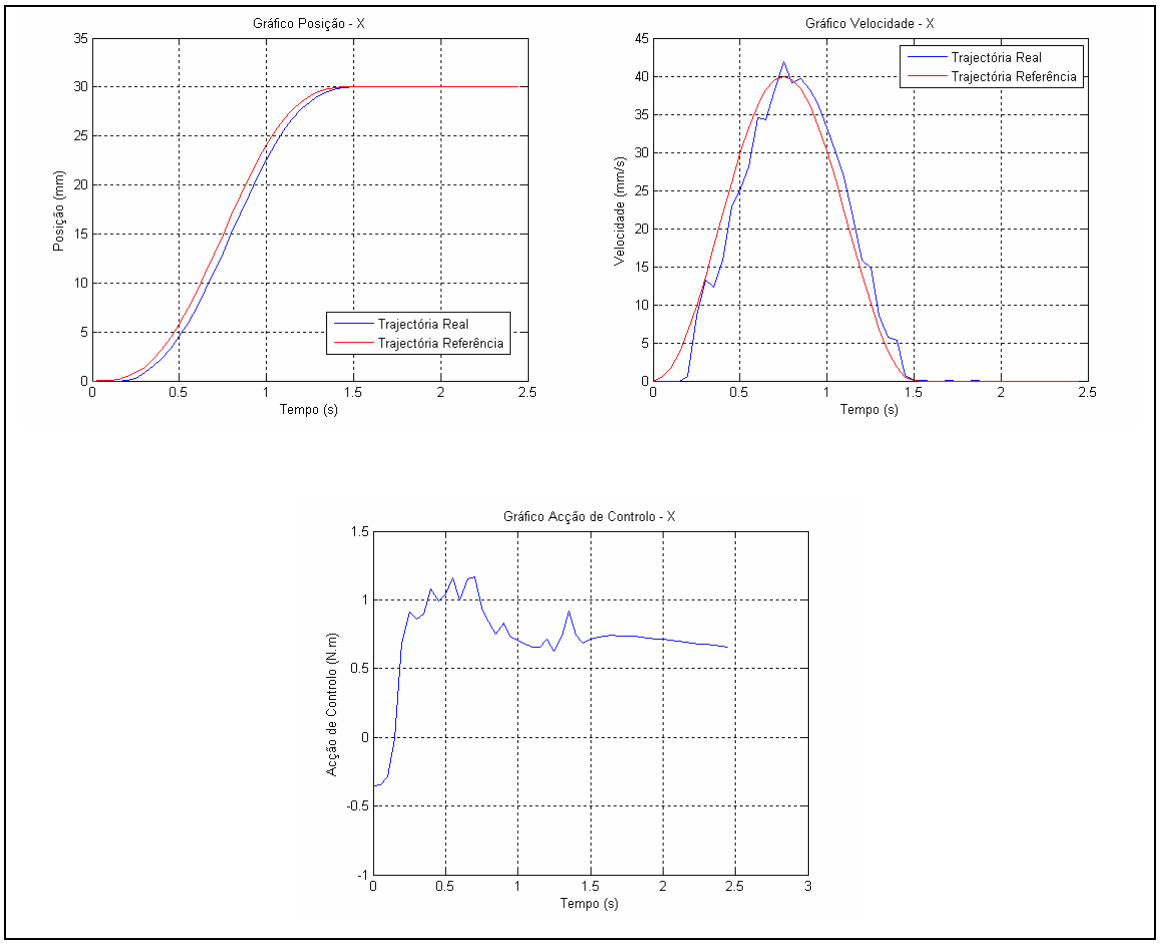


Figura 2. 22 – Trajectória do Eixo X (0 – 30 mm; 20 mm/s)

Da análise da figura 2.22 e por comparação com os resultados da figura 2.20, verifica-se que o aumento da amplitude de movimento, para o mesmo valor de velocidade, a resposta do controlador é mais próxima das referências de posição e velocidade. Sendo que, a acção de controlo não passa de valores positivos para negativos, ou seja, não ocorre *overshoot*.

Nota-se um ligeiro atraso na resposta do controlador, mas pode verificar-se que em regime permanente o erro de posição é desprezável, o que revela uma elevada precisão de posicionamento.

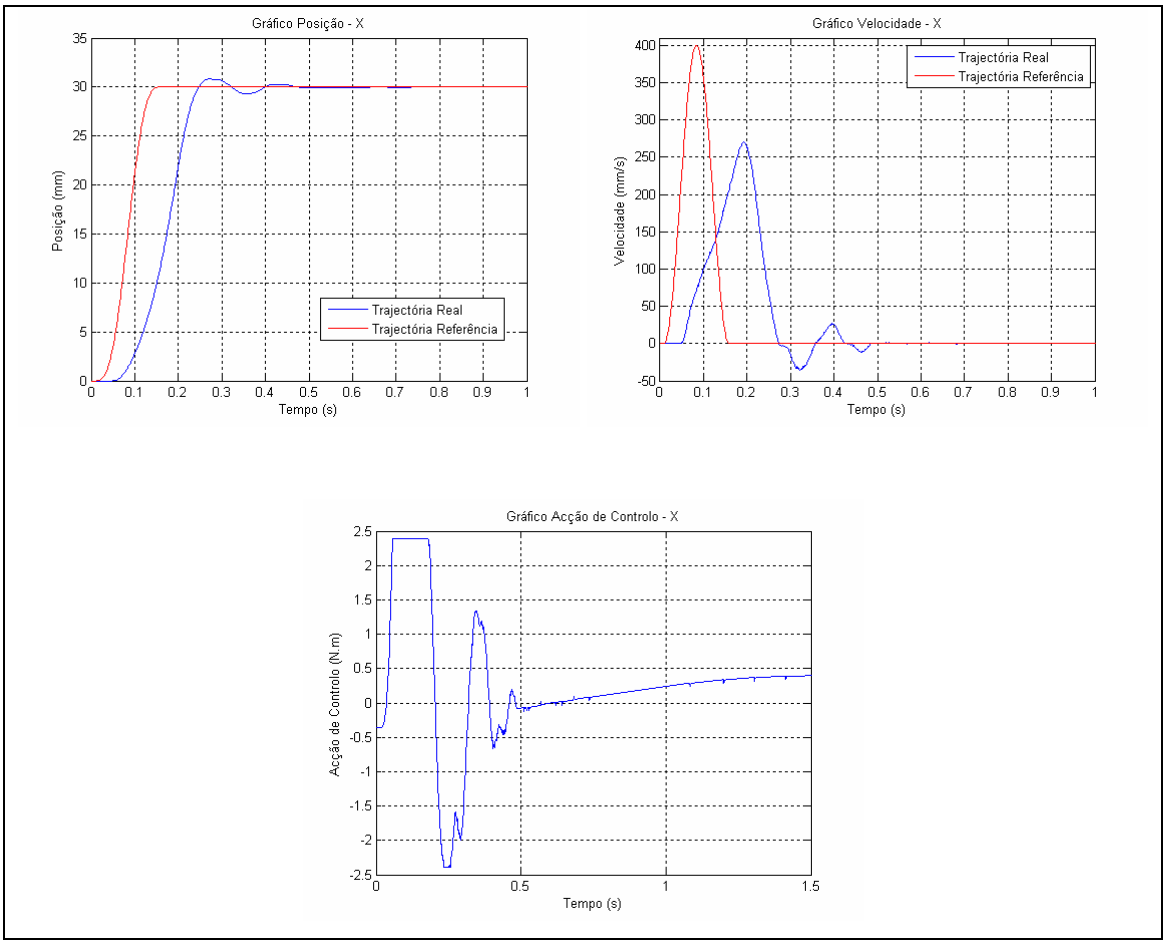


Figura 2. 23 – Trajectória do Eixo X (0 – 30 mm; 200 mm/s)

Os resultados da figura 2.23 mostram que, com o aumento de velocidade, a resposta do controlador deixa de conseguir acompanhar as referências de posição e velocidade. A acção de controlo acaba por atingir os valores de saturação, ou seja o controlador é levado aos limites. Verifica-se também a ocorrência de um pequeno *overshoot*, decorrente da acção de controlo passar de um valor positivo para um negativo. Ainda assim, o erro de posição em regime permanente continua a ser desprezável e a recuperação da situação de saturação é bastante boa.

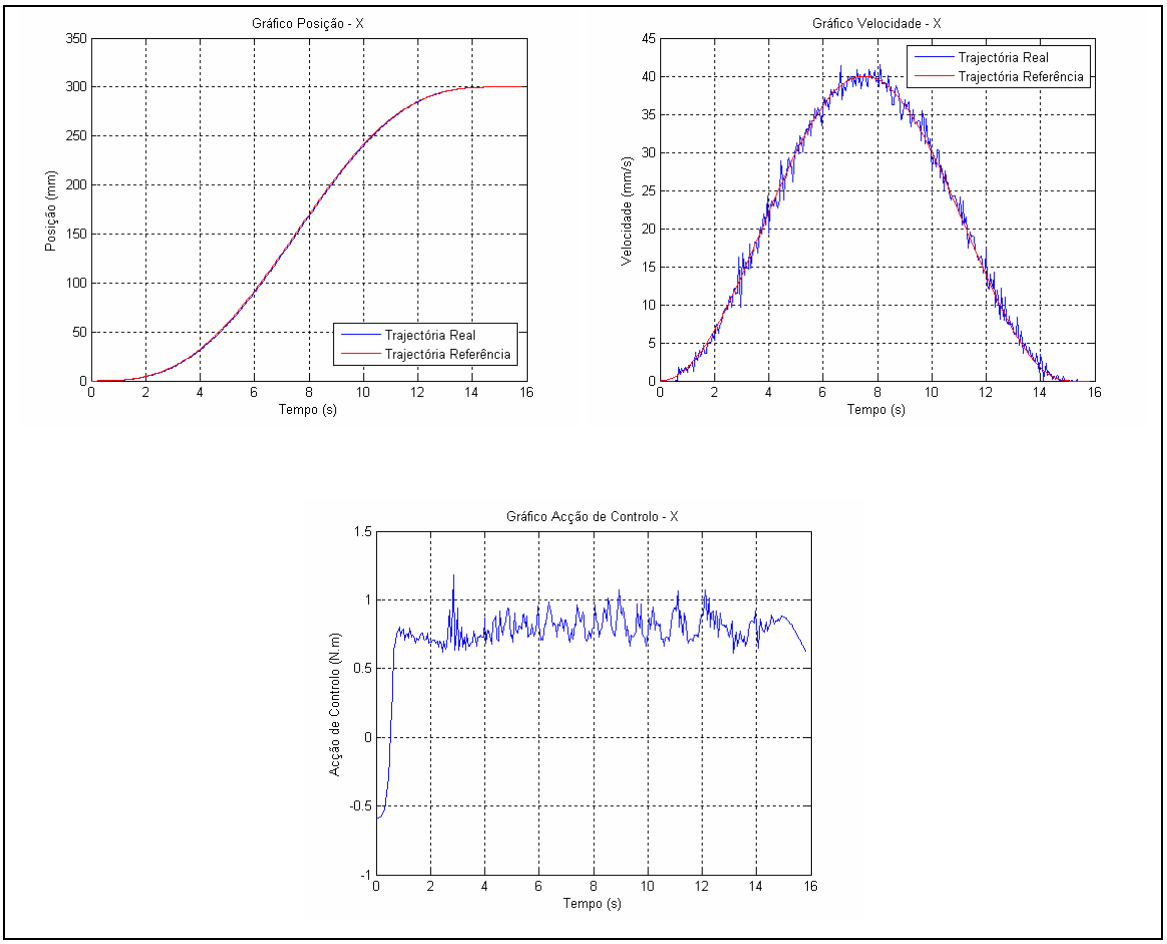


Figura 2. 24 – Trajectória do Eixo X (0 – 300 mm; 20 mm/s)

Da análise dos resultados da figura 2.24, pode concluir-se que, com aumento da amplitude do movimento, a resposta do controlador tende a aproximar-se das referências de posição e velocidade. Neste caso, o atraso da resposta do controlador é praticamente desprezável. Em relação ao erro em regime permanente pode dizer-se também que é desprezável.

Pode ainda verificar-se na figura 2.24 que, para baixas velocidades, o efeito do atrito estático do sistema se faz notar com maior intensidade. O efeito do atrito estático provoca uma perturbação na velocidade do eixo em movimento, tal como se pode observar na figura 2.24.

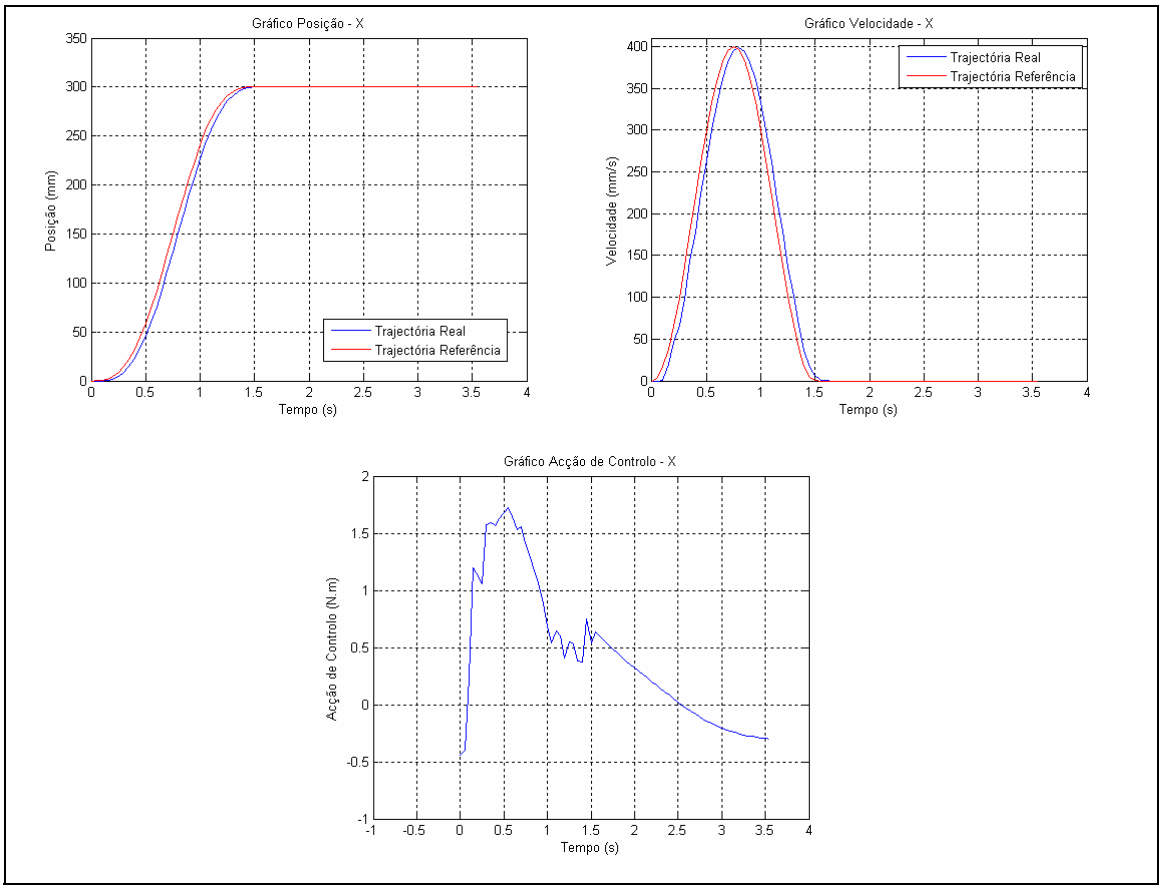


Figura 2. 25 – Trajectória do Eixo X (0 – 300 mm; 200 mm/s)

Comparando os resultados da figura 2.25 e os resultados da figura 2.24, pode constatar-se que o aumento da velocidade implica um ligeiro atraso na resposta do controlador em relação às referências de posição e velocidade. Outro aspecto que se pode verificar, é que o aumento da velocidade elimina a perturbação na execução da trajectória provocada pelo efeito do atrito estático.

Pode verificar-se ainda que tal como aconteceu nos resultados anteriormente apresentados, o erro em regime permanente permanece quase nulo (desprezável).

2.4 Geração de Trajectórias

Trajectória é a sequência temporal de posições assumidas pelo robô quando se movimenta no seu espaço de trabalho. A trajectória desejada é especificada pelo utilizador e gerada pelo Gerador de Trajectórias, tendo em conta a informação fornecida pelo utilizador e as restrições cinemáticas e dinâmicas do robô e do ambiente. O controlador do robô recebe a trajectória desejada, interpretando-a como uma referência que tentará fazer cumprir (figura 2.26).

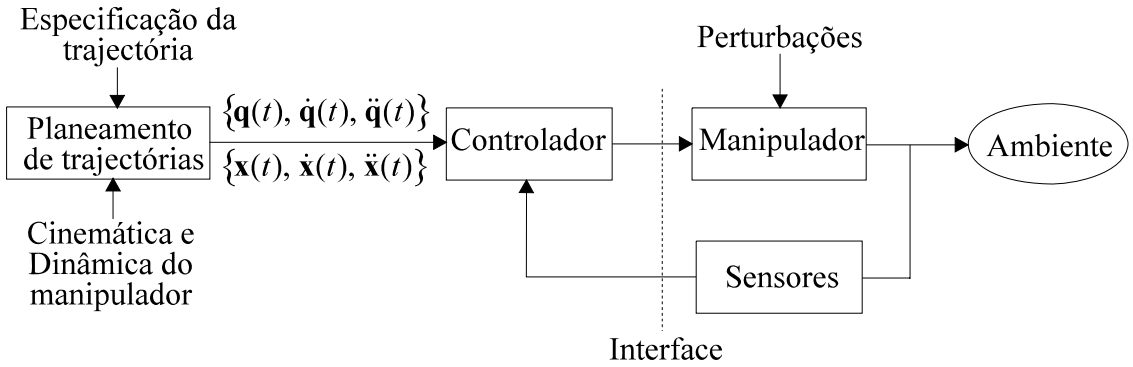


Figura 2. 26 – Sistema de controlo de um robô cartesiano (Lopes, 1994)

O planeamento de trajectórias pode ser visto como um bloco que aceita um conjunto de variáveis indicando as condições a que deve obedecer a trajectória devolvendo, em cada instante de amostragem, novos valores de posição, velocidade e aceleração para cada eixo do robô. Esses valores são depois usados como referências no bloco de controlo (figura 2.26).

São comuns duas abordagens distintas ao planeamento de trajectórias. A primeira requer apenas a especificação de um conjunto de pontos (onde o órgão terminal do robô deve passar em determinados instantes de tempo) e de um conjunto de condições a serem obedecidas (por exemplo, continuidade e "suavidade" na posição, na velocidade e na aceleração ao longo de toda a trajectória). Deste modo, o movimento entre pontos assim definidos pode ser qualquer, sendo suficiente interpolar uma função que satisfaça as condições pré-estabelecidas. As funções ortogonais tais como: funções polinomiais, funções trigonométricas, polinómios de *Chebychev*, polinómios de *Legendre* e polinómios de *Laguerre* podem ser usadas neste tipo de interpolação. A segunda

abordagem requer a especificação do caminho geométrico que o órgão terminal deve descrever (ex. linha recta, arco de circunferência). Neste caso é necessário gerar uma sequência de pontos que aproxime o caminho desejado.

Nesta subsecção é descrita a implementação dos geradores de trajectórias utilizados no modo de operação autónoma do robô cartesiano. A finalidade da utilização dos geradores de trajectórias corresponde à geração de referências de posição, velocidade e aceleração que são fornecidas ao controlador PID de cada eixo.

2.4.1 Movimento Linear entre dois Pontos

Caso se pretenda efectuar uma trajectória linear (segmento de recta) entre dois pontos tem de se utilizar um algoritmo de interpolação no espaço operacional. Um algoritmo bastante comum para gerar esse tipo de trajectória consiste em efectuar uma interpolação linear entre dois pontos, por exemplo $\mathbf{x}_0$ e $\mathbf{x}_1$, de acordo com um dado perfil de velocidade (Fu, *et al.*, 1987). Isto é:

$$\mathbf{x}(t) = \mathbf{x}_0 + p(t)(\mathbf{x}_1 - \mathbf{x}_0), 0 \le t \le T \qquad 2.34$$

ou

$$\dot{\mathbf{x}}(t) = \dot{p}(t)(\mathbf{x}_1 - \mathbf{x}_0), 0 \le t \le T \qquad 2.35$$

onde $p(t) \in [0,1]$ é um parâmetro escalar que especifica a distribuição de velocidade ao longo da trajectória e T o período de tempo de execução da trajectória. Dependendo da função $p(t)$ escolhida podem ser adoptados vários perfis de velocidade $\dot{\mathbf{x}}(t)$, tais como o constante, triangular, trapezoidal, parabólico ou sinusoidal (Lopes, 1994). Optou-se por usar o perfil sinusoidal (figura 2.27), uma vez que este tem a capacidade de gerar trajectórias "suaves" na posição e em todas as suas derivadas.

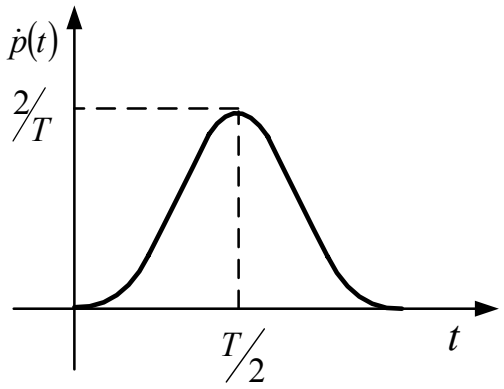


Figura 2. 27 – Perfil Sinusoidal de Velocidade

O perfil sinusoidal satisfaz as condições $\dot{p}(0) = \dot{p}(T) = \ddot{p}(0) = \ddot{p}(T) = 0$ (velocidade e acelerações nulas no início e no fim da trajectória) e é dado por:

$$p(t) = \frac{t}{T} - \frac{1}{2\pi} sen \frac{2\pi}{T} t \qquad 2.36$$

$$\dot{p}(t) = \frac{1}{T}\left(1 - \cos\frac{2\pi}{T} t\right), 0 \le t \le T \qquad 2.37$$

$$\ddot{p}(t) = \frac{2\pi}{T^2} sen \frac{2\pi}{T} t \qquad 2.38$$

Com este algoritmo de interpolação, equações (2.34) e (2.35), os perfis de velocidade $\dot{\mathbf{x}}(t)$ são proporcionais aos perfis $\dot{p}(t)$ e o movimento é realizado segundo o segmento de recta que une $\mathbf{x}_0$ a $\mathbf{x}_1$ no espaço operacional.

Para um perfil de velocidade sinusoidal a velocidade máxima é duas vezes a velocidade média especificada. Isso pode ser comprovado partindo das equações (2.34) e (2.37), chegando-se a:

$$\dot{\mathbf{x}}(t) = \frac{1}{T}\left(1 - \cos\frac{2\pi}{T}\right)(\mathbf{x}_1 - \mathbf{x}_0)t \qquad 2.39$$

sabendo que:

$$-1 \le \cos\left(\frac{2\pi}{T}\right) t \le 1, \quad 0 \le t \le T \tag{2.40}$$

então,

$$0 \le \dot{\mathbf{x}}(t) \le \frac{2}{T}(\mathbf{x}_1 - \mathbf{x}_0), \ 0 \le t \le T \tag{2.41}$$

como

$$Vmédia = \frac{\mathbf{x}_1 - \mathbf{x}_0}{T} \ \Rightarrow \ Vmáx = 2\,Vmédia \tag{2.42}$$

tal como se pretendia mostrar.

Em seguida é apresentado um fluxograma (figura 2.28) que ilustra a implementação de um algoritmo de interpolação linear com perfil de velocidade sinusoidal, equações (2.36), (2.37) e (2.38):

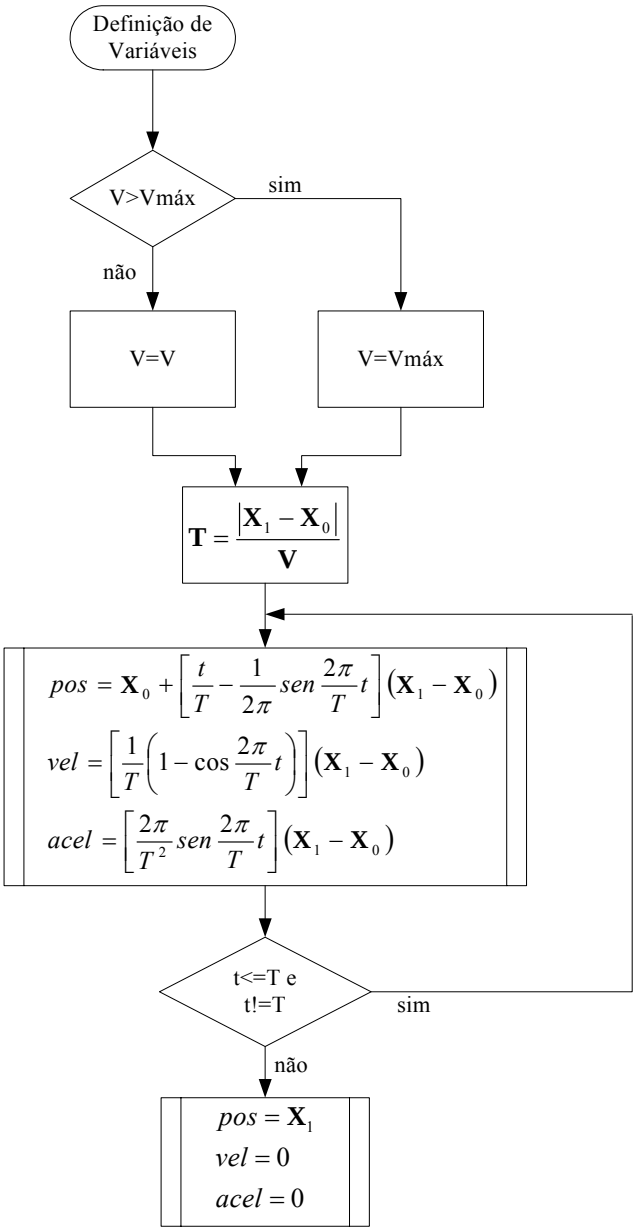


Figura 2. 28 – Algoritmo de um Gerador de Trajectórias de Interpolação Linear com Perfil Sinusoidal de Velocidade

O algoritmo apresentado foi implementado com o recurso a um bloco do *Simulink*, *S-Function Builder*, em linguagem *C*. Um exemplo do resultado da implementação deste algoritmo são as referências de posição, velocidade e aceleração da figura 2.29.

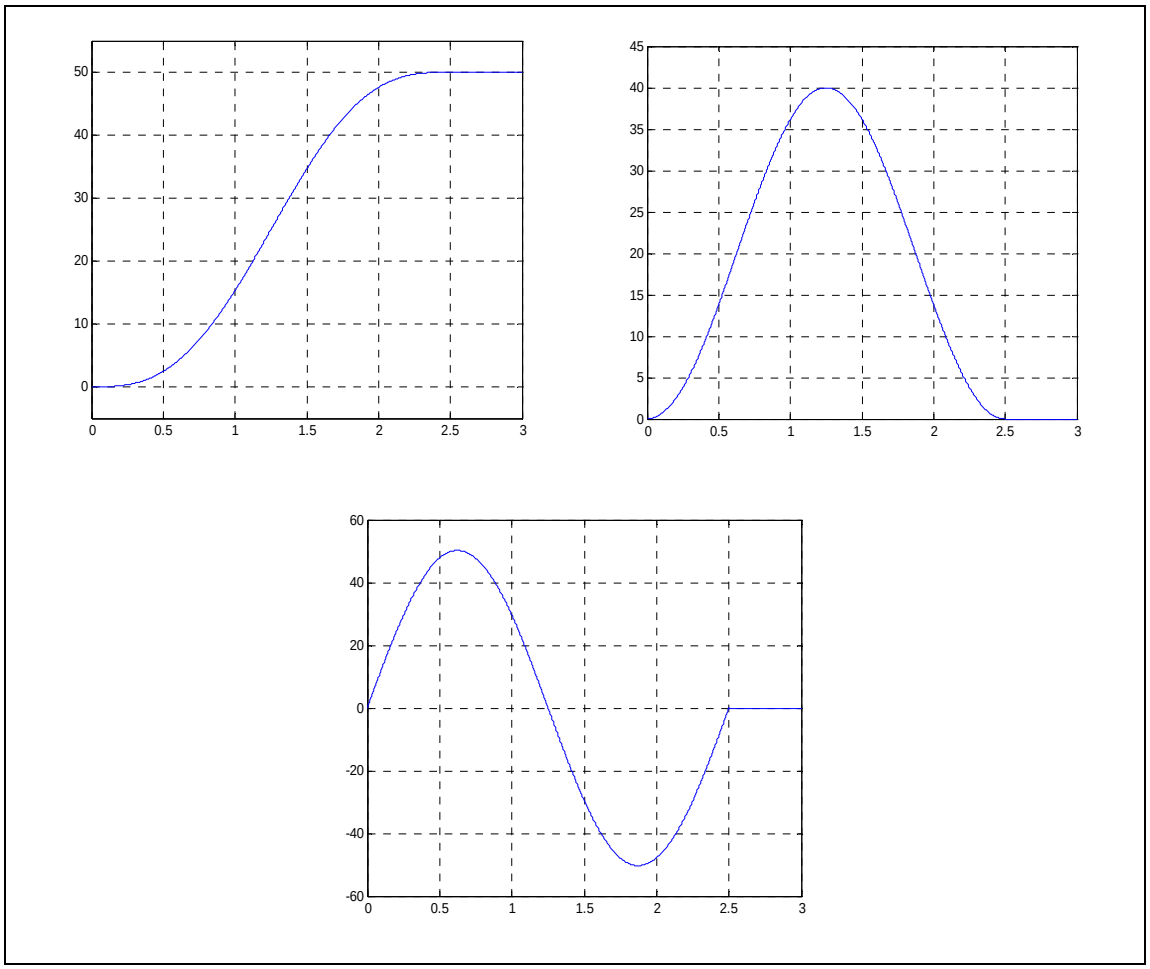


Figura 2. 29 – Gráficos de Posição, Velocidade e Aceleração de Referência

Para se efectuar uma trajectória linear com os três eixos em movimento, produzindo um segmento de recta no espaço tridimensional, é necessário decompor o valor da velocidade média definida para a execução da trajectória em três componentes. Assim para a execução da trajectória entre os pontos $\mathbf{P_0\,(x_0, y_0, z_0)}$ e $\mathbf{P_1\,(x_1, y_1, z_1)}$ com uma velocidade média $\mathbf{V_{méd}}$, a distância a percorrer é dada pela expressão que se segue:

$$Distância = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2} \qquad 2.43$$

sabendo que o período de tempo de execução da trajectória é obtido por:

$$T = \frac{Distância}{V_{méd}} \qquad 2.44$$

obtêm-se as três componentes da velocidade correspondentes aos eixos:

$$Vx = \frac{\left|x_0 - x_1\right| V_{méd}}{Distância};\ Vy = \frac{\left|y_0 - y_1\right| V_{méd}}{Distância};\ Vz = \frac{\left|z_0 - z_1\right| V_{méd}}{Distância} \qquad 2.45$$

Caso os valores calculados anteriormente excedam o valor máximo de velocidade permitido para os eixos, têm de ser recalculadas todas as componentes da velocidade. Primeiro é necessário identificar qual das componentes tem maior valor:

$$Vmáx = Maximo(Vx, Vy, Vz) \qquad 2.46$$

em seguida todas as componentes são recalculadas:

$$Vx = \frac{Vx\, Vmáx}{V_{méd}}; Vy = \frac{Vy\, Vmáx}{V_{méd}};\ Vz = \frac{Vz\, Vmáx}{V_{méd}} \qquad 2.47$$

Deste modo, não existe o perigo de se efectuarem movimentos com elevadas velocidades, assegurando que a trajectória final corresponde ao que se pretendia.

Após a implementação do algoritmo de geração de trajectórias lineares entre dois pontos, foram efectuados alguns testes experimentais, para avaliar os efeitos práticos da utilização deste tipo de geradores de trajectórias na execução de movimentos lineares. De seguida, são apresentados dois exemplos de movimentos lineares, entre dois pontos, no espaço tridimensional:

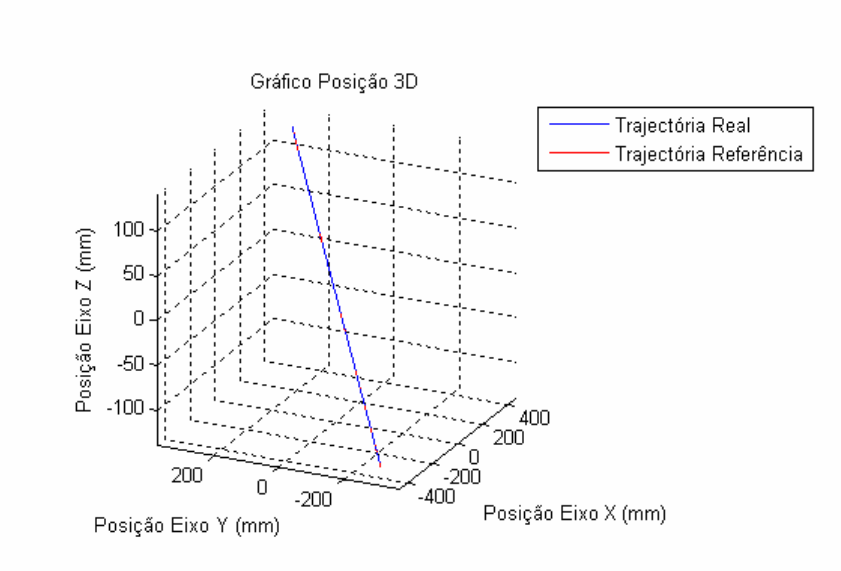


Figura 2. 30 – Movimento Linear entre dois Pontos no Espaço Tridimensional (V = 20 mm/s)

No exemplo da figura 2.30, pode visualizar-se a execução de uma trajectória linear entre dois pontos no espaço tridimensional, cujo movimento corresponde ao deslocamento de -300 mm a 300 mm no eixo X, -250 mm a 250 mm no eixo Y e -140 mm a 140 mm no eixo Z, com a velocidade média de 20 mm/s. Como se pode observar (figura 2.30) a trajectória é executada com os três eixos em movimento, sendo o resultado final um segmento de recta no espaço tridimensional.

Para a execução do movimento representado na figura 2.30, foi necessário calcular as velocidades médias correspondentes para cada eixo e aplicar o algoritmo de geração de trajectória linear entre dois pontos, de maneira a serem geradas as referências de posição e de velocidade para cada controlador de eixo na execução da trajectória. Assim, para cada eixo foram obtidos os seguintes resultados experimentais:

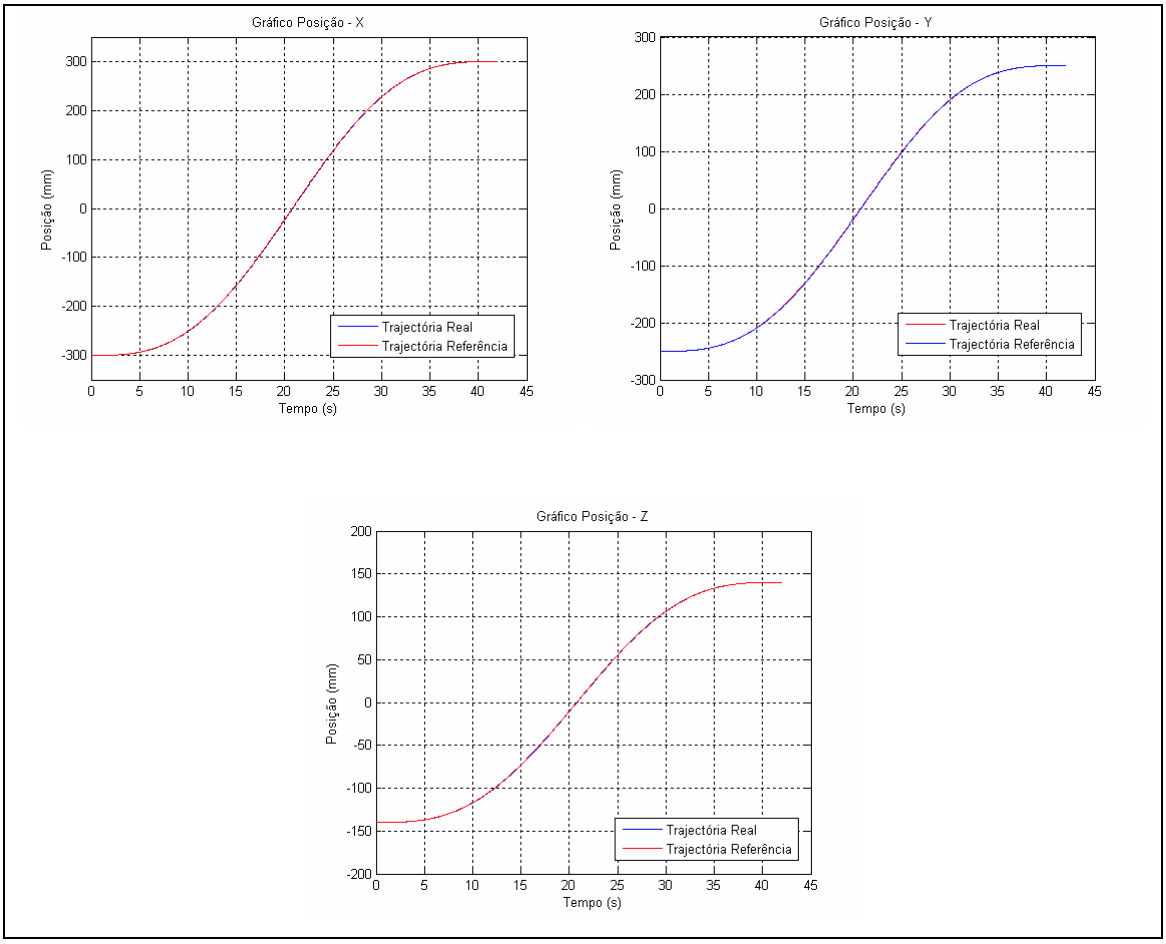


Figura 2. 31 – Gráficos de Posição do Movimento Linear entre dois Pontos (V = 20 mm/s)

Da análise da figura 2.31, verifica-se que para cada eixo foi gerada uma trajectória com um perfil sinusoidal, podendo-se constatar que todos os eixos terminam a execução do movimento no mesmo instante de tempo, deste modo o resultado final corresponde à execução de uma trajectória rectilínea (figura 2.30).

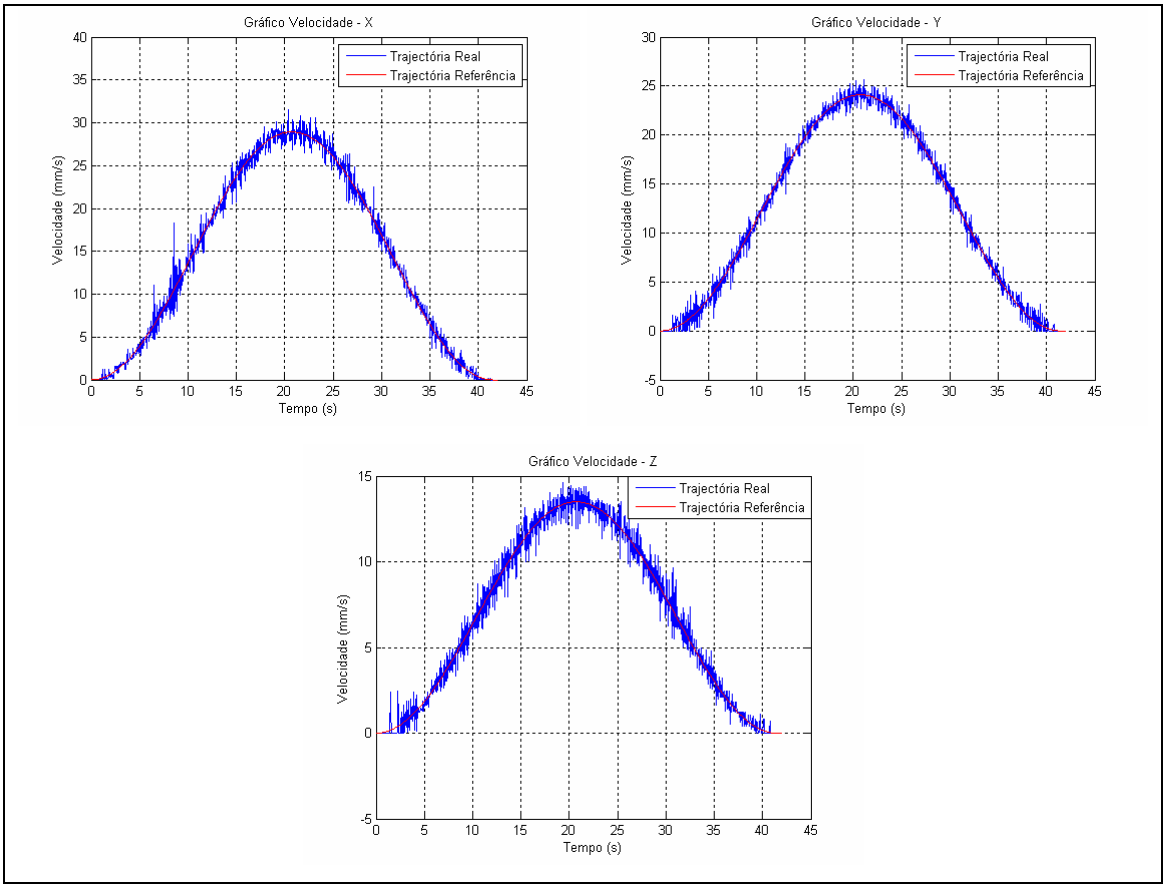


Figura 2. 32 – Gráficos de Velocidade do Movimento Linear entre dois Pontos (V = 20 mm/s)

Em relação à geração das referências de velocidade para cada eixo, pode verificar-se (figura 2.32) que correspondem a perfis sinusoidais, o que permite a execução da trajectória de uma forma "suave", deste modo, o desempenho dos controladores torna-se mais eficiente. Da análise da figura 2.32, pode constatar-se que cada eixo possui uma velocidade média que corresponde a metade da velocidade máxima, equação (2.42), sendo que, o valor da velocidade média de cada eixo foi obtido através das equações (2.45), (2.46) e (2.47).

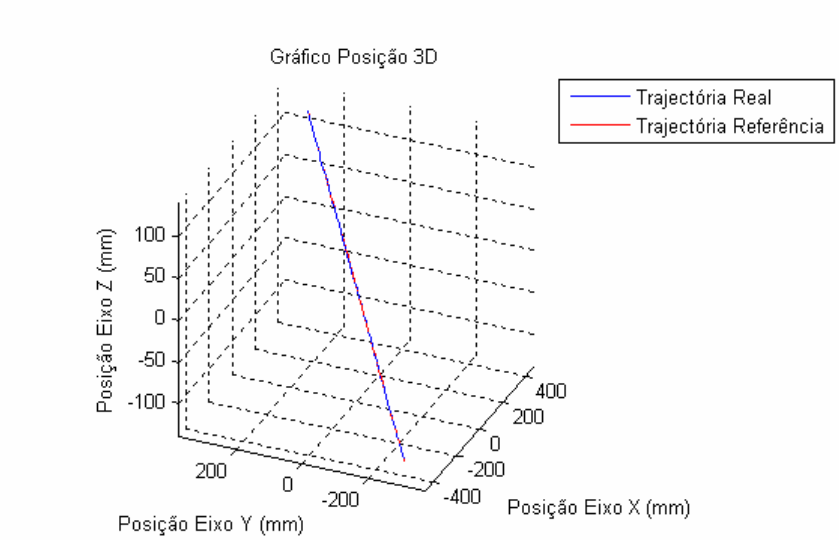


Figura 2. 33 – Movimento Linear entre dois Pontos no Espaço Tridimensional (V = 200 mm/s)

Na figura 2.33 pode observar-se a execução, no espaço tridimensional, do movimento apresentado na figura 2.30, sendo que a velocidade média foi elevada para 10 vezes mais. Pode visualizar-se que em ambas as figuras (2.30 e 2.33) o movimento do robô cartesiano é praticamente igual.

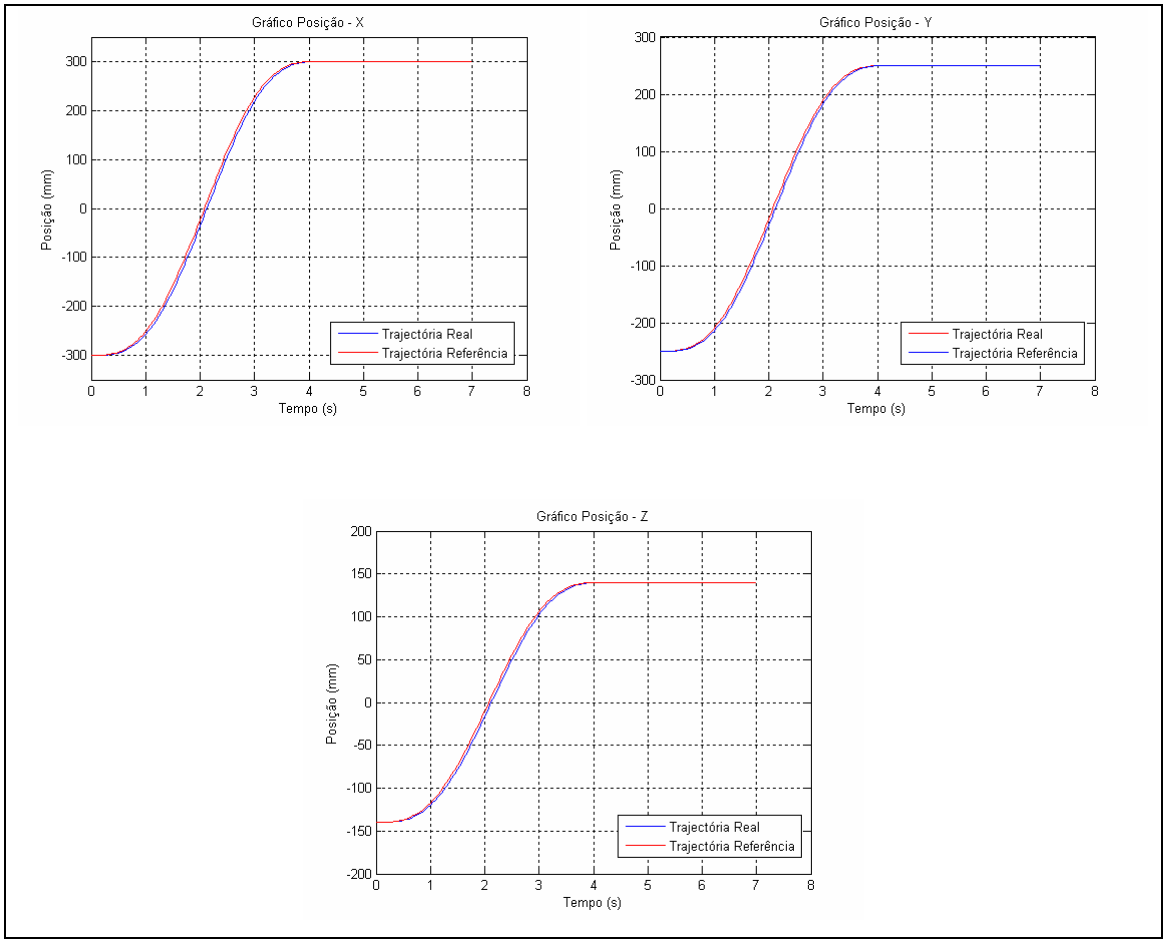


Figura 2. 34 – Gráficos de Posição do Movimento Linear entre dois Pontos (V = 200 mm/s)

Na figura 2.34 pode visualizar-se que as referências de posição, apesar do aumento da velocidade, continuam a ser do tipo sinusoidal e que a resposta de cada controlador consegue acompanhar bastante bem a respectiva referência.

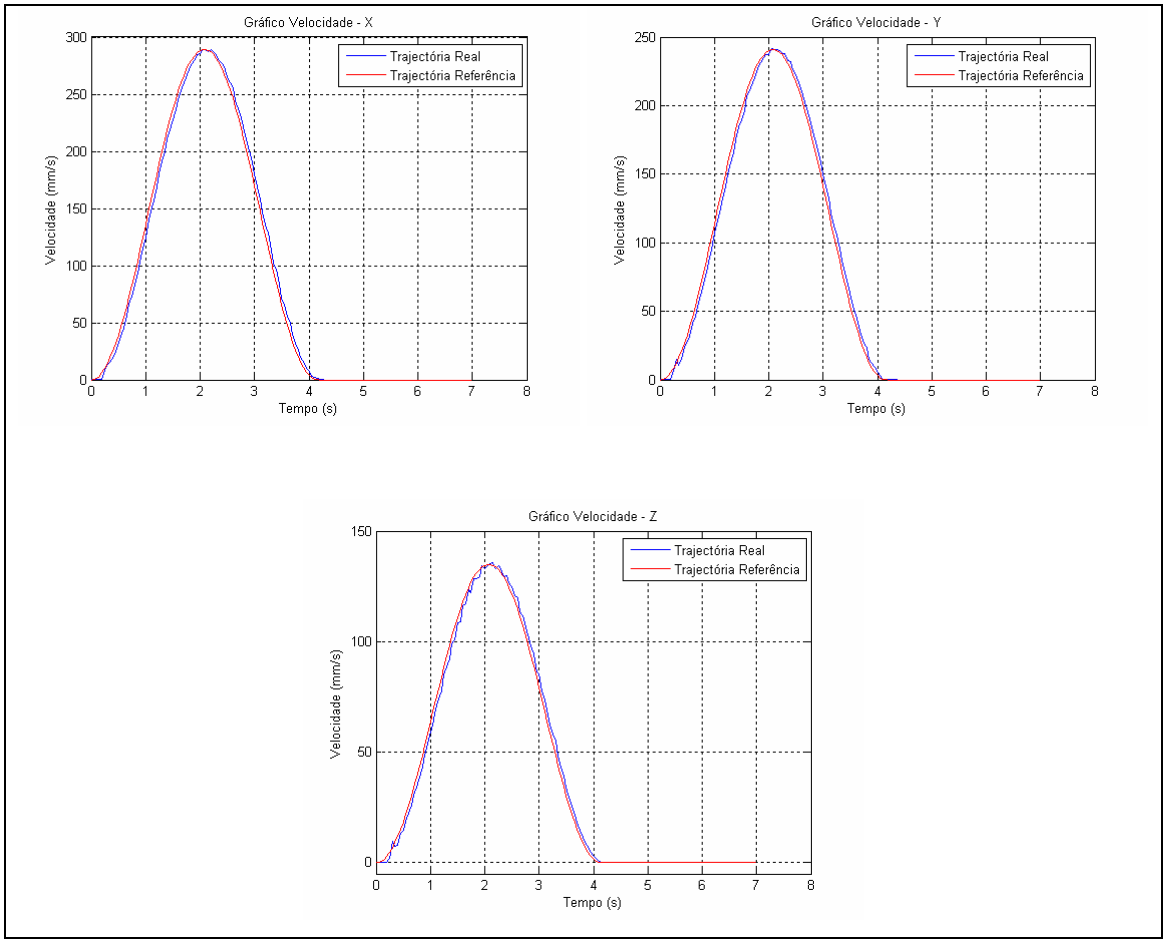


Figura 2. 35 – Gráficos de Velocidade do Movimento Linear entre dois Pontos (V = 200 mm/s)

Na figura 2.35 pode observar-se que a referência de velocidade apresenta um valor máximo 10 vezes maior em comparação com a figura 2.32. O período de tempo de geração das referências é 10 vezes menor do que a trajectória apresentada na figura 2.32, o que demonstra que algoritmo de geração de trajectórias se adapta para diferentes valores de velocidade.

2.4.2 Movimento Linear entre um Número arbitrário de Pontos

Uma trajectória ligando vários pontos por segmentos de recta pode ser conseguida utilizando uma interpolação linear (subsecção 2.4.1.) entre cada dois pontos consecutivos. No entanto, tal procedimento implica que o órgão terminal do robô tenha de parar em cada ponto de transição entre dois segmentos, ou seja, passando por esses pontos com velocidade nula. Este facto é justificado pelas descontinuidades na velocidade, nos pontos de transição entre dois segmentos, o que origina acelerações infinitas. Uma das soluções para o problema consiste em não passar pelos pontos de transição, aplicando para o efeito um algoritmo de interpolação capaz de "suavizar a trajectória" na vizinhança desses pontos (figura 2.36) (Paul, 1982; Lopes, 1994).

Considera-se então que se pretende ligar os pontos $\mathbf{x}_i$, $i = 1,\ldots,k$ (pelos quais o órgão terminal do robô deve passar no instantes t_i) por segmentos de recta no espaço cartesiano. A trajectória gerada deve ainda ser contínua na posição e nas suas duas primeiras derivadas para qualquer instante $t \in [0, T]$.

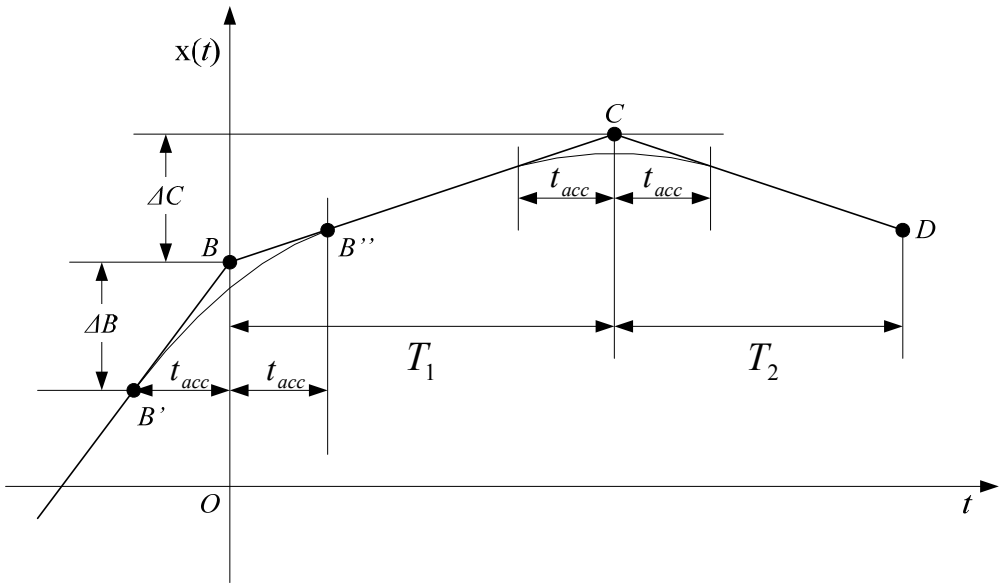


Figura 2. 36 – Transição entre dois segmentos (Trajectória Multi-ponto)

Para isso é necessário começar a preparar a transição um tempo t_{acc} (ponto *B'*) antes do fim do segmento corrente e completá-la um tempo t_{acc} (ponto *B''*) depois de começar o novo segmento. Na figura 2.36 representa-se parte de uma trajectória em que o robô acabou de passar no ponto *B'* a caminho do ponto *B*. No instante $-t_{acc}$ começa a dar-se a transição para o segmento *BC*. O tempo de duração da transição entre os dois segmentos é $2t_{acc}$ que permite, caso necessário, acelerar desde a máxima velocidade negativa até à máxima velocidade positiva ou *vice-versa*. Para se poder interpolar uma função polinomial na vizinhança do ponto de transição (entre os instantes $-t_{acc} \le t \le t_{acc}$) seria necessário um polinómio de quinta ordem, uma vez que se verifica que existem seis condições de fronteira (posição, velocidade e aceleração nos pontos *B'* e *B''*). No entanto, devido à simetria da região de transição, é suficiente interpolar um polinómio de quarta ordem. Assim, para o intervalo $-t_{acc} \le t \le t_{acc}$ a posição, velocidade e a aceleração são dadas por (Paul, 1982):

$$\mathbf{x}(t) = \left[\left(\Delta C \frac{t_{acc}}{T_1} + \Delta B\right)\right] h + B + \Delta B \qquad 2.48$$

$$\dot{\mathrm{x}}(t) = \left[\left(\Delta C \frac{t_{acc}}{T_1} + \Delta B\right)(1.5 - h)2h^2 - \Delta B\right] \frac{1}{t_{acc}} \qquad 2.49$$

$$\ddot{\mathbf{x}}(t) = \left(\Delta C \frac{t_{acc}}{T_1} + \Delta B\right)(1 - h)\frac{3h}{t_{acc}^2} \qquad 2.50$$

onde

$$h = \frac{t + t_{acc}}{2t_{acc}},\ -t_{acc} \le t \le t_{acc} \qquad 2.51$$

$$\Delta C = C - B \qquad 2.52$$

$$\Delta B = B' - B \qquad 2.53$$

Fora desse intervalo (fora das regiões de transição) a velocidade é constante:

$$\mathbf{x}(t) = \Delta C + B \tag{2.54}$$

$$\dot{\mathbf{x}}(t) = \frac{\Delta C}{T_1} \tag{2.55}$$

$$\ddot{\mathbf{x}}(t) = 0 \tag{2.56}$$

$$h = \frac{t}{T_1} \tag{2.57}$$

Segundo Vukobratovic e Kircanski (1996), o tempo t_{acc} pode ser estimado a partir da equação:

$$t_{acc} = \frac{3}{4}\frac{\Delta v}{a} \tag{2.58}$$

onde Δv corresponde ao incremento de velocidade entre dois segmentos e a é a aceleração máxima permitida.

Com este algoritmo consegue-se "suavizar" a trajectória na proximidade dos pontos de transição entre segmentos. No entanto existem dois pontos onde a velocidade é descontínua, o que implica que a aceleração seja infinita, que são os pontos inicial e final. A solução encontrada para este problema, consiste em repetir o primeiro ponto um tempo t_{acc} antes e repetir o último ponto um tempo t_{acc} depois dos, respectivamente, primeiro e último pontos especificados para a trajectória. A trajectória passa assim a ter mais dois pontos de transição (mais dois segmentos) que são tratados como os restantes.

Assim que a transição começa (ponto *B'*) pode iniciar-se o cálculo da trajectória entre os pontos *C* e *D*. No instante de tempo $t = T_1 - t_{acc}$ são feitas as seguintes atribuições para iniciar a transição para o segmento seguinte:

$T_1 = T_2;$

$B' = X;$ posição actual

$B = C;$

$C = D;$

$\Delta C = C - B;$

$\Delta B = B' - B;$

$t = -t_{acc;}$ "*reset*" do tempo

De seguida serão apresentados alguns resultados experimentais, que demonstram a implementação do algoritmo descrito anteriormente.

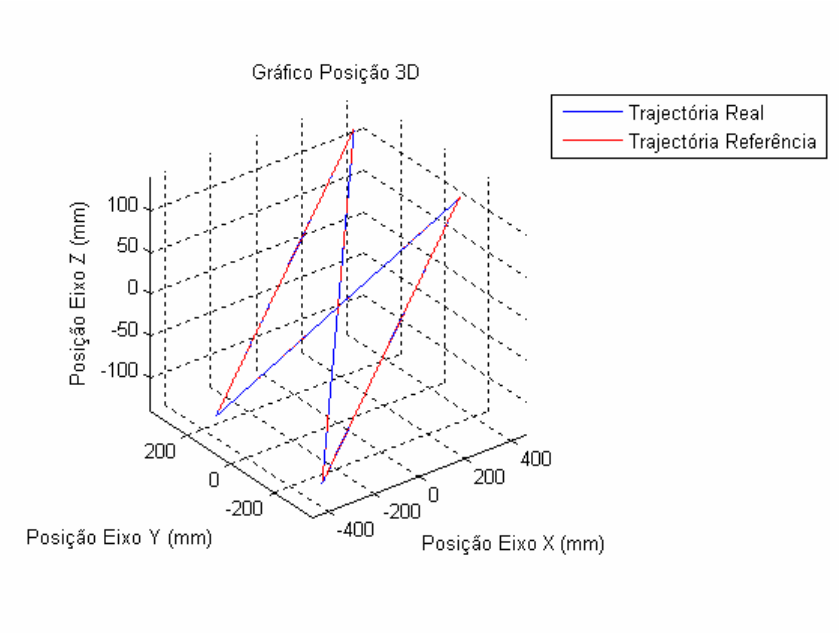


Figura 2. 37 – Movimento Linear Multi-ponto no Espaço Tridimensional (V = 20 mm/s)

No exemplo da figura 2.37, pode visualizar-se a execução de uma trajectória multi-ponto no espaço tridimensional. Para a execução desta trajectória foi utilizado o algoritmo descrito anteriormente.

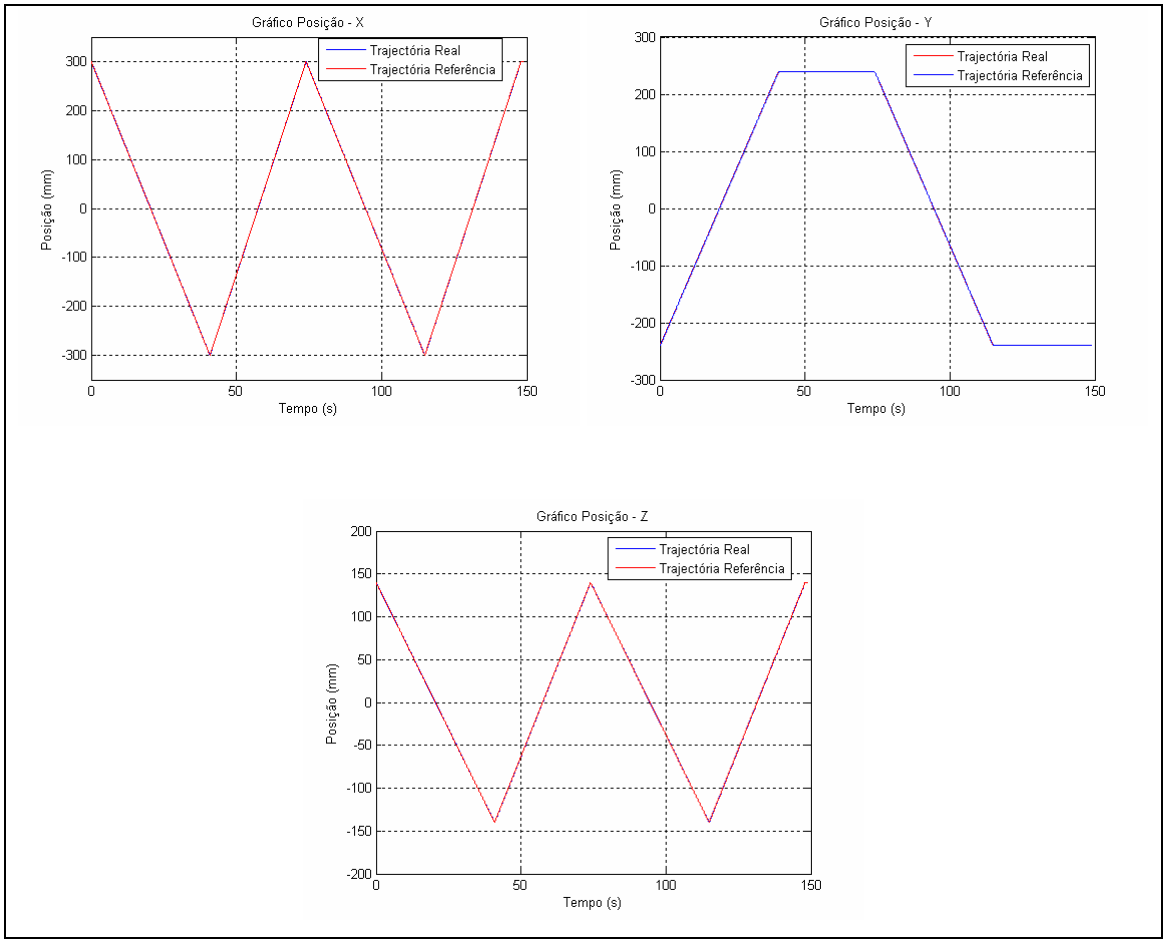


Figura 2. 38 – Gráficos de Posição do Movimento Linear Multi-ponto (V = 20 mm/s)

Na figura 2.38 podem visualizar-se as referências de posição geradas para cada um dos eixos, que estão na origem da trajectória tridimensional da figura 2.37. Estas referências foram geradas através da implementação do algoritmo ilustrado na figura 2.36.

Na figura 2.39 podem visualizar-se as referências de velocidade geradas para cada eixo, decorrentes da implementação do algoritmo de trajectória multi-ponto, bem como as trajectórias executadas.

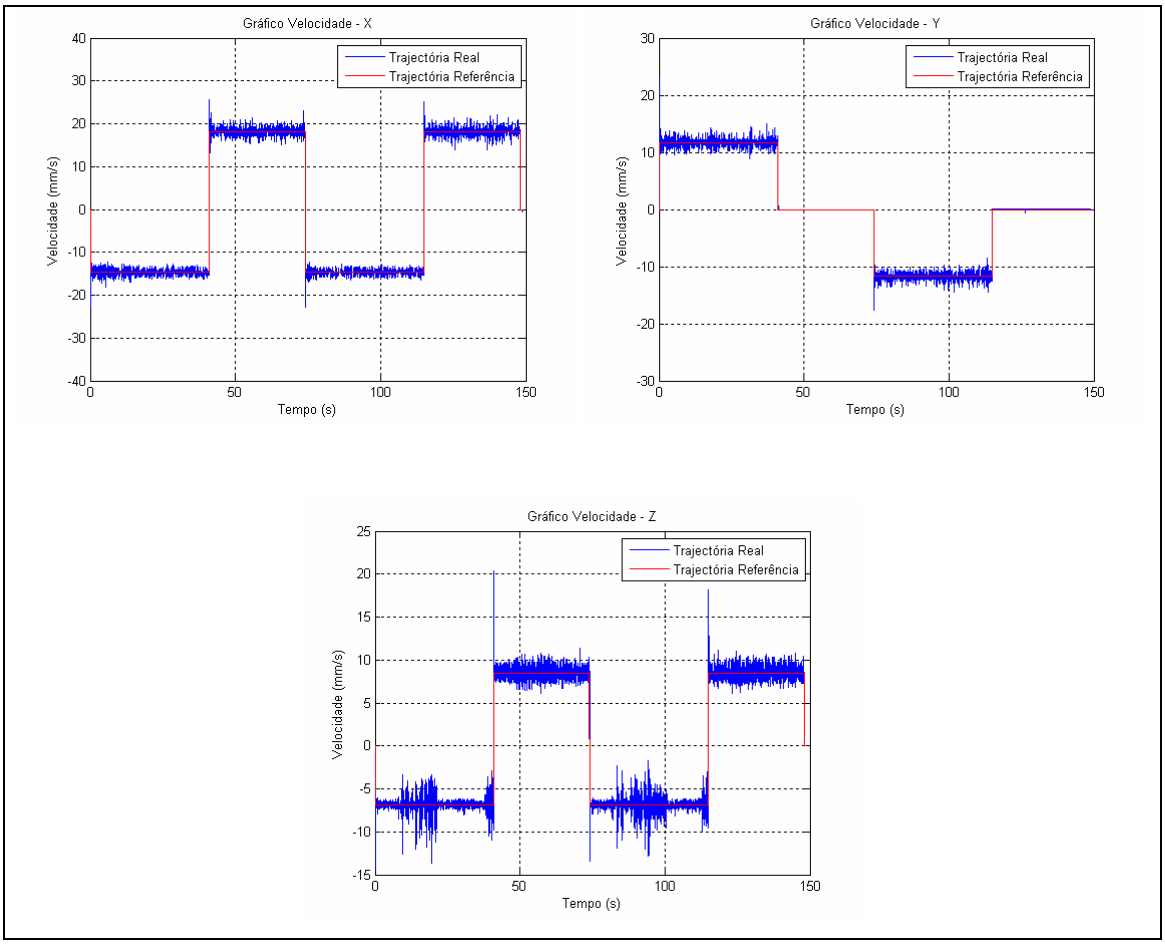


Figura 2. 39 – Gráficos de Velocidade do Movimento Linear Multi-ponto (V = 20 mm/s)

2.5 Arquitectura de Controlo

Nesta secção será descrita toda a arquitectura de controlo do sistema, o *software* utilizado e a sua interacção com o *hardware*. É descrita a configuração *Host-Target* utilizada como modelo de controlo. É também descrita a máquina de estados, *Statefow*, que gere o comportamento global do sistema, bem como o modelo *Simulink* que implementa as acções correspondentes ao estado em que o sistema se encontra.

2.5.1 *Software* de Controlo (*Matlab – Simulink*)

Um item importante no desenvolvimento da estratégia de controlo do sistema é a escolha apropriada do sistema operativo de tempo-real. Se a arquitectura de controlo for baseada em *PC* existe uma variedade de alternativas, que passam pela utilização de sistemas operativos de tempo-real dedicados (ex. *QNX*, *VxWorks*, *VRTX*), ou pelo uso de módulos de tempo-real (ex. *RTX*, *RTLinux*, *RTAI*) (Kim, *et al.*, 2004) próprios para sistemas operativos que não são de tempo-real, como o *Windows* e o *Linux*.

Uma outra alternativa corresponde à utilização de *software* que permite desenvolvimento, simulação e validação de estratégias de controlo, e que dispõe também de um *kernel* de tempo-real, como é o caso do *LabVIEW* (Hercog, *et al.*, 2005) e do *Matlab/Simulink* (Bisták e Žáková, 2003; Low, *et al.*, 2005). Esta solução apresenta particulares vantagens quando o objectivo principal corresponde ao desenvolvimento de uma arquitectura de controlo aberta, onde tanto os controladores como os parâmetros de controlo podem ser facilmente modificados, simulados, implementados e validados. Para além disso, este tipo de solução dispõe de ambientes de desenvolvimento de fácil utilização. Outro aspecto positivo a salientar é o facto deste tipo de aplicações dispor de *device drivers* de tempo-real que permitem uma interface com uma vasta gama de cartas I/O e ainda permitem uma fácil integração com aplicações gráficas de interface com o operador, desenvolvidas através de ferramentas como *Microsoft Visual Studio* (*Visual Basic*, *C*$^{++}$ ou *C*$^{\#}$).

A escolha do *software* de desenvolvimento, teste e validação da estratégia de controlo, bem como a sua implementação recaiu sobre o *Matlab/Simulink*. Sendo possível, deste modo, a modelação e simulação do comportamento do sistema a controlar, bem como a simulação e validação dos controladores de eixo e de toda a estratégia de controlo, isto graças à ferramenta de desenvolvimento *Simulink*. O espaço de trabalho *Simulink* é caracterizado pela sua intuitiva interface gráfica, disponibilizando uma biblioteca de blocos. Sendo que na base dos sistemas dinâmicos estão as funções de transferência, estas podem facilmente ser implementadas com o recurso a diagramas de blocos, blocos estes que na maioria já existem na biblioteca de blocos do *Simulink*. Outra vantagem consiste na possibilidade de implementação por parte do utilizador dos seus próprios blocos, com as especificações pretendidas, com o recurso a programação quer em linguagem *M* (própria do *Matlab*), ou em linguagem *C*.

Uma das aplicações do *Matlab/Simulink*, bastante importante no desenvolvimento da estratégia de controlo foi o *Stateflow*, que representa uma ferramenta poderosa no desenvolvimento e simulação de sistemas comandados por eventos. O *Stateflow* disponibiliza uma interface gráfica muito intuitiva para o utilizador, o seu conceito baseia-se numa máquina de estados. O utilizador tem a possibilidade de criar um diagrama com vários estados (em que cada estado executa uma determinada acção), definir as transições entre os vários estados, bem como os eventos que podem levar a mudanças dos mesmos.

Outra mais valia da solução *Matlab/Simulink* é a possibilidade de utilização de aplicações de tempo-real como é o caso do *Real Time Windows Target* ou *xPC Target*. Sendo que o *xPC Target* como uma solução *stand-alone* apresenta mais vantagens do que o *Real Time Windows Target*. A utilização do *xPC Target* permite ao utilizador o desenvolvimento de toda a estratégia num *PC*, ao qual se dá nome de *Host PC*, e posteriormente descarregar a aplicação de controlo para o *Target PC*, este último é responsável pela execução em tempo-real dessa aplicação de controlo.

2.5.2 *xPC Target* – Configuração *Host/Target*

Como já foi referido anteriormente o *xPC Target* representa um ambiente de tempo-real que disponibiliza ferramentas como *device drivers* que permitem a interface entre a aplicação e o *hardware* de controlo. Permite também que o algoritmo de controlo seja executado em exclusividade e em tempo-real por intermédio de um *kernel* de tempo-real de 32 bits.

A aplicação de controlo desenvolvida com o auxílio do *Simulink* é convertida em código portátil e optimizado, utilizando a ferramenta *Real Time Workshop* (*RTW*). O *RTW* foi configurado de modo a utilizar o compilador *Microsoft Visual C/ C*$^{++}$. Depois da aplicação de controlo ser compilada, o ficheiro resultante da compilação é então transferido para o *Target PC* onde é executado sobre um *kernel* de tempo-real.

No início, o *kernel* de tempo-real é carregado no *Target PC* por intermédio de uma disquete de arranque. O processo de geração dessa disquete de arranque é descrito de seguida:

- Na linha de comandos do *Matlab* digitar: **xpcexplr**

O comando anterior invoca a ferramenta *xPC Target Explorer*, figura 2.40, que permite a configuração do canal de comunicação *Host/Target* e dos parâmetros do *kernel* de tempo-real, bem como a geração da disquete de arranque e a transferência de aplicações do *Host PC* para o *Target PC*.

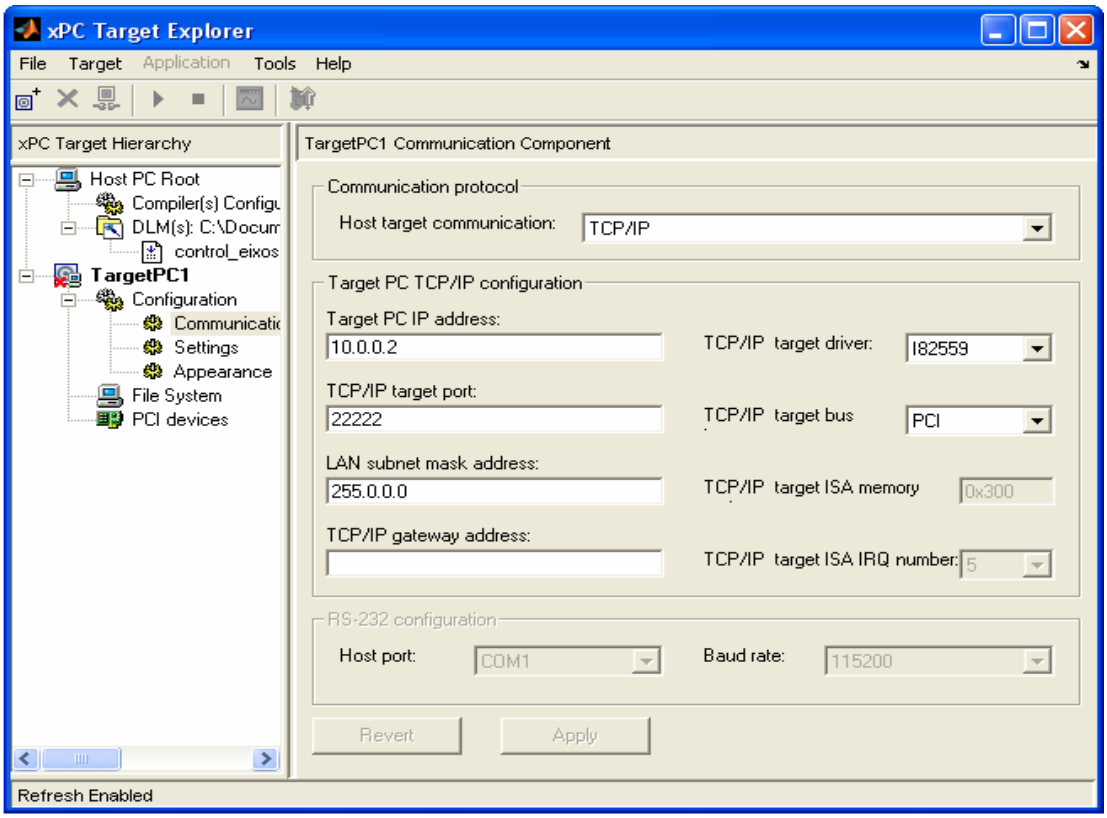


Figura 2. 40 – Configuração da comunicação *Host/Target*

A comunicação entre o *Host PC* e o *Target PC* pode ser efectuada via porta série (*RS232*) ou utilizando o protocolo *TCP/IP* (figura 2.41). Para a realização deste trabalho foi escolhida a comunicação *TCP/IP* uma vez que esta apresenta velocidades de comunicação muito superiores relativamente à comunicação *RS232*, podendo atingir velocidades de transferência entre 10/100 Mbps (1 Mbps – 1048576 bps), dependendo das características rede. Numa comunicação *RS232* a velocidade máxima permitida é de 115200 bps. Uma outra vantagem do uso de uma comunicação *TCP/IP* em relação ao *RS232* deve-se ao facto de possibilitar maiores distâncias entre o *Host* e o *Target*, sendo necessário para o efeito o uso *repeaters* e *gateways*.

Numa comunicação *TCP/IP* entre o *Host* e o *Target*, em que ambos devem possuir cartas de rede, são possíveis dois tipos de ligação. Uma delas corresponde a uma ligação dos dois *PC*'s por cabo *UTP* (cabo de rede) a uma rede local (*LAN*). Outra possibilidade é estabelecer uma ligação dedicada (directa) através de um cabo *crossover*. Neste caso, cada um dos *PC*'s deve possuir um *IP* estático.

Para a realização deste trabalho optou-se por efectuar uma ligação dedicada (cabo *crossover*), uma vez que a ligação a uma rede do tipo *LAN* acarreta problemas associados ao controlo do sistema global, tais como atrasos temporais e perdas de informação, que têm de ser geridas eficazmente pelo controlador do sistema, não sendo esse um dos objectivos deste trabalho.

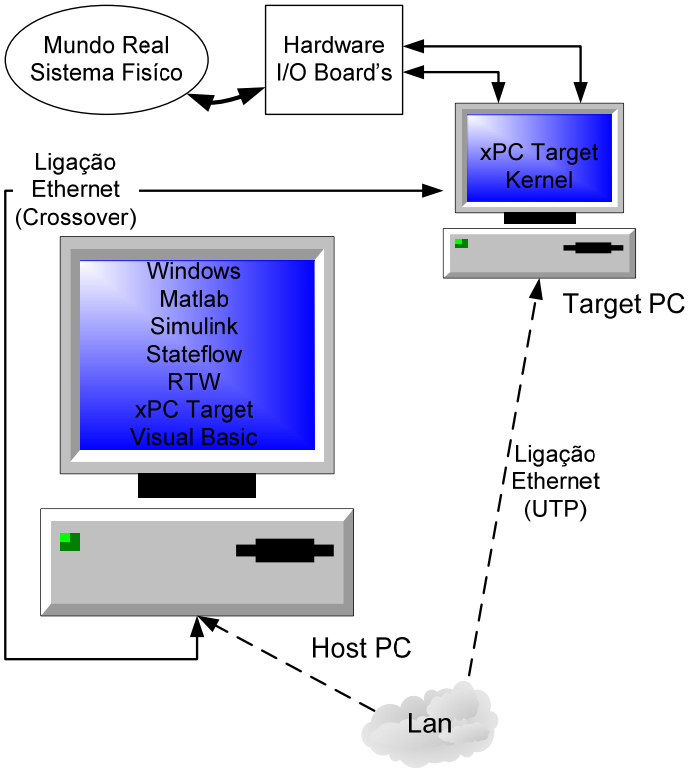


Figura 2. 41 – Arquitectura de Controlo do Sistema

No processo de configuração do canal de comunicação *TCP/IP* é necessária a indicação do IP estático e o *Port Number* do *Target PC*, bem como a indicação da máscara da sub-rede *LAN* em que está inserido o *Target PC*. Em relação ao endereço de *gateway* a sua indicação só é necessária se o canal de comunicação não for dedicado. É necessário também a indicação do *driver* da carta de rede que está instalada no *Target PC*. Esta carta tem de ser específica uma vez que a lista de cartas suportadas pelos *drivers* é limitada, desta forma, foi necessária a aquisição da carta de rede *Intel PRO 100/S*.

Para a configuração do *kernel* de tempo-real deve aceder-se ao menu *Settings*, sendo possível nesse menu a configuração do tamanho máximo que uma aplicação pode ocupar, bem como o tamanho de memória RAM a ser utilizada no *Target PC*.

Após terem sido efectuadas todas as configurações necessárias, deve aceder-se ao menu *Configuration* (figura 2.41) para se proceder à geração da disquete de arranque. Depois de gerada a disquete de arranque (figura 2.42), esta deve ser introduzida no *driver* de disquetes do *Target PC*, para que quando se inicia o *Target PC*, o *kernel* de tempo-real seja carregado e fique pronto para executar a aplicação de controlo transferida para o *Target PC*.

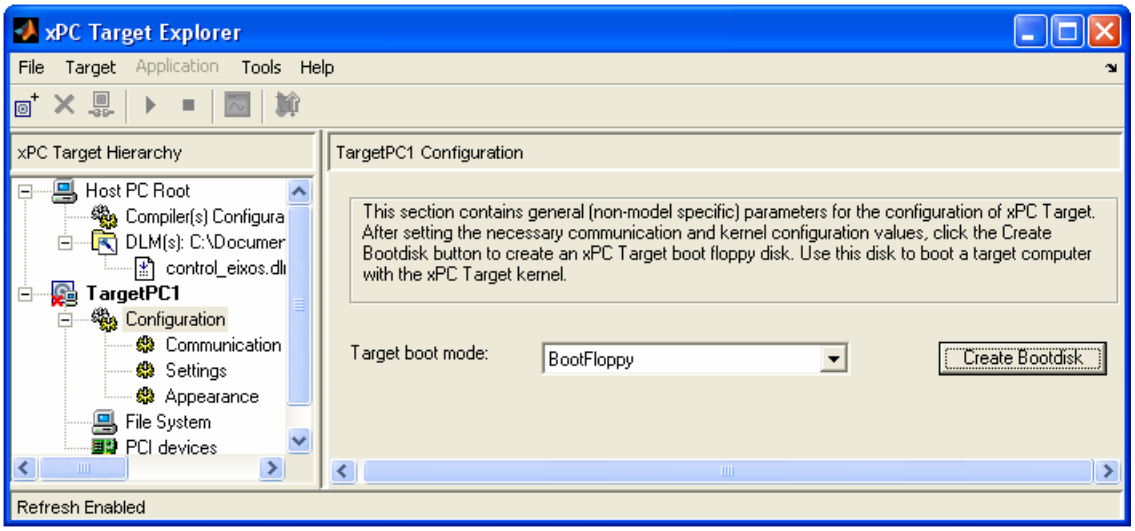


Figura 2. 42 – Geração da disquete de arranque

2.5.3 Arquitectura do *Software*

O sistema de controlo foi construído com base numa arquitectura *Host-Target* (figuras 2.40 e 2.41), explorando as potencialidades da *toolbox xPC Target* do *Matlab*/*Simulink*. O *Host PC* funciona como plataforma de desenvolvimento e como terminal de supervisão e de comando do sistema. No *Host PC* foram desenvolvidas a aplicação de controlo (algoritmos de controlo) e a interface gráfica, sendo para isso necessário a instalação do sistema operativo *Windows XP Professional*, o *Matlab/Simulink* (desenvolvimento da aplicação de controlo) e o *Microsoft Visual Basic* (desenvolvimento da interface gráfica).

O *Target PC* executa em tempo-real a aplicação de controlo desenvolvida, sendo que o código dessa aplicação é descarregado a partir do *Host PC*. No *Target PC* estão instaladas as cartas de interface entre o computador e o sistema físico (robô cartesiano). Estas cartas são do tipo *PCI* e os *device drivers* são disponibilizados pelo *xPC Target*. Através das cartas de I/O é possível monitorizar o estado do sistema físico, bem como enviar ordens de comando para que este realize determinadas acções (figura 2.43).

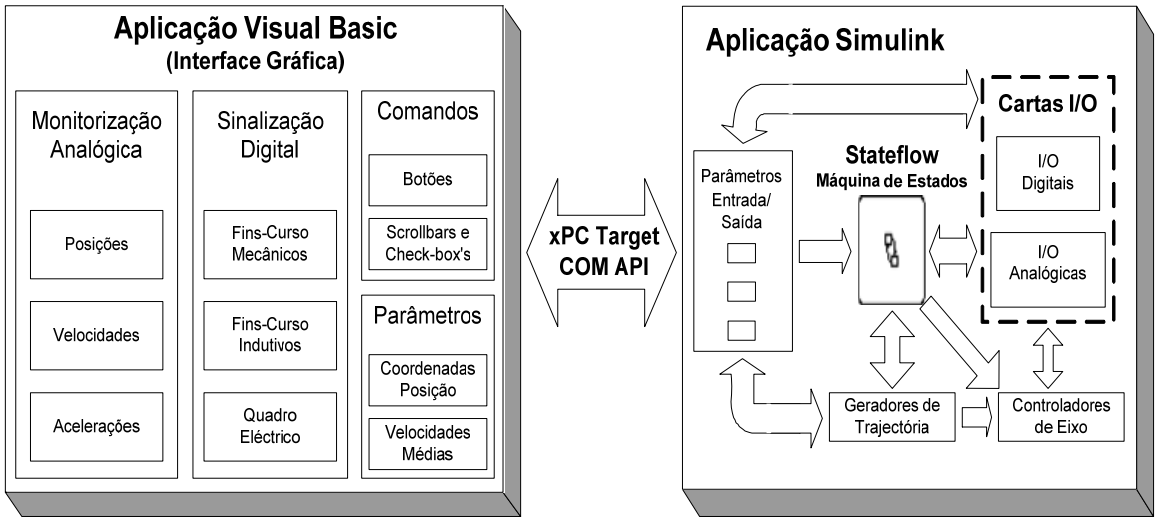


Figura 2. 43 – Arquitectura do *Software*

A máquina de estados, *Stateflow*, é responsável pela recepção dos eventos produzidos pelas entradas digitais (actuação de fins-de-curso), pela mudança de estado dos parâmetros de entrada (accionamento de alguns botões da interface gráfica) ou pelas *flags* (geradas pela aplicação *Simulink*) de fim de trajectória do movimento do robô.

Mediante a recepção destes eventos, um dos estados da máquina de estados ficará activo, o que implica a execução de um subsistema *Simulink* e a alteração do estado das saídas digitais e analógicas.

Na aplicação *Simulink* foram implementados geradores de trajectórias responsáveis por gerar as referências de posição, que são fornecidas aos controladores de eixo. Essas referências são calculadas a partir das coordenadas do ponto final, armazenadas nos parâmetros de entrada, e das coordenadas do ponto inicial armazenadas em variáveis da máquina de estados (figura 2.43).

A trajectória é gerada *online*. Assim, os controladores de eixo recebem a referência de posição e geram a acção de controlo correspondente, que é enviada para as saídas analógicas responsáveis pela aplicação da tensão de controlo aos *drivers* de potência.

A aplicação de controlo foi desenvolvida no *Simulink* (*Matlab*) utilizando blocos predefinidos através de *S-Functions* (bloco do *Simulink* que permite o desenvolvimento de rotinas de código C/C^{++}) e do *Stateflow*, que possibilita o desenvolvimento de uma máquina de estados. A aplicação de controlo é essencialmente composta por (figura 2.43):

- **Parâmetros de Entrada/Saída:**

  - Vectores onde são armazenadas as coordenadas de posição e a velocidade média que define uma trajectória introduzida pelo utilizador na interface gráfica;

  - Vectores onde são armazenadas as coordenadas de posição, velocidade e aceleração instantâneas do robô, que são monitorizadas na interface gráfica;

  - Vectores unitários que armazenam as informações digitais resultantes da operação do utilizador da interface gráfica (ex. a actuação de botões);

- Blocos que armazenam a informação digital dos sensores (fins-de-curso) que compõem o robô cartesiano, bem como a informação de fim de determinadas operações do sistema de controlo (fim de execução de trajectórias).

- **Geradores de Trajectória:**

  - *S-Functions* (código *C*) onde foram desenvolvidos os geradores de trajectória para cada eixo;

  - *S-Functions* onde foi desenvolvido o gerador de trajectórias multi-ponto;

  - Gerador de trajectórias incremental que permite o movimento em modo manual de cada eixo.

- **Controladores de eixo:**

  - Controladores PID com *Anti-Windup Loop* para cada eixo, que permitem o controlo em posição do robô.

- **Máquina de Estados:**

  - Máquina de Estados desenvolvida em *Stateflow* e que possibilita a gestão de eventos decorrentes da operação do sistema robótico.

- ***Drivers* das Cartas I/O:**

  - *Drivers* para as cartas de entrada/saída digitais, que permitem a interface entre a aplicação de controlo e o sistema a controlar;

  - *Drivers* para as cartas de entrada/saída analógicas, que permitem a interface entre a aplicação de controlo e o sistema a controlar.

A interface gráfica permite ao utilizador operar o sistema robótico (robô cartesiano), disponibilizando informação relativa ao comportamento do mesmo. Através da interface gráfica o operador pode definir uma trajectória entre dois ou mais pontos, necessitando para isso de preencher os campos associados à definição de trajectórias.

Na interface gráfica o utilizador pode monitorizar o estado dos fins-de-curso mecânicos e indutivos do robô cartesiano, os sinais digitais presentes no quadro eléctrico do sistema robótico, bem como as variáveis de posição, velocidade e aceleração instantâneas resultantes do movimento do robô cartesiano.

Na troca de informação entre a interface gráfica (*Host PC*) e a aplicação de controlo (*Target PC*) é utilizada a *xPC Target COM API* (figura 2.43). Esta *API* consiste numa *Dinamic Library Link* (*DLL)* que está organizada num conjunto de objectos, classes e métodos. Pode aceder-se às suas funcionalidades através de um ambiente de desenvolvimento gráfico tal como o *Microsoft Visual Basic* ou *C#. NET*, podendo assim criar-se uma interface gráfica que interage com uma aplicação desenvolvida em *Simulink*. Esta *API* possibilita também funcionalidades de procura e capacidade de navegação bidireccional.

A *xPC Target COM API* permite realizar funções, tais como:

- Estabelecer a comunicação entre o *Host PC* e o *Target PC* através de *Ethernet* ou comunicação série;
- Efectuar o *load* do ficheiro de aplicação (*.dlm*) para o *Target PC*;
- Iniciar a aplicação que corre no *Target PC*;
- Monitorizar e modificar o comportamento da aplicação que decorre no *Target PC*;
- Aceder à *file system* do *Target PC*, podendo-se desta forma aceder ou guardar informação no seu disco;
- Parar a aplicação que corre no *Target PC*;
- Efectuar o *unload* do ficheiro de aplicação no *Target PC*;
- Fechar a comunicação com o *Target PC*.

A partir dos métodos disponibilizados pela *API*, os dados inseridos pelo utilizador na interface gráfica são armazenados em blocos *Simulink* (*Constant*), denominados de parâmetros de entrada (figura 2.43). As variáveis e os sinais monitorizados na interface são transferidos da aplicação de controlo para a interface gráfica a partir dos blocos de saída (*Gain*) através de um dos métodos da *API*.

Apresenta-se de seguida um exemplo (figura 2.44) do armazenamento das coordenadas de um ponto de uma trajectória:

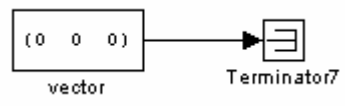


Figura 2. 44 – Exemplo de Armazenamento das Coordenadas de um Ponto

- target_obj.SetParam(target_obj.GetParamIdx("vector", "Value"), [0 1 0])

A função anterior é utilizada na aplicação *Visual Basic* (interface gráfica) e permite escrever um conjunto de valores num bloco do tipo vector da aplicação *Simulink* (sendo obrigatório indicar o nome identificador desse bloco). Neste caso, através da função **target_obj.SetParam** é escrito o conjunto de valores [0 1 0] no bloco designado por "vector" (nome ilustrativo) da aplicação *Simulink*.

O exemplo seguinte ilustra (figura 2.45) a aquisição de um sinal de uma carta de I/O:

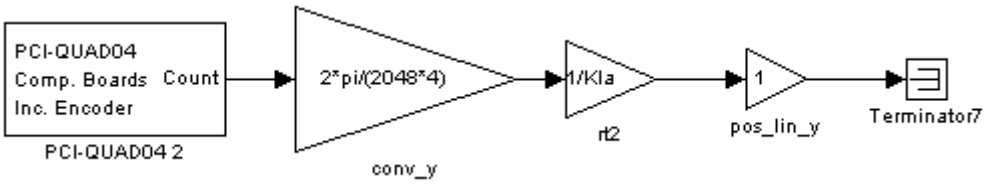


Figura 2. 45 – Exemplo da Aquisição de um Sinal de uma Carta I/O

- target_obj.GetSignal(target_obj.GetSignalIdx("pos_lin_y")

A função anterior permite que através da interface gráfica se possam monitorizar variáveis da aplicação *Simulink*. Neste caso, através da função **target_obj.GetSignal,** utilizada na aplicação *Visual Basic*, é possível adquirir o valor instantâneo da variável

do *Simulink* “pos_lin_y” (que corresponde à posição linear do eixo Y do robô cartesiano).

2.5.4 Aplicação de Controlo – *Simulink*

Nesta subsecção é descrito na generalidade o processo de desenvolvimento da aplicação de controlo. São descritos os principais modos de operação implementados e as técnicas inerentes.

1. Cartas I/O (sinais e variáveis da aplicação de controlo)

As cartas de aquisição de dados implementam a interface entre o meio físico e o controlador do *robô slave*. Sendo que, todos os sinais adquiridos pelas cartas de aquisição e utilizados pelo controlador são obtidos pelo *Simulink* através de *device drivers* disponibilizados, de acordo com o modelo e fabricante, pela *toolbox xPC Target*. Deste modo, para cada sinal pretendido é necessário configurar o bloco (*driver*) disponibilizado pelo *xPC Target* (figura 2.46).

Deste modo, as cartas de aquisição permitem adquirir os sinais digitais dos fins-de-curso mecânicos e indutivos, os sinais digitais presentes no armário eléctrico de controlo do robô *slave*, os valores de posição angulares de cada eixo (através da leitura dos *encoders* dos servomotores) e os valores de aceleração instantânea (através da leitura dos acelerómetros de cada eixo).

Através da carta D/A e I/O digital é possível enviar as acções de controlo, produzidas pelos controladores de eixo do robô *slave*, para os *drivers* de potência. É também possível alterar o estado do sinal de activação (*Run*) dos *drivers* de potência, bem como alterar o estado dos sinais *Overtravel* (indicação de activação de um dos fins-de-curso) e *Brake* (travão do eixo Z).

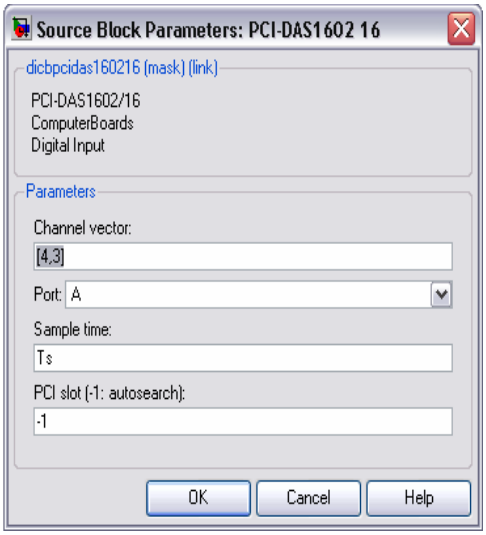


Figura 2. 46 – Configuração de um Bloco *Simulink* das Cartas de Aquisição de Dados

Todos os sinais utilizados pelo controlador tiveram de ser filtrados por *software* através de filtros implementados com recurso a blocos do *Simulink*. Esta necessidade deve-se ao facto de a maioria destes sinais possuírem um ligeiro ruído que, não sendo importante fisicamente para o sistema, provocava situações conflituosas no que diz respeito às reacções aos eventos produzidas pela máquina de estados do controlador. Por exemplo, estando o sistema em movimento, executando uma trajectória definida, se o sinal de um fim-de-curso mecânico apresentar o estado de actuado, quando fisicamente esse mesmo fim-de-curso não se encontra actuado, o sistema entra numa situação de emergência despoletada por um falso evento. Pelos testes efectuados, tal facto, aparentemente, não está associado a problemas ao nível do *hardware*. Deste modo, foi implementado o filtro em *software*, tal como sugere a figura 2.47:

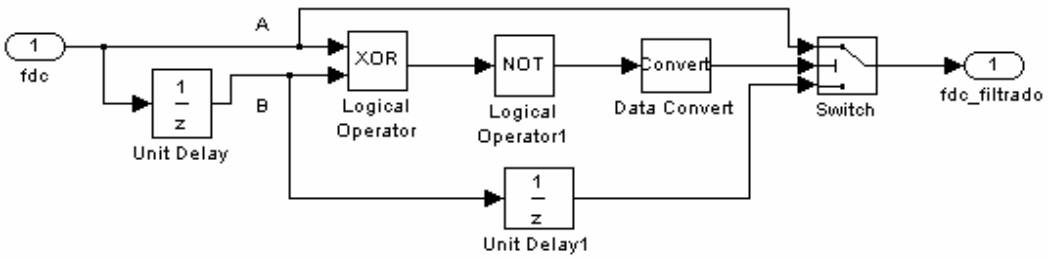


Figura 2. 47 – Filtro dos Sinais de Entrada Digitais

O sinal a ser filtrado é atrasado um período de amostragem, sendo que ambos os sinais são comparados. Se no instante de amostragem anterior o sinal amostrado possuía o mesmo valor que no instante actual, então a saída corresponde ao sinal actual, caso

contrário a saída corresponde ao sinal no instante de dois períodos de amostragem anteriores.

2. Procedimento "Zero do Robô Cartesiano"

Corresponde a um procedimento que consiste em dar ordem de movimento em direcção aos fins-de-curso mecânicos de cada eixo, respectivamente. Quando um dos fins-de-curso é actuado, o sentido de rotação do servomotor respectivo é invertido. Desta maneira o eixo dirige-se em direcção ao fim-de-curso indutivo associado, que quando actuado em conjunto com o sinal correspondente ao *índex* do *encoder* do servomotor efectua o zero do eixo. Após esta operação, a posição em que o robô cartesiano se encontra torna-se o ponto de referência para as restantes operações.

Como foi referido anteriormente para a implementação deste procedimento foi utilizado um controlador proporcional de velocidade. Tendo em conta a descontinuidade no valor dos *encoders* que ocorre na execução deste procedimento quando o fim-de-curso indutivo é actuado, provocando desta forma o *reset* do *encoder*, foi implementado o seguinte algoritmo através de uma *S-Function* para solucionar o problema (figura 2.48).

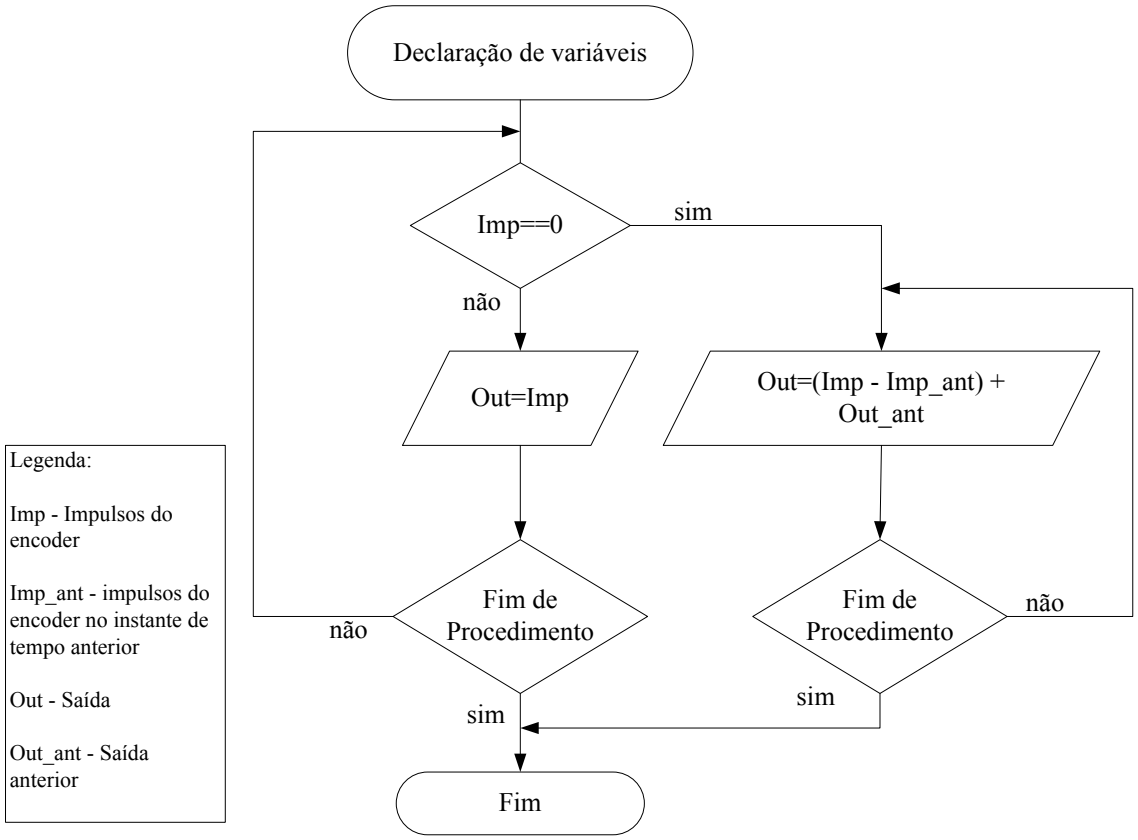


Figura 2. 48 – Algoritmo implementado para suprimir a descontinuidade dos Impulsos do *Encoder*

Através da análise do fluxograma da figura 2.48, pode verificar-se que quando é detectado o *reset* do *encoder* (Imp = = 0), o valor de impulsos utilizado para obter a posição instantânea do robô, depende dos valores dos impulsos do instante anterior. Desta forma, evita-se a ocorrência da descontinuidade do valor da posição instantânea do robô.

Para ser executado este procedimento é fornecida inicialmente ao controlador de velocidade uma referência com perfil sinusoidal, fazendo com que o movimento de arranque seja suave no sentido negativo do deslocamento. O perfil sinusoidal de velocidade é obtido através de uma *S-Function*. Assim que o fim-de-curso mecânico é actuado, o sinal de referência de velocidade fornecido ao controlador é invertido através de um bloco *"Constant"* com um valor de velocidade contrário ao anterior. Quando o fim-de-curso indutivo é actuado é fornecida uma referência de velocidade nula ao controlador.

3. Movimento Manual

O movimento manual foi implementado com o uso de um gerador de trajectórias incremental, sendo que a referência de posição gerada por este é fornecida ao controlador de posição PID.

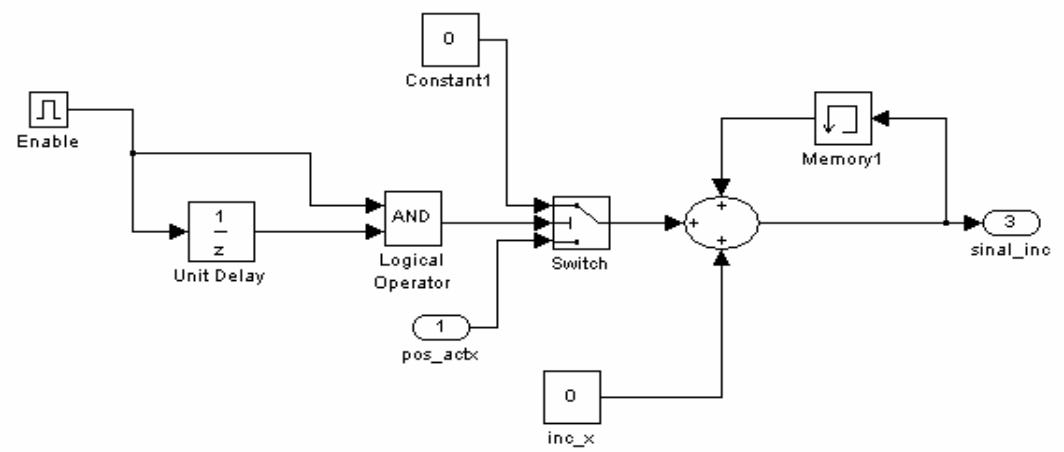


Figura 2. 49 – Gerador de Trajectórias Incremental

Como se pode verificar na figura 2.49, sempre que o modo manual é activado, no primeiro instante de amostragem, o valor da variável de referência de posição *sinal_inc* é igual à soma da posição inicial (*pos_actx*) e do valor do incremento (*inc_x*). Nos restantes instantes de amostragem o valor de saída é igual a soma da posição actual com o incremento de posição. O valor do incremento de posição depende da velocidade pretendida pelo utilizador, se o utilizador pretender uma velocidade maior, consequentemente o valor do incremento *"inc_x"* também é aumentado. Desta maneira chega-se à relação entre os dois valores. Sabendo que o período de amostragem é de 1E-3 s (1 ms), para um valor de velocidade de 20 mm/s (por exemplo), obtém-se:

$$inc = \frac{v}{T} \Leftrightarrow inc = \frac{20\ mm/s}{1*10^{-3}\ s} = 20*10^{-3}\ mm \qquad 2.58$$

É utilizado um bloco *Memory* para evitar um *loop algébrico* que ocorre na soma do valor *sinal_inc* com o valor do incremento.

4. Trajectória Eixo-a-Eixo

Este procedimento é executado quando o utilizador pretende efectuar uma trajectória linear para um determinado ponto, com um movimento singular de cada eixo. Para isso, foram desenvolvidos três geradores de trajectória, um para cada eixo. Embora o algoritmo desenvolvido seja igual para todos, cada um recebe a coordenada correspondente ao seu eixo e é responsável pela geração da trajectória para o mesmo (figura 2.50).

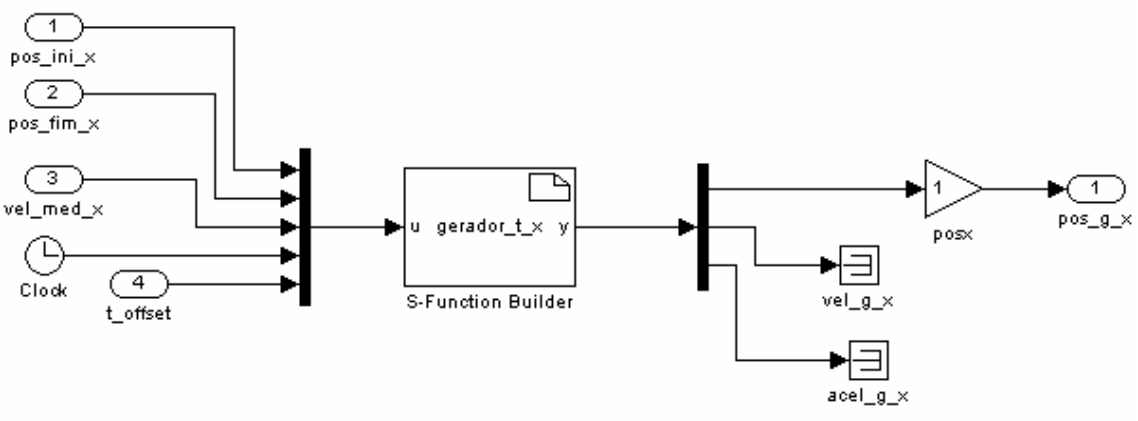


Figura 2. 50 – Gerador de Trajectórias Linear Eixo-a-Eixo

A descrição e implementação do algoritmo de geração de trajectórias já foi abordada numa das subsecções anteriores. De seguida será descrito o procedimento implementado para disponibilizar todos os dados necessários à geração da trajectória.

Para a geração de uma trajectória é necessário obter a posição de partida (inicial), a posição final, a velocidade média e o tempo de execução da trajectória. A posição inicial é armazenada numa variável da máquina de estados no momento em que se inicia a execução da trajectória. Os valores de posição final e velocidade média são fornecidos pelo utilizador através da interface gráfica, sendo armazenados num vector (bloco *"Constant"* do *Simulink*). O valor do tempo, necessário para a geração da trajectória, é obtido subtraindo o valor do tempo corrente da execução da aplicação e uma constante (armazenada numa variável da máquina de estados) correspondente ao instante de tempo no início da execução da trajectória.

Depois de obtidas as variáveis anteriores, o algoritmo de geração de trajectórias (implementado numa *S-Function*, figura 2.50) é executado, sendo geradas as referências de posição, velocidade e aceleração que são transmitidas ao controlador de eixo correspondente.

Para a detecção do fim de trajectória de cada eixo foi desenvolvido um algoritmo de teste de velocidade e do erro de posição (figura 2.51). Este algoritmo consiste em verificar se o valor absoluto da velocidade se mantém abaixo de um determinado patamar (neste caso 1E-6 mm/s) um número de vezes estipulado (neste caso 1000 amostras), em conjugação com a verificação se o valor absoluto do erro de posição se mantêm abaixo de 0.01 mm, durante o mesmo período de tempo. Quando estas condições são verificadas, é activada a *flag* correspondente ao fim de trajectória.

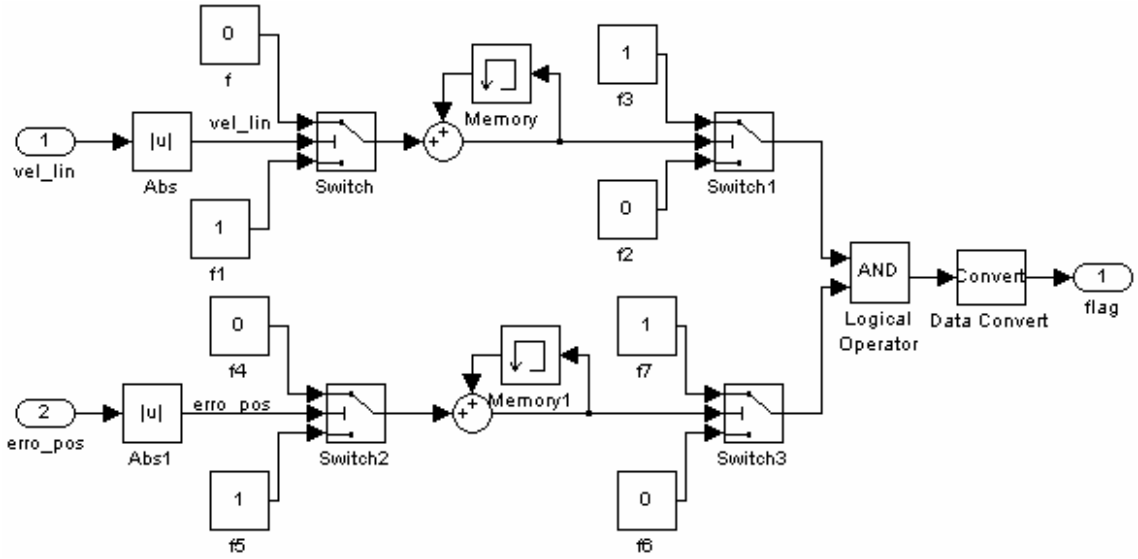


Figura 2. 51 – Algoritmo de Detecção de Fim de Trajectória

5. Trajectória Linear 3 Eixos

Este tipo de trajectória resulta na execução de um segmento de recta no espaço tridimensional, uma vez que é efectuada com os três eixos em movimento. Para a execução deste tipo de movimento foi necessário implementar um gerador de trajectórias capaz de gerar *online* a trajectória para cada eixo. Este gerador de trajectórias (figura 2.52) é uma conjugação dos três geradores de trajectórias utilizados na implementação do procedimento descrito no item anterior (figura 2.50). A diferença

entre os dois procedimentos reside no facto de que para se realizar um segmento de recta no espaço tridimensional é necessário decompor e reajustar os valores de velocidade de cada eixo.

Para a realização deste tipo de movimento é necessário calcular as componentes de velocidade de cada eixo a partir das coordenadas do ponto final e da velocidade média da trajectória. O cálculo dessas componentes é efectuado através de um algoritmo que, além de calcular as componentes de velocidade para cada eixo, verifica se os valores de velocidade excedem o máximo valor permitido. Caso uma das componentes de velocidade ultrapasse o valor máximo permitido, é necessário recalcular novamente todas as componentes, de modo a que a trajectória seja executada uniformemente.

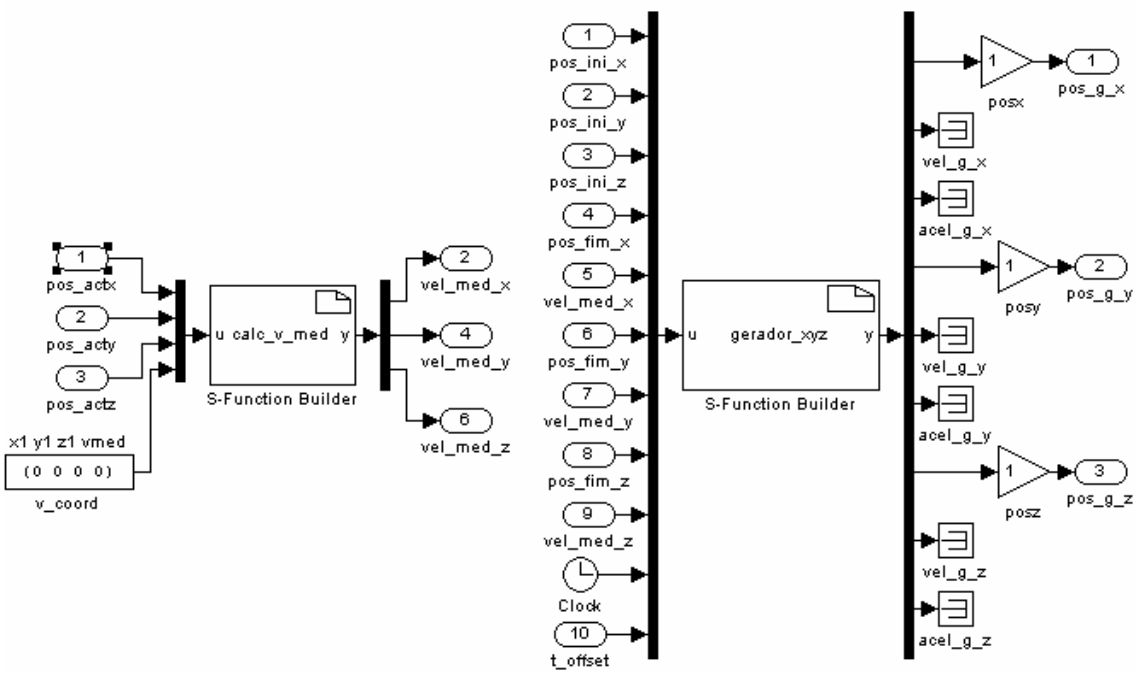


Figura 2. 52 – Gerador de Trajectória Linear 3 Eixos

O fluxograma da figura 2.53 descreve a rotina de implementação de uma trajectória entre dois pontos. O primeiro passo é a leitura das coordenadas do novo ponto, posteriormente é executado o ciclo de implementação da trajectória. Durante o ciclo de execução da trajectória, são geradas as referências de posição, velocidade e aceleração, são obtidas as coordenadas de posição instantâneas do robô e geradas as acções de controlo de cada controlador de eixo (PID). No fim da trajectória os resultados são guardados em ficheiro para posterior análise.

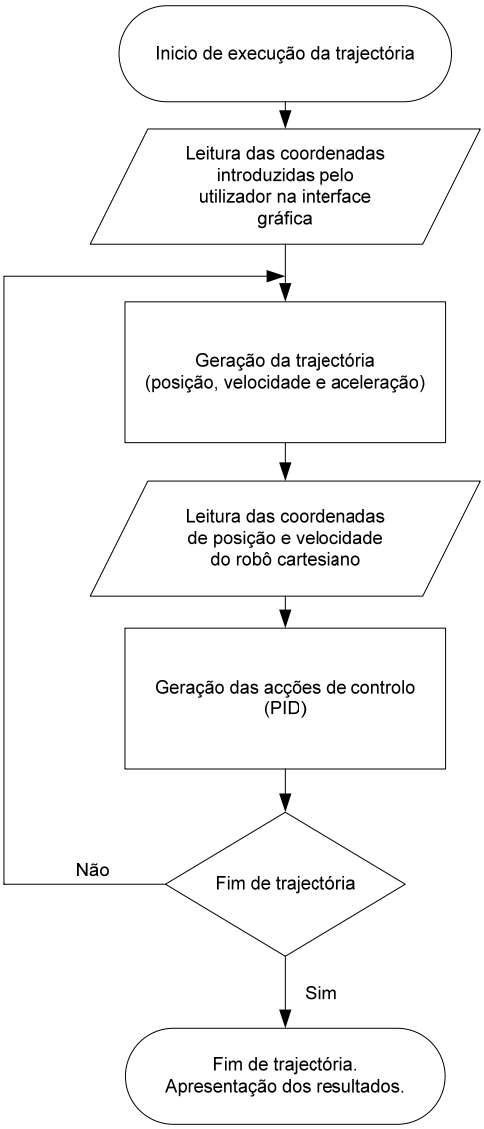


Figura 2. 53 – Rotina de Execução da Trajectória Entre dois Pontos

6. Trajectória Multi-ponto

Este tipo de trajectória implica a utilização do algoritmo de geração de trajectórias "Movimento entre um número arbitrário de pontos" que foi descrito numa das subsecções anteriores.

Para a execução deste tipo de movimento foi necessária a construção de uma matriz de dimensões consideráveis que armazena todos os pontos que definem uma trajectória. A tarefa de preenchimento dessa matriz é realizada por um algoritmo (implementado numa *S-Function*, figura 2.54) que também é responsável pelo cálculo das diferentes

componentes de velocidade. Desta forma, a cada ponto da trajectória foi atribuído um número identificador, ou seja, o primeiro ponto tem como identificador o número 1, o segundo ponto o 2, até chegar ao número de pontos da trajectória. Assim, as coordenadas de cada número são armazenadas por ordem na matriz. O número de pontos também é guardado na última posição da matriz, para que se saiba quantas posições são ocupadas pelos pontos na matriz.

Quando é dada a ordem de execução da trajectória, a matriz que contém todos os pontos da trajectória é processada e é utilizada na geração das referências de posição, velocidade e aceleração fornecidas aos controladores de eixo.

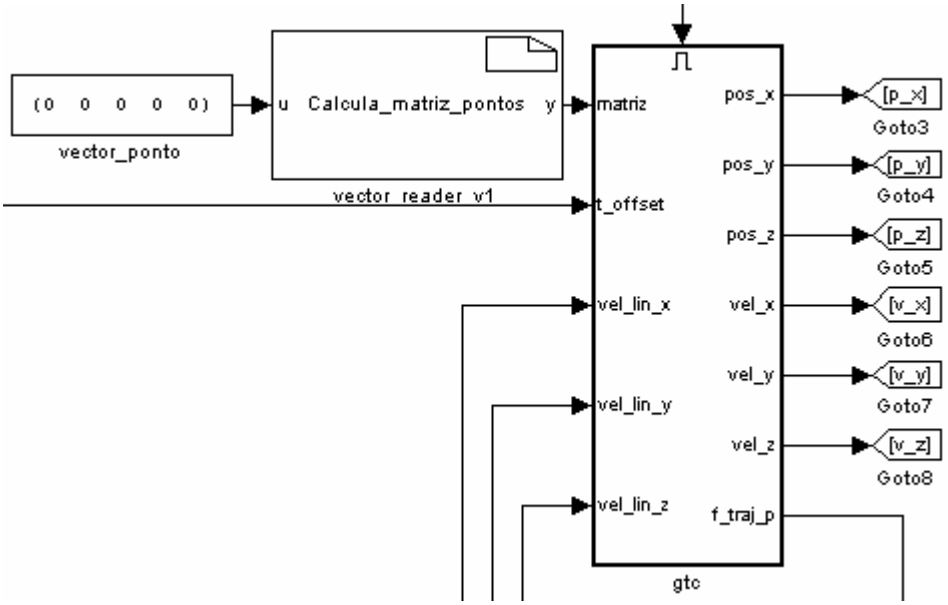


Figura 2. 54 – Gerador de Trajectória Multi-ponto

De seguida é apresentado o fluxograma (figura 2.55) que descreve a rotina de execução de uma trajectória multi-ponto. Como se pode verificar o primeiro passo é a introdução (na interface gráfica) de todos os pontos que constituem a trajectória multi-ponto. Os pontos introduzidos são lidos e armazenados em ficheiro. Depois de definidos todos os pontos da trajectória, o controlador executa o posicionamento do robô cartesiano no primeiro ponto dessa trajectória. Após o posicionamento do robô no primeiro ponto da trajectória, o controlador executa o ciclo de implementação da trajectória. Deste modo, são geradas as referências de posição, velocidade e aceleração, são lidas as coordenadas de posição do robô e com estes dados são geradas as acções de controlo correspondentes

através de cada controlador de eixo (PID). No final da trajectória multi-ponto o robô é posicionado no ponto de partida.

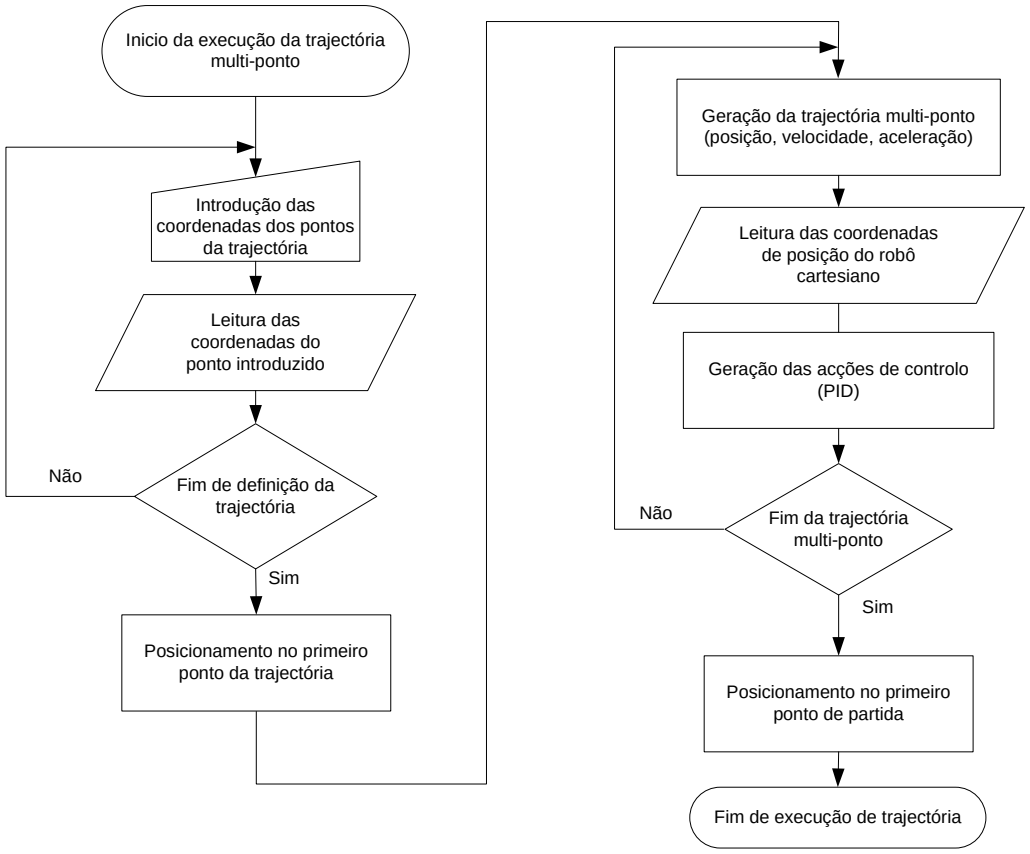


Figura 2. 55 – Rotina de Execução de Trajectória Multi-ponto

2.5.5 Máquina de Estados – *Stateflow*

O *Stateflow* é uma ferramenta do *Matlab*/*Simulink* muito poderosa no contexto dos sistemas de eventos discretos. Assim, esta ferramenta foi essencial na implementação da aplicação de controlo, simplificando a difícil tarefa de realizar uma máquina de estados recorrendo apenas a blocos *Simulink*, o que é possível mas desaconselhável. Desta forma, a aplicação de controlo desenvolvida inclui uma máquina de estados que contempla todos os modos de funcionamento (estados) do robô, bem como as transições possíveis entre eles.

A filosofia da máquina de estados corresponde a associar a cada estado do robô um subsistema *Simulink*, que é responsável pela implementação da acção de comando apropriada. A cada transição de estado está associada uma receptividade (evento ou condição). Existindo dois tipos de receptividades que podem causar transições: receptividade local, que pode ser gerada por *software*; receptividade exterior à máquina de estados, que corresponde a entradas provenientes de blocos *Simulink*. Estas receptividades encontram-se subdivididas em três grupos: receptividades geradas pelo *hardware* (ex. fins-de-curso); receptividades geradas pelo utilizador durante a operação da interface gráfica; receptividades geradas pelo término de determinadas acções (*flags* do modelo *Simulink*, como por exemplo as associadas à conclusão de uma trajectória).

A máquina de estados implementada além de possuir como entradas os eventos do sistema, possui também variáveis de entrada que são utilizadas para armazenar dados importantes do sistema, tais como o tempo corrente de execução e as posições lineares do sistema robótico. As saídas da máquina de estados efectuam a habilitação (*enable*) dos subsistemas *Simulink* e também disponibilizam informações como o tempo corrente ou as posições lineares do robô em cada instante (figura 2.56).

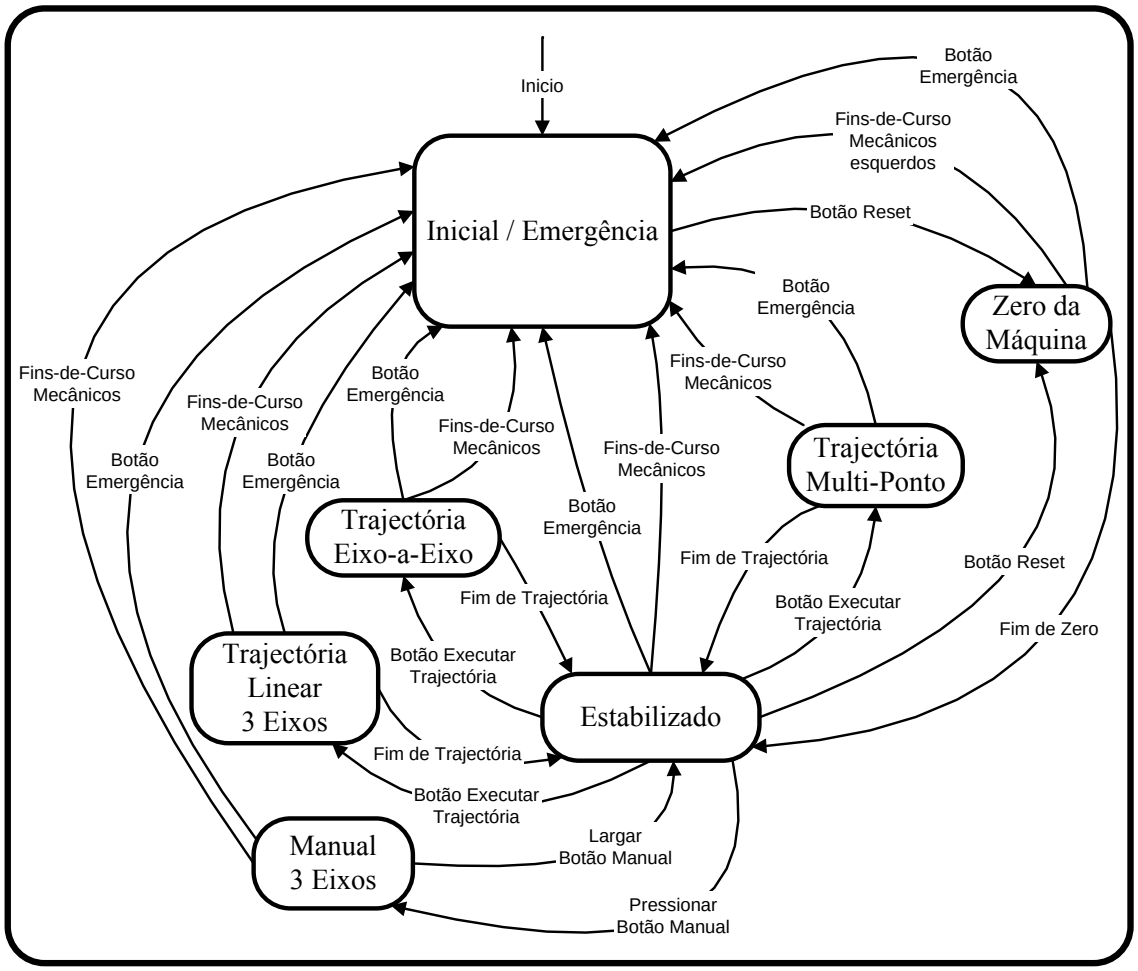


Figura 2. 56 – Diagrama da Máquina de Estado do Controlador

Conforme é possível observar no diagrama de estado (figura 2.56) existe um estado inicial ("*Inicial/Emergência*") que é o ponto de partida. Quando o sistema se encontra no estado inicial são activados os sinais digitais *Run* e *Brake*, sendo fornecidas referências nulas aos *drivers* de potência. Deste modo o controlador do sistema fica preparado para qualquer ordem de movimento.

Tal como se pode verificar no diagrama da figura 2.56, o sistema só sai do estado inicial se for dada ordem para se efectuar o procedimento designado *Zero da Máquina*. Este procedimento corresponde ao posicionamento do robô num ponto de referência no espaço cartesiano, sendo que todos os movimentos efectuados após este procedimento são referenciados a esse mesmo ponto. Por isso, a única transição possível do estado inicial é para o estado "*Zero da Máquina*", esta transição ocorre quando o utilizador acciona este procedimento na interface gráfica (botão *Reset*).

Durante a execução do procedimento *Zero da Máquina*, o utilizador pode em caso de emergência parar a execução do mesmo. Para isso, o utilizador deve actuar o botão *Emergency* na interface gráfica. Sempre que o botão *Emergency* é actuado na máquina de estados ocorre uma transição para o estado *Inicial/Emergência*. Este estado tem a mesma função do estado inicial, com a diferença de que os sinais *Run* e *Brake* não se encontram activados.

Ainda durante a execução do procedimento *Zero da Máquina*, se um dos fins-de-curso mecânicos "esquerdos" for actuado, o procedimento é abortado e na máquina de estados ocorre uma transição para o estado *Inicial/Emergência*.

Depois de executado o procedimento *Zero da Máquina*, a máquina de estados transita para o estado *Estabilizado*. Este estado faz activar o subsistema dos controladores de posição PID, atribuindo referências de posição fixas aos mesmos. Referências essas que correspondem às coordenadas de posição do robô cartesiano no momento em que ocorre a transição para o estado *Estabilizado*.

Encontrando-se activo o estado *Estabilizado*, o sistema só evolui para outro estado por acção do utilizador, quer pela operação da interface gráfica ou pela actuação de um dos componentes do sistema físico (por exemplo a actuação de uma botoneira de emergência). A partir do estado *Estabilizado* o sistema pode transitar para um dos estados de execução de trajectórias, sendo que essa transição só ocorre quando o operador actuar na interface gráfica o evento (botão) associado à execução do tipo de trajectória pretendido.

Existem quatro estados associados à execução de trajectórias: *Manual 3 Eixos*; *Trajectória Eixo-a-Eixo*; *Trajectória Linear 3 Eixos* e *Trajectória Multi-ponto*. Em todos esses estados, que implicam a movimentação do robô cartesiano, estão contempladas as transições para o estado *Inicial/Emergência*, se ocorrerem os seguintes eventos: actuação de um dos fins-de-curso mecânicos; actuação do botão de emergência da interface gráfica ou actuação de uma das botoneiras de emergência que compõem o sistema robótico. No fim de cada execução de trajectória ocorre uma transição na máquina de estados para o estado *Estabilizado*.

A identificação do fim de trajectória foi implementada através de um conjunto de blocos *Simulink* (figura 2.51), que testam a condição de conjunção das velocidades e dos erros de posição de cada eixo permanecerem abaixo de um determinado valor (10E-6 mm/s para a velocidade e 0.01 mm para o erro) durante 1 s, ao fim desse tempo é activada uma *flag*. Sempre que é detectado o evento associado à *flag* de fim de trajectória ocorre uma transição na máquina de estados para o estado *Estabilizado*.

De seguida são descritos os procedimentos inerentes aos *super-estados* representados no diagrama de estados da figura 2.56, são eles: *Zero da Máquina*, *Manual 3 Eixos* e *Trajectória Eixo-a-Eixo*.

1. *Zero da Máquina*

Este *super-estado* está associado ao procedimento *Zero da Máquina* que foi descrito na subsecção anterior. Como se pode observar no diagrama da figura 2.57, este *super-estado* é composto essencialmente por 3 *sub-estados*, sendo que cada um deles implementa o procedimento para cada eixo.

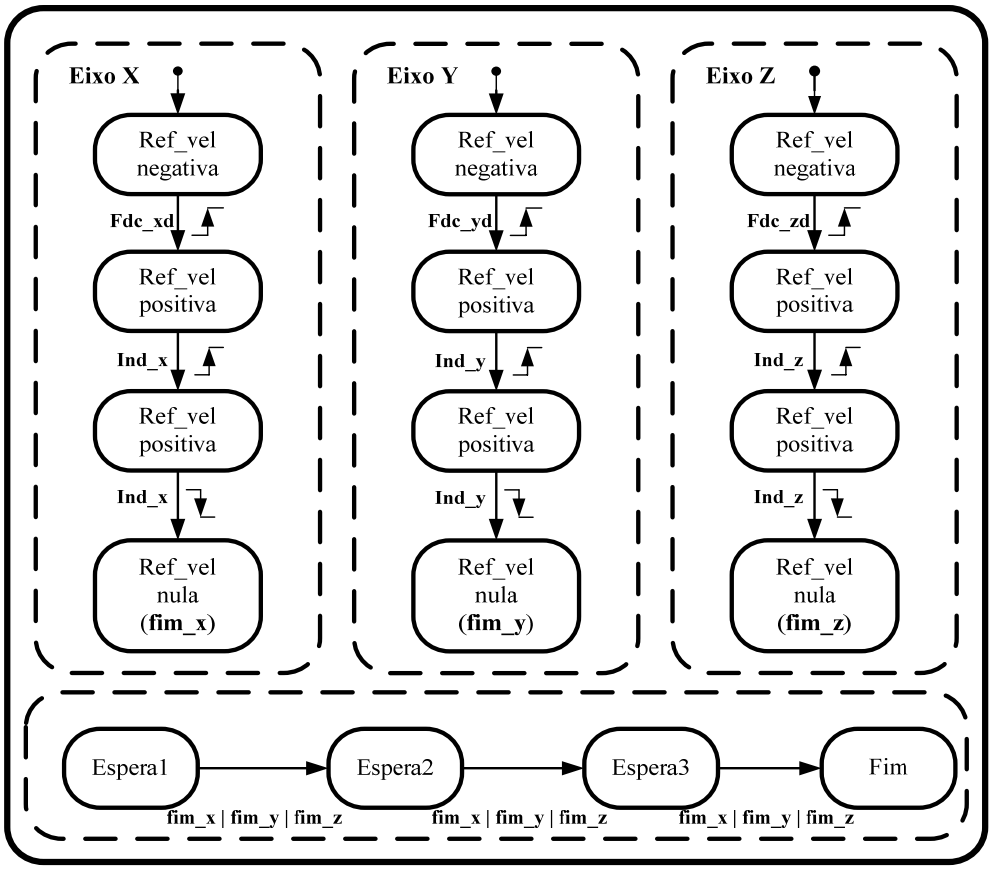


Figura 2. 57 – Diagrama do *Super-Estado Zero da Máquina*

Quando este super-estado é activado, os sub-estados que o compõem são também activados. Desta forma, a evolução dos 3 sub-estados é efectuada em paralelo, o que significa que o procedimento *Zero da Máquina* é efectuado com todos os eixos em movimento.

A cada um dos estados que compõem o super estado *Zero da Máquina* está associado um subsistema *Simulink*, estes subsistemas fornecem referências de velocidade aos controladores de velocidade de cada eixo, estas referências alteram-se mediante o estado que se encontra activo.

O procedimento efectuado para cada um dos eixos obedece a uma sequência de operações. A primeira operação a ser efectuada corresponde a atribuir uma referência constante negativa aos controladores de velocidade, o que implica que cada eixo se mova em direcção ao seu fim-de-curso direito. Quando ocorre o evento respectivo à mudança de estado (flanco ascendente) de um fim-de-curso, sucede-se então uma transição para o estado seguinte, que implica a atribuição de uma referência positiva ao

controlador de velocidade, ou seja, o movimento do eixo é invertido. Após ser invertido o movimento do eixo, no momento em que ocorre uma mudança de estado (flanco descendente) do fim-de-curso indutivo é atribuída uma referência nula ao controlador de velocidade, ou seja, fica imobilizado nessa posição.

Quando todos os eixos se encontram parados depois de efectuarem este procedimento, é necessário efectuar o *reset* do sinal de *Overtravel*. Este sinal encontra-se activo devido ao accionamento dos fins-de-curso mecânicos durante a execução do procedimento *Zero da Máquina*.

Depois de todos os eixos estarem na posição de referência e de se ter efectuado o *reset* ao sinal de *Overtravel*, ocorre então uma transição do estado *Zero da Máquina* para o estado *Estabilizado*. Quando essa transição ocorre dá-se uma mudança do tipo de controlo exercido sobre o sistema robótico, passando-se de um controlo em velocidade para um controlo em posição.

2. *Manual 3 Eixos*

Quando o sistema se encontra no estado *Estabilizado* e, por acção do utilizador na interface gráfica (botão *manual*), é activado o movimento manual de um dos eixos, ocorre então uma transição para o estado associado ao movimento manual desse eixo. A velocidade com que o movimento é efectuado deve ser definida pelo utilizador na interface gráfica.

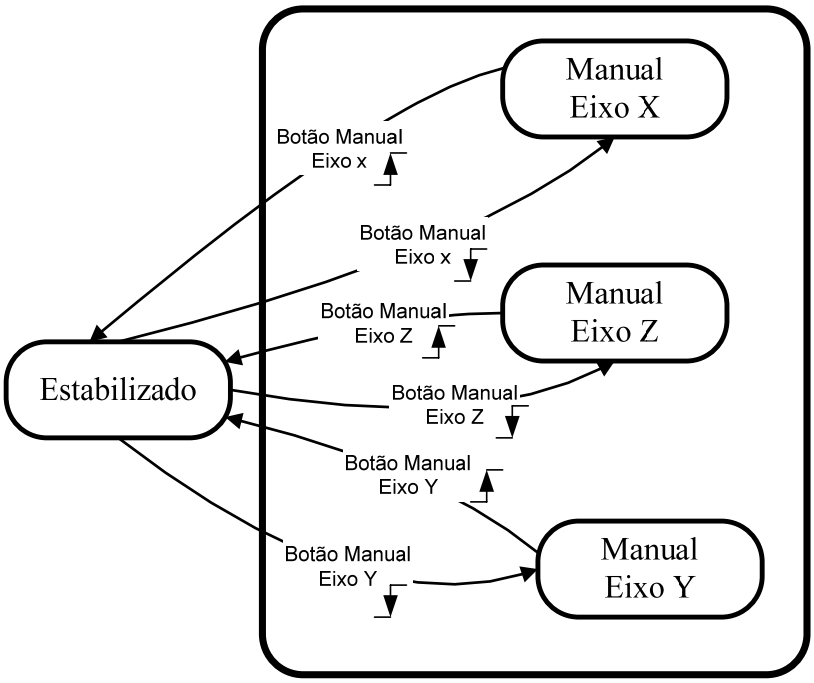


Figura 2. 58 – Diagrama do *Super-Estado Manual 3 Eixos*

Conforme se pode verificar no diagrama da figura 2.58, o movimento de um eixo é executado enquanto o utilizador pressionar na interface gráfica o botão associado. Quando o utilizador deixar de pressionar esse botão, o movimento do eixo pára, uma vez que ocorre um evento (flanco descendente) que origina a transição de novo para o estado *Estabilizado*.

3. *Trajectória Eixo-a-Eixo*

O utilizador ao definir, na interface gráfica, uma trajectória entre dois pontos dispõe de dois tipos de escolha de execução da mesma. Um desses tipos corresponde a *trajectória eixo-a-eixo* que consiste no movimento singular de cada eixo, ou seja, a trajectória é executada num espaço tridimensional com o movimento de um eixo de cada vez.

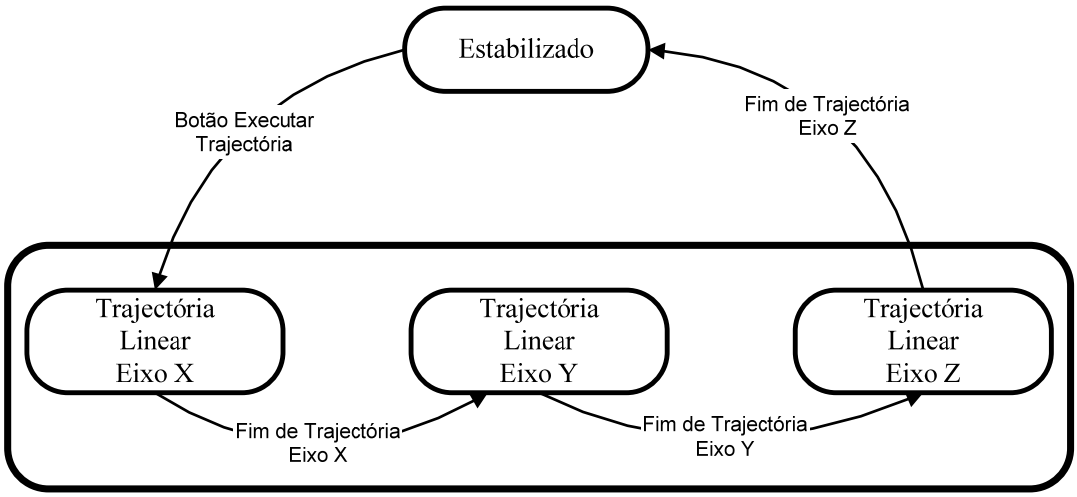


Figura 2. 59 – Diagrama do *Super-Estado Trajectória Eixo-a-Eixo*

Como se pode verificar no diagrama da figura 2.59, quando o utilizador dá a ordem de execução de trajectória (botão da interface gráfica), ocorre uma transição para o estado responsável pela execução do movimento do eixo X. Quando o movimento do eixo X termina, ocorre uma transição para o estado responsável pela execução do movimento do Y e posteriormente seguir-se-á o eixo Z. Quando o movimento dos três eixos está completo, ou seja, a trajectória foi terminada, então o sistema regressa ao estado *Estabilizado*.

2.6 Conclusões

Neste capítulo foi efectuada a descrição do sistema robótico a controlar, sendo também apresentada a estratégia de controlo implementada. Na descrição do sistema robótico foram apresentados os seus principais componentes e descritas as suas funções, bem como apresentada a modelação do sistema e a respectiva síntese dos controladores de eixo. A descrição da estratégia de controlo incluiu a apresentação da arquitectura de *hardware* e *software* implementadas para realizar as diversas tarefas de controlo.

O uso da *toolbox xPC Target* revelou-se essencial para a execução em tempo-real dos algoritmos desenvolvidos. Salientando o facto de a *xPC Target COM API* permitir o desenvolvimento de aplicações gráficas (*web*, *.NET*, etc.) que possibilitam, em tempo-real, a interacção com a aplicação de controlo, este facto revela-se importante quando se pretende implementar o controlo de sistemas remotos.

O uso do *Stateflow* na implementação da máquina de estados do sistema de controlo permitiu um desenvolvimento mais fácil da aplicação de controlo e subsequentemente uma melhor interface com a aplicação gráfica de comando. De salientar o facto de a máquina de estados ser um elemento essencial da aplicação de controlo, uma vez que é através dela que são produzidas as diversas reacções aos eventos gerados pela operação do sistema.

Neste capítulo foram ainda apresentados e comentados alguns resultados experimentais, que ilustram o desempenho dos controladores de eixo do sistema robótico. Da análise dos resultados, verificou-se que para maiores amplitudes de movimento, os controladores de eixo apresentavam bons desempenhos tanto no regime transitório como no regime permanente. Para movimentos de baixa amplitude, os controladores de eixo apresentam um ligeiro atraso na resposta em relação às referências de posição e velocidade, sendo que, o erro em regime permanente é quase desprezável. Deste modo, um dos principais objectivos da implementação dos controladores de eixo foi atingido, ou seja, a precisão no posicionamento.

Por comparação dos diversos resultados experimentais apresentados, pode concluir-se que o controlador demonstra melhores desempenhos para trajectórias de grande amplitude de movimentos, para elevadas velocidades de execução de trajectórias.

Capítulo 3

Interface Gráfica de Comando do Robô Cartesiano

3.1 Introdução

A interface gráfica de comando foi desenvolvida em *Microsoft Visual Basic 6.0*, uma vez que a *toolbox xPC Target* dispõe de uma *API* (*xPC Target COM API*) que efectua a troca de informação entre a aplicação de controlo desenvolvida em *Simulink* e a interface gráfica. No desenvolvimento desta interface gráfica pretendeu-se implementar as principais funcionalidades de um robô industrial, tal como se poderá verificar na descrição efectuada neste capítulo.

A interface gráfica possibilita a um utilizador a operação do sistema robótico. Permite que o utilizador efectue operações de definição e execução de trajectórias entre dois ou mais pontos, com a possibilidade de escolha de diferentes tipos de movimento. Sendo que, após a conclusão das trajectórias é possível a visualização de gráficos que possuem informação relativa aos movimentos efectuados. A interface gráfica permite também o movimento manual de cada um dos eixos do robô.

Durante a operação do sistema robótico, o utilizador pode monitorizar através da interface gráfica o comportamento do mesmo. O utilizador pode monitorizar sinais

digitais e analógicos adquiridos pelas cartas I/O, que efectuam a interface entre o sistema de comando (interface gráfica) e o sistema físico (sistema robótico).

O painel principal da interface gráfica (figura 3.1) permite ao utilizador explorar as potencialidades do sistema robótico em modo de aprendizagem. No menu *Trajectory* o utilizador tem a possibilidade de definir trajectórias mais complexas (multi-ponto), que são executadas de uma forma autónoma pelo sistema de controlo.

Neste capítulo serão descritos os componentes que compõem a interface gráfica, bem como as funcionalidades da mesma. São também descritas as funcionalidades do sistema robótico inerentes às operações efectuadas na interface gráfica.

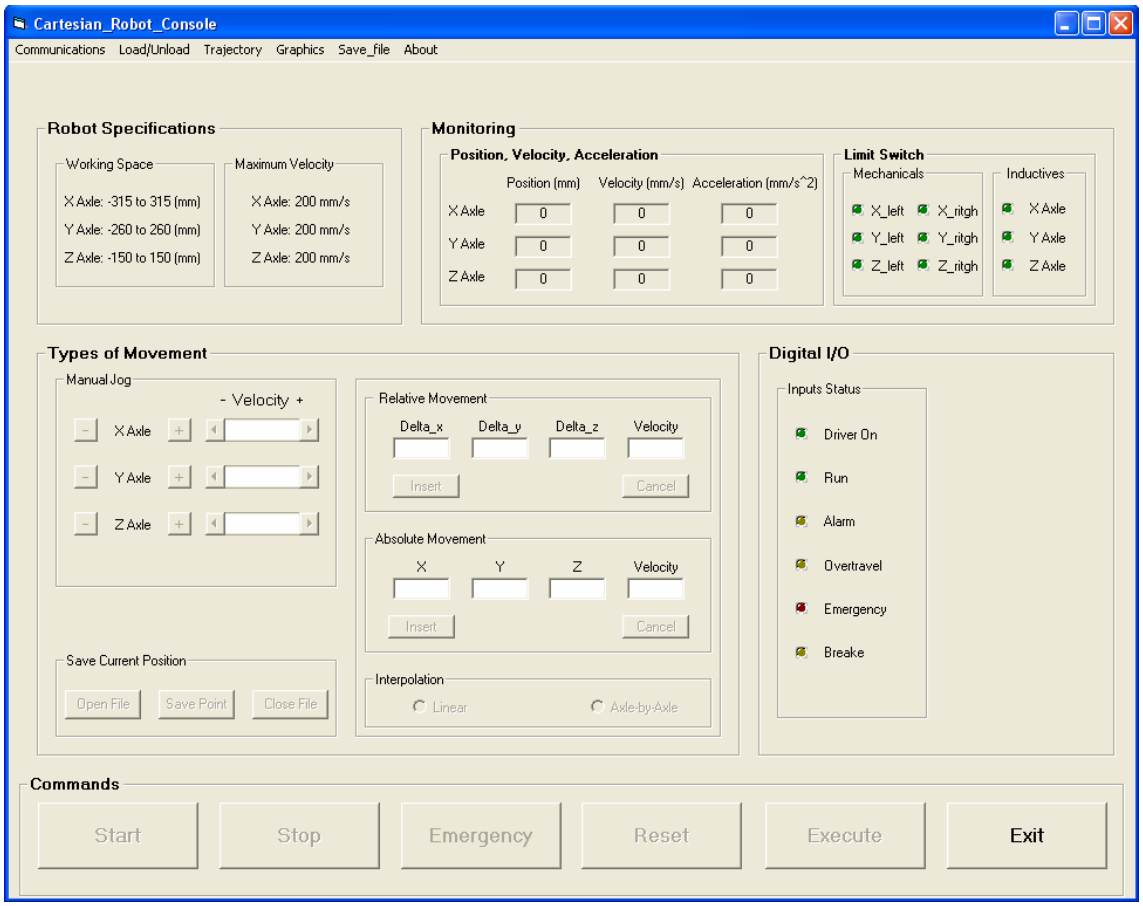


Figura 3. 1 – Interface Gráfica de Comando

3.2 Painel Principal da Interface Gráfica

O painel principal da interface gráfica (figura 3.1) é composto por diferentes sub-painéis, com funções diversas, permitindo ao utilizador explorar as potencialidades do sistema robótico em modo de aprendizagem.

3.2.1 Painel *Robot Specifications*

Este painel fornece ao utilizador informações importantes sobre o sistema robótico, permitindo-lhe um nível de operação mais seguro e consciente. Desta forma, são fornecidos ao utilizador dados sobre a área de trabalho do robô, ou seja, os cursos máximos de cada eixo, bem como a indicação da velocidade máxima que cada eixo pode atingir (figura 3.2).

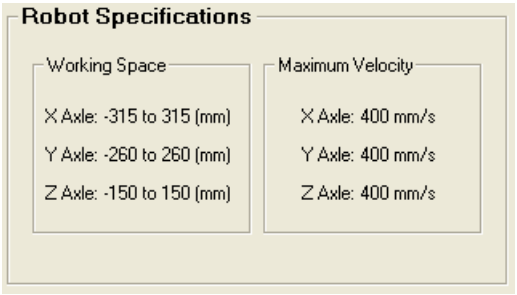


Figura 3. 2 – Especificações do Robô Cartesiano

3.2.2 Painel *Monitoring*

Este painel permite ao utilizador monitorizar a cada instante o estado operacional do sistema robótico. O utilizador pode visualizar a informação digital e analógica, fornecida através das cartas de aquisição, referente ao estado dos diversos componentes que compõem o sistema robótico.

Neste painel o operador pode visualizar os valores instantâneos da posição, velocidade e aceleração de cada eixo do robô cartesiano. Pode também visualizar sinais digitais referentes ao estado dos fins-de-curso mecânicos e indutivos (figura 3.3):

- **Fins-de-curso mecânicos**
    - Eixo X direito
    - Eixo X esquerdo
    - Eixo Y direito
    - Eixo Y esquerdo
    - Eixo Z direito
    - Eixo Z esquerdo

- **Fins-de-curso indutivos**
    - Eixo X
    - Eixo Y
    - Eixo Z

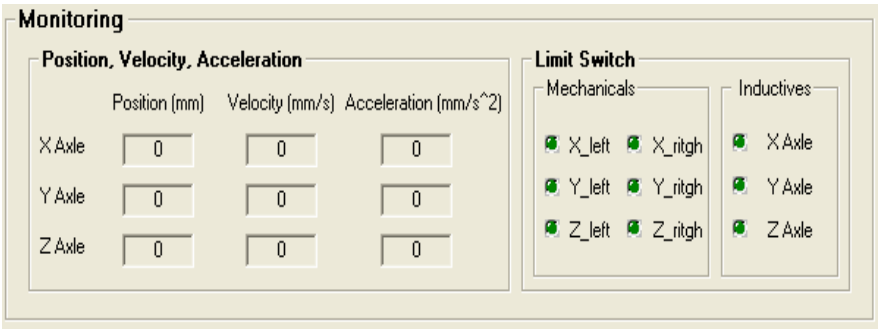


Figura 3. 3 – Monitorização de Variáveis e Sinais do Sistema Robótico

Outro painel onde é possível monitorizar sinais digitais relevantes sobre o estado do sistema é o *Digital I/O*. Este painel efectua a monitorização dos sinais presentes no quadro eléctrico. Através dele o utilizador pode detectar uma condição de alarme ou de emergência (sinais *Alarm*, *Emergency* e *Overtravel*). O utilizador pode também verificar em que condição se encontra a operacionalidade do sistema através dos sinais *Driver On*, *Run* e *Brake* (figura 3.4).

Figura 3. 4 – Painel que replica os Sinalizadores do Armário Eléctrico do Robô Cartesiano

3.2.3 Painel *Types of Movement*

Este painel foi concebido com o intuito de disponibilizar ao utilizador ferramentas que lhe possibilitassem movimentar o robô desde a posição corrente até uma posição desejada.

O utilizador pode movimentar o robô de três formas diferentes, executando movimentos manuais, relativos ou absolutos.

1. Movimento manual

Para efectuar este tipo de movimento foi construído um sub-painel em forma de consola, que dispõe de um conjunto de botões e *scrollbars*. Esses botões permitem ao utilizador efectuar um movimento manual individual para cada eixo, devendo para isso pressionar o botão correspondente ao eixo que pretende movimentar e o sentido do deslocamento. A velocidade do movimento é definida no *scrollbar* correspondente ao eixo a deslocar. Quando o utilizador pretender parar o movimento basta deixar de pressionar o botão. Sempre que o utilizador carrega num dos botões, para executar o movimento de um dos eixos, é gerado um evento correspondente na aplicação de controlo, o que origina uma mudança na máquina de estados (*Stateflow*) para o estado associado ao movimento do eixo seleccionado. Tal mudança de estado implica que a aplicação de controlo execute o gerador de trajectórias manual do eixo seleccionado (figura 3.5).

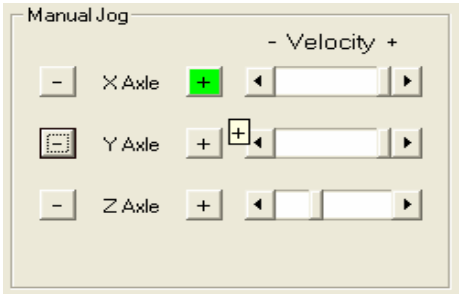


Figura 3. 5 – Movimento Manual

2. Movimento relativo

Para executar este tipo de movimento o utilizador deverá introduzir os incrementos de posição que serão somados às coordenadas actuais do robô, originado o ponto de destino do movimento do robô. Além dos incrementos de posição, o utilizador terá também de indicar a velocidade média com que pretende que o movimento seja efectuado. Estes parâmetros deverão ser introduzidos nas respectivas caixas de texto da interface gráfica (figura 3.6), são verificados e validados, caso estejam correctos.

3. Movimento absoluto

Para a execução deste tipo de movimento, o utilizador tem de indicar as coordenadas absolutas do ponto de destino do movimento do robô, bem como a velocidade média com que esse movimento é efectuado. Tal como acontece no movimento relativo, os parâmetros inseridos pelo utilizador são verificados antes da execução da trajectória (figura 3.6).

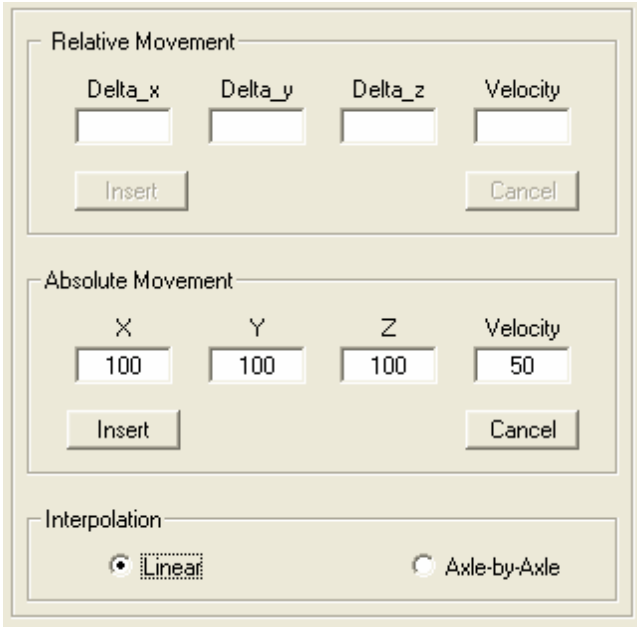


Figura 3. 6 – Definição de Movimento entre dois Pontos

Os parâmetros introduzidos pelo utilizador, referentes à definição da trajectória a ser executada, são guardados num vector da aplicação *Simulink* para serem usados no processamento da trajectória.

Em ambos os movimentos, relativo e absoluto, depois de indicados os parâmetros necessários para a definição da trajectória, o utilizador deverá indicar qual o tipo de execução de movimento (figura 3.6) que pretende efectuar. Existem dois tipos de execução de movimento à escolha, são eles o movimento linear e o movimento eixo-a-eixo (figura 3.6). Na realização do movimento linear é executado o gerador de *trajectória linear 3 eixos* (figura 2.52). Deste modo, o movimento efectuado pelo robô corresponderá a uma trajectória linear uniforme no espaço tridimensional. No caso do movimento eixo-a-eixo (*Axle-by-Axle*) é executado o gerador de *trajectória eixo-a-eixo* (figuras 2.60 e 2.51), o que corresponde à execução de um gerador de trajectórias associado a cada eixo, resultando desta forma na realização de um movimento de um eixo de cada vez.

Após a definição da trajectória, o utilizador deverá carregar no botão *Execute* (painel *Commands*) para que o movimento do robô seja executado. Durante a movimentação do robô, todas as funcionalidades da interface gráfica encontram-se bloqueadas, por razões de segurança, com excepção do botão *Emergency* (painel *Commands*, figura 3.7) que permite paragem do movimento em caso de emergência.

3.2.4 Painel *Commands*

Este painel (figura 3.7) disponibiliza alguns botões com comandos específicos dentro da interface gráfica. A actuação de um destes botões provoca efeitos na interface gráfica, nomeadamente no desbloquear ou bloquear de certas funcionalidades, como também produz determinadas acções do controlador do sistema. Em seguida serão descritas as propriedades de cada um destes botões de comando, bem como as acções resultantes da sua actuação.

Figura 3. 7 – Botões de Comando da Interface Gráfica

1. Botão *Start*

A activação deste botão implica o início da aplicação, a partir desse momento o sistema está pronto a ser controlado, sendo que o primeiro procedimento a ser efectuado é o *Zero da Máquina*, por esse motivo o único botão activo no início da aplicação é o *Reset*.

Deste modo, ao ser accionado este botão ocorre uma transição no flanco descendente no estado do bloco *Simulink* associado a esse botão. Esse bloco é uma das entradas da máquina de estados construída no *Stateflow*, sendo o responsável pela geração do evento que activa o estado *Inicial/Emergência* da máquina de estados (figura 2.56). Este estado por sua vez activa o subsistema do *Simulink* responsável por colocar as referências de binário dos eixos a zero, excepto o eixo Z cuja referência do binário é a suficiente para equilibrar o efeito da força gravítica. Nesse subsistema os sinais de *Run* e o sinal do travão do eixo Z (*Brake*) são activados, deste modo, os *drivers* de potência ficam prontos para receber e processar as referências de binário enviadas pelo controlador, bem como o travão do eixo do Z fica destravado.

2. Botão *Stop*

O accionamento deste botão provoca a paragem da aplicação, o que implica a suspensão do sistema de controlo. Neste caso, o sistema fica inoperante a qualquer ordem de comando executada pelo utilizador na interface. Para retomar a execução da aplicação é necessário accionar de novo o botão *Start*.

Como já foi referido, este botão suspende a execução do controlador, não gerando qualquer evento que implique uma mudança de estado na máquina de estados do controlador do sistema. Desta forma, nenhuma acção é executada uma vez que a aplicação de controlo se encontra suspensa.

3. Botão *Emergency*

Este botão tem como única função parar a execução da aplicação de uma forma segura. Caso o utilizador entenda que existe necessidade, por questões de segurança, de parar a execução da aplicação, o accionamento deste botão permite que o sistema entre num estado de segurança (*Inicial/Emergência* – figura 2.56). Quando é accionado este botão, é sugerido ao utilizador que resolva os problemas que levaram ao accionamento do mesmo, devendo regressar novamente à execução da aplicação quando verificar que existem condições para prosseguir. Depois de uma paragem de emergência, por razões de segurança o único botão que está activo é o *Reset*, o que obriga o utilizador a executar o procedimento *Zero da Máquina*.

Ao ser accionado o botão *Emergency* ocorre uma transição no flanco descendente numa das variáveis da aplicação de controlo *Simulink*, o que provoca a geração de um evento na máquina de estados (*Stateflow*) do controlador. Esse evento origina uma transição para o estado *Inicial/Emergência* (figura 2.56) da máquina de estados. Como já foi referido anteriormente, quando o estado *Inicial/Emergência* fica activo, são atribuídas referências de binário nulas aos controladores de eixo, mas neste caso os sinais *Run* e *Brake* são desactivados (os *drivers* de potência ficam inoperacionais e o eixo Z fica travado).

4. Botão *Reset*

A acção inerente ao accionamento deste botão é a execução do procedimento *Zero da Máquina*. Este procedimento é executado sempre que se inicia a aplicação, após o utilizador accionar o botão *Emergency*, sempre que um dos fins-de-curso mecânicos seja actuado ou se uma das botoneiras de emergência for premida, entrando o sistema numa situação de emergência. Assim, sempre que o sistema recupere de uma situação de emergência, o utilizador é obrigado a executar novamente o procedimento *Zero da Máquina*.

Ao actuar-se o botão *Reset* ocorre uma transição no flanco descendente no estado numa das variáveis da aplicação de controlo *Simulink*, o que origina a ocorrência de um evento na máquina de estados do controlador. Tal evento causa uma transição do estado *Inicial/Emergência* para o *Zero da Máquina* (figura 2.56). O *Zero da Máquina* é um dos *super-estados* da máquina de estados *Stateflow*, quando este estado fica activo, a aplicação de controlo executa o procedimento que implementa o zero dos *encoders* de cada eixo do robô.

Assim que o procedimento *Zero da Máquina* esteja concluído ocorre uma transição, na máquina de estados, do estado *Zero da Máquina* para o *Estabilizado*, sendo activada a *flag* que indica o fim deste procedimento. A activação da referida *flag* implica mudanças na interface gráfica, nomeadamente o bloqueio e o desbloqueio de certas funcionalidades da mesma, tais como as funcionalidades inerentes à execução de movimentos (painel *Types of Movement* – figura 3.1).

5. Botão *Execute*

Sempre que o utilizador pretenda executar qualquer movimento entre dois pontos, com excepção do movimento manual, deve accionar o botão *Execute*.

Quando este botão é accionado, ocorre um evento na aplicação de controlo *Simulink*, o que provoca uma transição para um dos estados responsáveis pela execução da trajectória. O evento gerado, bem como o estado que fica activo depende da escolha do utilizador referente ao tipo de trajectória a executar. Assim, se o utilizador tiver optado

pela interpolação linear entre dois pontos, ocorre um evento que produz a transição para o estado *Trajectória linear 3 eixos* da máquina de estados, sendo executado um movimento linear uniforme no espaço tridimensional. Caso o utilizador opte por uma interpolação eixo-a-eixo, ocorre um evento que causa a transição para o estado *Trajectória Eixo-a-Eixo*, o que implica a execução do movimento seja efectuada por um eixo de cada vez.

Durante a execução de qualquer movimento as funcionalidades da interface gráfica encontram-se bloqueadas, com excepção do botão *Emergency*.

No final de cada movimento é activada uma *flag* que origina um evento na máquina de estados. Esse evento causa uma transição do estado que se encontra activo para o estado *Estabilizado*, o que implica que o sistema fique em regime estacionário e em controlo de posição no ponto de destino da trajectória. Depois de concluída a trajectória, todas as funcionalidades da interface gráfica ficam operacionais novamente.

6. Botão *Exit*

Este botão tem como principais funcionalidades o parar da execução da aplicação de controlo e o fechar da sessão da interface gráfica. Quando este botão é accionado todas as tarefas em execução pelo sistema de controlo são abortadas e as rotinas de monitorização são inibidas.

3.2.5 Painel *Save Current Point*

Durante a operação da interface gráfica, o utilizador pode executar inúmeros tipos de movimentos para diferentes coordenadas do espaço de trabalho do robô cartesiano. Os pontos visitados pelo robô durante uma sessão podem ser guardados em ficheiro. Para o efeito foi criado o painel *Save Current Point*, que permite ao utilizador gravar os pontos das várias trajectórias executadas ao longo de uma sessão. Esses pontos são gravados em ficheiro no formato *.txt*, o que permite ao utilizador a sua posterior utilização na execução de trajectórias multi-ponto (figura 3.8).

Sempre que o utilizador pretenda guardar um conjunto de pontos, durante a execução das trajectórias (em modo de ensino), deverá em primeiro lugar criar um ficheiro através do botão *Open File*, indicando o nome do respectivo ficheiro. Depois de criado o ficheiro, o utilizador deverá utilizar o botão *Save Point* para gravar os pontos pretendidos. Quando tiver todos os pontos desejados gravados em ficheiro, o utilizador terá de o fechar para poder utilizá-lo, para tal deverá accionar o botão *Close File*.

Como já foi referido, o utilizador pode utilizar o ficheiro que contém os pontos gravados na execução de uma trajectória multi-ponto. Deste modo, o utilizador terá de aceder ao menu *Trajectory* e escolher a opção *Load From File* (figura 3.12), nesse menu poderá carregar o dito ficheiro e executar a trajectória correspondente. Essa trajectória será executada automaticamente de acordo com as opções indicadas pelo utilizador.

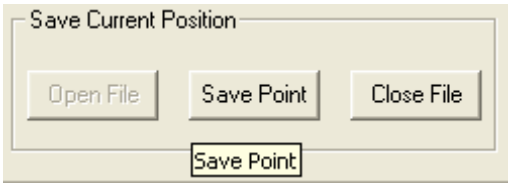


Figura 3. 8 – Gravação de um Ponto da Trajectória

3.3 Menus da Interface Gráfica

A interface gráfica dispõe de menus (figura 3.9) que permitem ao utilizador o acesso a determinadas funcionalidades que não estão disponíveis no painel principal. Assim, de seguida serão descritas as funcionalidades de cada menu da interface gráfica:

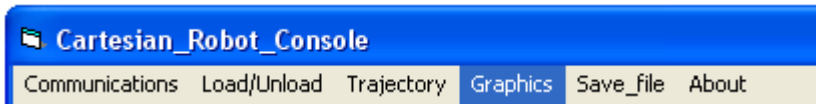


Figura 3. 9 – Menus da Interface Gráfica

1. Menu *Communications*

Este menu possibilita ao utilizador a escolha do protocolo de comunicação entre a interface gráfica (*Host PC*) e a aplicação de controlo (*Target PC*). Existem dois tipos de protocolo de comunicação suportados pelo *software xPC Target*, são eles a comunicação via porta série e *TCP/IP*. A comunicação via porta série embora esteja implementada, não se encontra activa, uma vez que a ligação física entre o *Host PC* e o *Target PC* é do tipo *TCP/IP*. No caso de se optar por uma ligação por porta série a opção desactivada terá de ser desbloqueada no *software* de desenvolvimento.

Assim, quando o utilizador acede ao menu *Communications* pode estabelecer uma ligação *TCP/IP* com o *Target PC* (onde corre a aplicação de controlo), sendo necessário a introdução do endereço de *IP* e do *Port Number* nos campos correspondentes (figura 3.10).

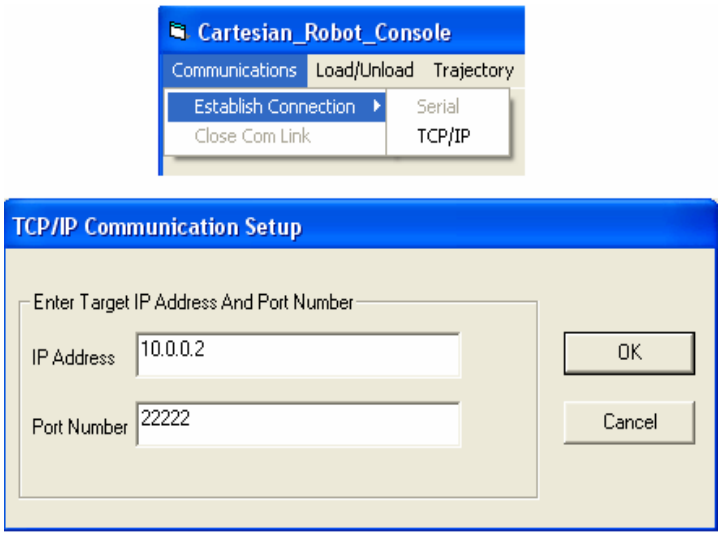


Figura 3. 10 – Menu *Communications* e Configuração de Comunicação *TCP/IP*

2. Menu *Load/Unload*

Este menu permite ao utilizador carregar o ficheiro que foi gerado após a compilação da aplicação de controlo no *Simulink* (*Real Time Workshop*). Esse ficheiro depois de ser carregado é transferido para o *Target PC* onde é executado pelo *kernel* de tempo-real do *xPC Target*.

Se o utilizador pretender executar um ficheiro de uma outra aplicação de controlo, não necessita de sair da aplicação corrente. Para isso, o utilizador deve aceder ao menu *Load/Unload* e efectuar o *unload* do presente ficheiro, sendo que de seguida deverá efectuar o *load* do novo ficheiro (figura 3.11).

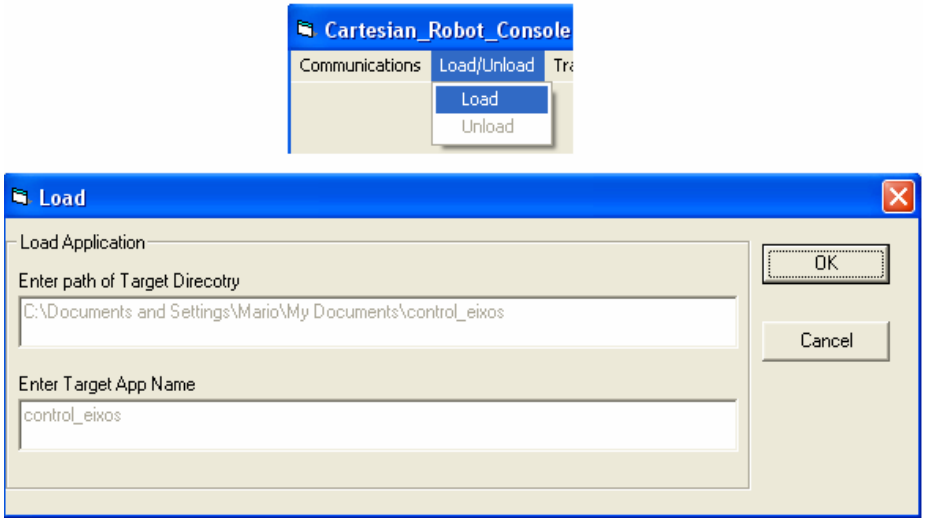


Figura 3. 11 – Menu *Load/Unload* e Janela de *Load* do Ficheiro de Controlo

3. Menu *Trajectory*

Este menu permite aceder a dois sub-menus: *Insert Points* e *Load From File*. Nesses dois sub-menus o utilizador pode definir e executar uma trajectória multi-ponto definida pelo próprio (figura 3.12).

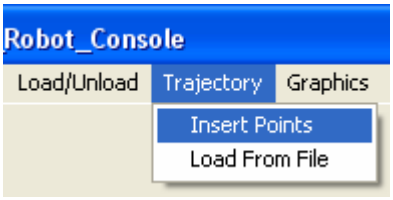


Figura 3. 12 – Menu *Trajectory*

Embora ambos os sub-menus implementem a definição e a execução de uma trajectória multi-ponto, existe uma diferença entre eles. Essa diferença reside no facto de no sub-menu *Insert Points*, o utilizador inserir as coordenadas dos diversos pontos que compõem a trajectória através de caixas de texto. Enquanto que no sub-menu *Load From File*, o utilizador define o ficheiro que contém a trajectória definida por diversos pontos (figura 3.13).

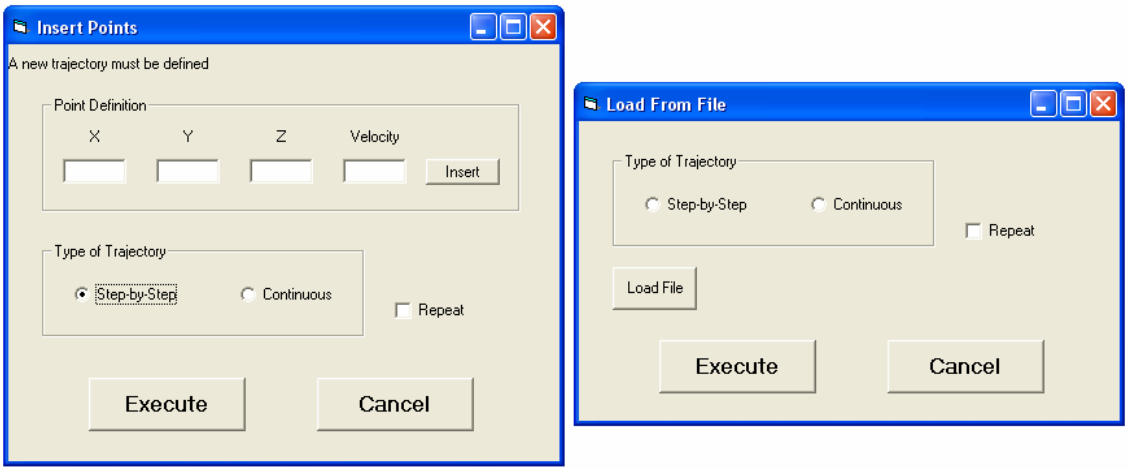


Figura 3. 13 – Sub-Menus *Insert Points* e *Load From File*

No sub-menu *Insert Points* o utilizador deve inserir as coordenadas de posição em cada campo respectivo, bem como o valor de velocidade média que pretende para a execução da trajectória que leva o robô ao ponto definido. Depois de preenchidos os campos associados à definição de um dos pontos da trajectória, o utilizador deve confirmar esse

ponto carregando no botão *Insert*, isso implica que as coordenadas do ponto inserido sejam verificadas. Se estiverem correctamente definidas são gravadas num ficheiro.

No caso do sub-menu *Load From File*, o utilizador tem a possibilidade de escolher um ficheiro que contenha a definição de uma trajectória multi-ponto, sendo que a escolha do ficheiro é efectuada através do botão *Load File*. Esse ficheiro pode ser criado e preenchido manualmente pelo utilizador (num editor de texto), ou pode ser gerado a partir do painel *Save Current Point* da interface gráfica.

Em ambos os sub-menus, tanto os dados contidos no ficheiro que armazena os pontos para uma trajectória, como os pontos inseridos pelo utilizador nas caixas de texto são verificados, em caso de detecção de qualquer incorrecção a trajectória é rejeitada (ex. pontos fora do espaço de trabalho).

A execução de uma trajectória multi-ponto é sempre efectuada cumprindo o seguinte procedimento:

- A primeira tarefa corresponde ao posicionamento automático do robô no primeiro ponto da trajectória definida. O robô é deslocado da posição em que se encontra para a posição correspondente ao primeiro ponto da trajectória definida, segundo um movimento linear entre os dois pontos;
- Depois é executada toda a trajectória, passando o robô pelos pontos definidos na trajectória;
- Por último, quando a trajectória é finalizada o robô regressa automaticamente ao primeiro ponto da trajectória, segundo um movimento linear entre os dois pontos.

Em ambos os sub-menus o operador pode escolher o modo de execução da trajectória, sendo que existem duas opções:

- ***Step-by-Step*:**

  Este modo de execução de trajectória implica que se efectue sempre um movimento linear entre dois pontos, ou seja, a trajectória é executada troço a troço. Desta forma, numa trajectória multi-ponto executada passo a passo implica que o movimento do robô pare em cada ponto da trajectória.

  Este modo de execução de trajectórias foi implementado através do gerador de trajectórias *linear de 3 eixos*, o que implica que entre cada troço que compõe a trajectória, o robô descreva um movimento linear uniforme no espaço tridimensional. Sempre que o movimento do robô complete um dos troços da trajectória, automaticamente é dada a ordem para a execução do troço seguinte. Quando é atingido o último ponto da trajectória, o robô recebe uma ordem para regressar ao ponto de origem.

- ***Continuous*:**

  Este modo de execução de trajectória é implementado através de um gerador de trajectórias multi-ponto, ou seja, a trajectória é executada em modo contínuo (sem paragens nos pontos da trajectória), isso implica que o movimento do robô passe na vizinhança dos pontos da trajectória.

  A escolha deste tipo de movimento implica que os pontos da trajectória sejam guardados numa matriz da aplicação de controlo. Quando é dada a ordem de execução do movimento, o gerador de trajectórias multi-ponto acede a essa matriz para obter os pontos necessários a execução da trajectória.

A trajectória multi-ponto pode ser executada em modo singular ou em modo repetitivo. Existe uma opção (*Repeat*) em cada sub-menu, que permite ao utilizador seleccionar o modo repetitivo de execução da trajectória. A escolha do modo repetitivo implica que a trajectória seja executada em modo cíclico até que seja dada ordem de paragem, quando isso sucede o ciclo de movimento é concluído, sendo que no final o robô regressa ao ponto de origem.

4. Menu *Graphics*

Este menu permite ao utilizador aceder à visualização de alguns gráficos que possuem informações relativas a um movimento executado pelo robô. Após a execução de um determinado movimento, o utilizador tem acesso aos gráficos de posição, velocidade, aceleração e acção de controlo fornecida pelo controlador ao sistema, para cada um dos eixos, em função do tempo. Além dos gráficos referidos anteriormente, o utilizador pode ainda observar um gráfico de posição *3D*, caso a trajectória executada seja efectuada com o movimento simultâneo dos 3 eixos (trajectória linear).

Existem tipos de movimento que não permitem a visualização dos referidos gráficos, são eles o movimento manual do robô e a execução de trajectórias multi-ponto em modo repetitivo.

Os gráficos apresentados são produzidos através de um executável desenvolvido em *Matlab*, o que permite a apresentação de resultados de uma forma clara e intuitiva, além de disponibilizar um conjunto de ferramentas que possibilitam a fácil manipulação dos gráficos em questão (ex. rotação, ampliação).

A informação contida nestes gráficos é obtida a partir de *Scopes* disponibilizados pela *toolbox xPC Target*, que possibilita guardar dados no disco do *Target PC* sobre a forma de um ficheiro binário. Posteriormente esses dados são enviados para o *Host PC* por intermédio de funções (*FileSystem*) disponibilizadas pela *API* do *xPC Target* para a aplicação *Visual Basic*. Os dados são então convertidos e armazenados no *Host PC* na forma de ficheiro no formato *.txt*. Esse ficheiro é utilizado pelo executável *Matlab* para gerar os gráficos que possuem informação do movimento do robô.

5. Menu *Save_file*

Depois da visualização dos gráficos mencionados no item anterior, o utilizador pode guardar a informação de uma trajectória em ficheiros de *backup*. Estes ficheiros são criados automaticamente, com um nome correspondente à data da geração dos mesmos, podendo ser utilizados para posteriores comparações de resultados obtidos em diferentes trajectórias realizadas (figura 3.14).

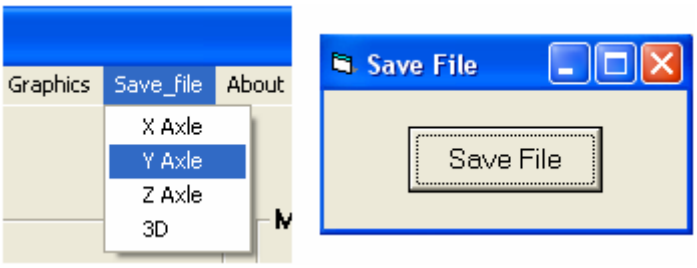


Figura 3. 14 – Menu *Save_file*

3.4 Conclusões

Pretendeu-se com o desenvolvimento da interface gráfica de comando, dotar o sistema robótico de algumas das funcionalidades encontradas num robô industrial. Nesse sentido, a interface gráfica permite ao utilizador efectuar diferentes tipos de movimentos com o robô cartesiano, tais como: movimento manual, movimento relativo e absoluto entre dois pontos, bem como o movimento linear multi-ponto (entre vários pontos).

A execução dos diferentes movimentos pode ser efectuada de diferentes formas. No caso da trajectória entre dois pontos pode efectuar-se um movimento linear com a movimentação de todos os eixos do robô, ou optar-se por um movimento singular de cada eixo. No que diz respeito à trajectória multi-ponto, o utilizador pode escolher entre executar o movimento ponto a ponto ou em modo contínuo.

Através da interface gráfica o utilizador tem a possibilidade de monitorizar o estado operacional do sistema robótico, bem como visualizar e armazenar os dados referentes às diversas trajectórias efectuadas pelo robô cartesiano.

Neste capítulo foram descritas na generalidade as funcionalidades da interface gráfica e o modo de operação da mesma, efectuando uma associação entre as acções da interface gráfica e as consequentes reacções do sistema de controlo.

Capítulo 4

Descrição do Sistema Robótico de Telemanipulação

4.1 Introdução

Um sistema robótico de telemanipulação é composto por dois subsistemas: *master* e *slave*. Tal sistema permite a um operador, agindo localmente sobre o robô *master*, interagir com ambientes remotos, por intermédio do robô *slave*. Tarefas que podem beneficiar de um tal sistema são, por exemplo, a manipulação de ambientes no espaço, ambientes hostis e a cirurgia remota (telemedicina).

Neste capítulo serão descritos os principais requisitos de funcionamento do sistema *master/slave*. É também apresentado e caracterizado o dispositivo *Phantom Haptic Device*, que desempenha as funções de *master* no sistema de telemanipulação descrito neste trabalho. Por último é descrita a arquitectura do sistema robótico *master/slave*.

4.2 Requisitos de Funcionamento do Sistema Robótico *Master/Slave*

Tipicamente, a arquitectura de controlo de um sistema robótico do tipo *master/slave* (telemanipulação) pode apresentar várias configurações. Deste modo, as arquitecturas de controlo para sistemas de telemanipulação podem ser classificadas considerando o fluxo de informação:

- **Unilateral:** a comunicação é efectuada num só sentido, em que o movimento e/ou a força do robô *master* é transmitida ao *slave* (seguimento de trajectória);
- **Bilateral:** a comunicação é efectuada em ambos os sentidos, sendo o mais usual a transmissão do movimento do robô *master* para o *slave* (seguimento de trajectória), havendo o *feedback* de força do *slave* para o *master*.

Nesta fase, pretende-se com o desenvolvimento e implementação do sistema robótico *master/slave*, que este permita um tipo de controlo unilateral, sendo o principal objectivo a implementação de seguimento de trajectórias: o robô *slave* deve seguir a trajectória do robô *master*, com ou sem factor de escala. No futuro a pretensão é avançar para um controlo do tipo bilateral (Cavusoglu, *et al.*, 2002), com *feedback* de força, o que implicará o desenvolvimento de um controlo do tipo força/impedância.

Outra característica de funcionamento deste sistema robótico é o tipo de ligação entre os controladores do robô *master* e do robô *slave*. A ligação entre os dois sistemas de controlo corresponde a uma comunicação dedicada do tipo *UTP* (cabo *crossover*), o que permite a troca de informação de uma forma segura e com elevada cadência. Um próximo passo nesta área será a transição para um tipo de comunicação partilhada (ex. *Internet*) (Andreu, *et al.*, 2008), isso implicará o desenvolvimento de algoritmos de controlo que sejam suficientemente robustos para poderem gerir eventuais perdas de informação e problemas de cadência que a utilização desse meio de comunicação acarreta.

4.3 Descrição do *Phantom Haptic Device*

No sistema robótico *master/slave* implementado, o robô *master* corresponde a um *Haptic Device*, sendo que foi escolhido o modelo comercial *Phantom Premium 1.5 High Force*, da *Sensable Inc*. (EUA). Este dispositivo possui um espaço de trabalho (381x267x191 mm) aproximado ao do antebraço humano, girando em torno do cotovelo. Possui 3 graus de liberdade de posicionamento (x,y,z) com *feedback* de força. Além disso, o dispositivo dispõe de um *gimbal encoder* que permite a utilização de mais 3 graus de liberdade (*pitch*, *rool*, *yaw*) sem *feedback* de força. Este *Haptic Device* possui uma interface com o *PC* através de uma porta paralela.

Em seguida são apresentadas as principais características técnicas do *Phantom Haptic Device*:

- Resolução de posição: 0.007 mm
- Força máxima exercida: 37.5 N
- Força contínua exercida: 6.2 N
- Rigidez: 3.5 N/mm
- Inércia (massa aparente na extremidade) sem o *gimbal encoder*: 150 g

A força motriz dos eixos é exercida por três servomecanismos, cada um composto por um servomotor DC e uma transmissão mecânica por cabos muito peculiar.

Nesta fase o *Haptic Device* é utilizado para enviar comandos de posição ao robô *slave*, não sendo utilizada a sua capacidade de *display* de força para o utilizador. Essa capacidade será importante quando o sistema evoluir para uma comunicação bilateral entre *master* e *slave*.

4.4 Arquitectura do Sistema Robótico de Telemanipulação

A arquitectura de controlo do sistema robótico baseia-se numa configuração *Host/Target* (figura 4.4) (Lopes, *et al.*, 2007). O *Host PC* corresponde a um *Desktop,* funcionando como plataforma de desenvolvimento, bem como centro de supervisão e comando do sistema de controlo *master*. Enquanto que o *Target PC* desempenha o papel de controlador do sistema robótico *slave*, uma vez que a aplicação responsável pelo controlo e operação do robô *slave* (robô cartesiano) é executada no *Target PC*, no *kernel* de tempo-real (*xPC Target*).

O *Host PC* comunica com o *Target PC* através de uma ligação dedicada (cabo *crossover*), embora possa optar-se pelo uso de uma ligação do tipo *LAN* (cabo *UTP*). Uma ligação do tipo *LAN* requer especiais cuidados, uma vez que este tipo de ligação acarreta problemas associados ao controlo global do sistema, tais como atrasos temporais e perdas de informação, que têm de ser geridas eficazmente pelo controlador do sistema, não sendo de momento esse um dos objectivos deste trabalho. Assim, o tipo de ligação (dedicada – cabo *crossover*) implementado é caracterizado pela sua elevada taxa de transferência de dados, na ordem dos 100 Mbps. Para que se estabeleça a ligação entre o *Host PC* e o *Target PC* é necessário que ambos possuam cartas de rede, sendo que a carta de rede instalada no *Target PC*, suportada pelos *drivers* disponibilizados pelo módulo *xPC Target*, foi a *Intel PRO 100/S*.

A comunicação entre o *Host PC* e o *Phantom Haptic Device* é efectuada através de uma interface porta paralela, sendo este tipo de interface caracterizado pela sua alta taxa de transferência de dados. No entanto, a utilização do *Phantom Haptic Device* com este tipo de interface trás algumas desvantagens face à interface *PCI*; a implementação de um controlador de tempo-real para um *Phantom Haptic Device* dotado de uma interface *PCI* encontra-se muito mais facilitada. A *Quanser* (Canadá) é uma empresa dedicada ao desenvolvimento, entre outros produtos, de soluções de controlo em tempo-real para *Haptic Devices*, disponibilizando *software* e *hardware* adequado. Neste momento dispõe de uma solução para o controlo em tempo-real do modelo *Phantom Haptic*

Device utilizado neste trabalho, sendo que as versões que disponibiliza de momento possuem interface *PCI* entre o dispositivo e o *PC*, encontrando-se ainda em fase de desenvolvimento uma aplicação para os dispositivos que dispõem de uma interface do tipo porta paralela.

Como será visto mais adiante, por não ser possível aceder ao *Phantom Haptic Device* em tempo-real, não é possível o envio de referências de posição do *master* para o *slave* a uma taxa constante.

A escolha do *Target PC* recaiu sobre um *Desktop*. No *Target PC* estão instaladas as cartas de aquisição de dados, que dispõem de uma interface do tipo *PCI*. Estas cartas são responsáveis pela interface entre o sistema de controlo e o robô cartesiano.

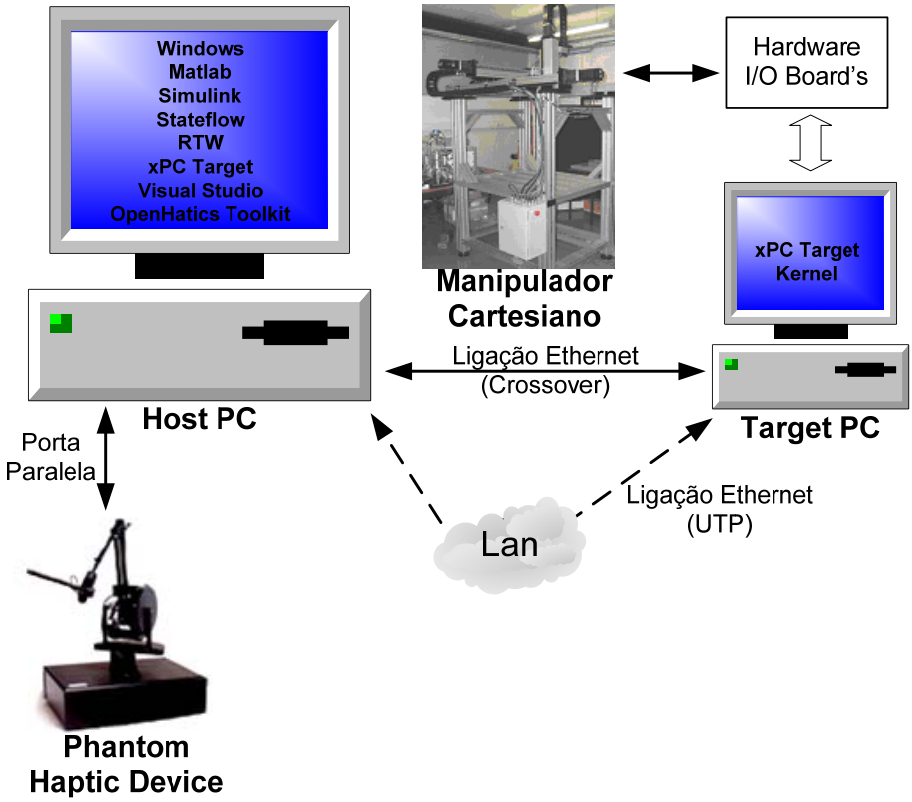


Figura 4. 1 – Arquitectura do *Hardware* do Sistema Robótico de Telemanipulação

Como pode observar-se na figura 4.1, o *Phantom Haptic Device* transmite as informações de posição e velocidade de cada um dos eixos ao *Host PC*, onde são processadas através de uma aplicação gráfica (interface gráfica de comando) e posteriormente enviadas para o *Target PC*. O *Target PC* recebe as coordenadas de

posição e velocidades provenientes do *Host PC* e atribui-as como referências a cada controlador de eixo respectivo, gerando-se assim as correspondentes acções de controlo (referências de binário) que são transmitidas aos *drivers* de potência.

Durante a operação do sistema robótico, as variáveis de posição, velocidade e aceleração (e força futuramente) de cada eixo, bem como os sinais digitais presentes no quadro eléctrico e os fins-de-curso (mecânicos e indutivos) do robô cartesiano são enviados do *Target PC* para o *Host PC*, onde são processados e monitorizados pela interface gráfica de comando. Futuramente será incorporado um transdutor de força na extremidade do órgão terminal do robô cartesiano (*slave*), cujo *device driver* para *Matlab/xPC Target* já foi desenvolvido no âmbito deste trabalho, que possibilitará a implementação de um controlador de força/impedância, sendo que as variáveis de força resultantes desse controlador serão enviadas do *Target PC* para o *Host PC* e deste para o *Phantom Haptic Device*.

4.5 Conclusões

Neste capítulo foram apresentadas as características mais importantes do *Phantom Haptic Device* (*Phantom Premium 1.5 High Force*, da *Sensable Inc*), tais como espaço de trabalho, força máxima suportada, rigidez e inércia.

Por último foi descrita a arquitectura do sistema robótico de telemanipulação desenvolvido, descrevendo-se o tipo de interligação de todos os componentes do sistema e o fluxo de informação entre eles. O sistema implementado é do tipo *master/slave*, em que tanto o *master* como o *slave* possuem controladores autónomos que interagem entre si. O meio de comunicação entre os dois sistemas (controladores) corresponde a uma canal de comunicação dedicado (*Ethernet* – cabo *crossover*), o que permite a transmissão de dados de uma forma fiável e segura.

Capítulo 5

Arquitectura do *Software* de Controlo do Sistema Robótico de Telemanipulação

5.1 Introdução

Neste capítulo será descrita a arquitectura de *software* do sistema de comando e controlo. Serão descritas as várias aplicações desenvolvidas, nomeadamente a aplicação gráfica de comando (*Visual Studio*) e a aplicação de controlo (*Simulink*).

Todo o *software* foi desenvolvido no *Host PC* que, como já foi mencionado, funciona como plataforma de desenvolvimento. Deste modo, no *Host PC* foram instalados o sistema operativo *Windows XP Professional*, da *Microsoft*, o *software* de cálculo numérico e de desenvolvimento/modelação/simulação de sistemas *Matlab/Simulink*, da *Mathworks*, e o *Microsoft Visual Studio* para o desenvolvimento da interface gráfica de comando e de uma *DLL* em C^{++}. O uso do *Matlab/Simulink* permitiu explorar as potencialidades dos módulos *xPC Target* e *Stateflow* na implementação da estratégia de controlo do sistema robótico. No que diz respeito ao *Phantom Haptic Device*, foi necessário a instalação dos *device drivers*, fornecidos pelo fabricante, no *Host PC*.

No *Target PC* é executado o *kernel* de tempo-real do *xPC Target*, que é ideal para aplicações de controlo robusto, uma vez que executa em exclusivo e em tempo-real a aplicação de controlo.

Para possibilitar a operação do sistema robótico de telemanipulação, foi necessário o desenvolvimento de uma aplicação gráfica de comando construída em *Microsoft Visual Studio* ($C^{\#}$), que permite a supervisão e controlo do sistema robótico. Através do *Microsoft Visual Studio* foi também possível desenvolver uma *DLL* em C^{++}. Essa *DLL* permite aceder à biblioteca de funções disponibilizada pelo fabricante do *Phantom Haptic Device*, sendo possível aceder a todas as funcionalidades deste dispositivo. Para se obter a biblioteca de funções do dispositivo é necessária a instalação do *software* de desenvolvimento de aplicações gráficas o *OpenHaptics Toolkit*, sendo que este *software* necessita que sejam instalados previamente os *Phantom Device Drivers* (para o sistema operativo *Windows*).

A *DLL* em C^{++} que foi desenvolvida permite aceder a variáveis como as posições, velocidade e forças que traduzem o comportamento operacional do *Phantom Haptic Device*. Neste caso, foi necessário a obtenção dos valores de posição e velocidade dos eixos (x,y,z) do *Phantom Haptic Device* (*master*), para que estas sejam transmitidas, com ou sem factor de escala, ao controlador do robô cartesiano (*slave*). Tal como já foi referido no capítulo anterior, futuramente a incorporação de um transdutor de força possibilitará a transmissão para o *Phantom Haptic Device* dos valores de força exercidos sobre o órgão terminal do robô cartesiano.

5.2 Arquitectura do *Software*

A arquitectura do *software* de controlo do sistema robótico de telemanipulação encontra-se dividida em duas aplicações. Uma das aplicações corresponde a um conjunto de algoritmos desenvolvidos com base no *software Matlab/Simulink/Stateflow*, que permitem o controlo das acções do robô *slave* (cartesiano). A outra aplicação desenvolvida em *Microsoft Visual Studio* permite através de uma *DLL* elaborada em C^{++} aceder e enviar informação para o *Phantom Haptic Device*, sendo que esta aplicação possibilita também a transferência de informação entre os dois sistemas robóticos (*master/slave*) (figura 5.1).

Deste modo, a *DLL* construída em C^{++} acede à biblioteca de funções disponibilizadas pela *OpenHaptics Toolkit*, o que permite obter os dados de posição e velocidade de cada eixo do *Phantom Haptic Device*. Os dados obtidos a partir da *DLL* são transferidos com ou sem factor de escala para o controlador do robô cartesiano (*Target PC*), sendo que esses dados são utilizados como referências dos controladores de eixo do robô cartesiano.

A aplicação de controlo do robô cartesiano, tal como já foi referido, recebe da aplicação instalada no *Host PC*, os valores instantâneos de posição e velocidade provenientes da leitura dos sensores do *Phantom Haptic Device*. É através dessa informação que, em conjunto com a informação obtida pelos sensores e detectores instalados no robô cartesiano, o controlador do mesmo gera as correspondentes acções de controlo, bem como executa os procedimentos adequados para a realização das tarefas de telemanipulação.

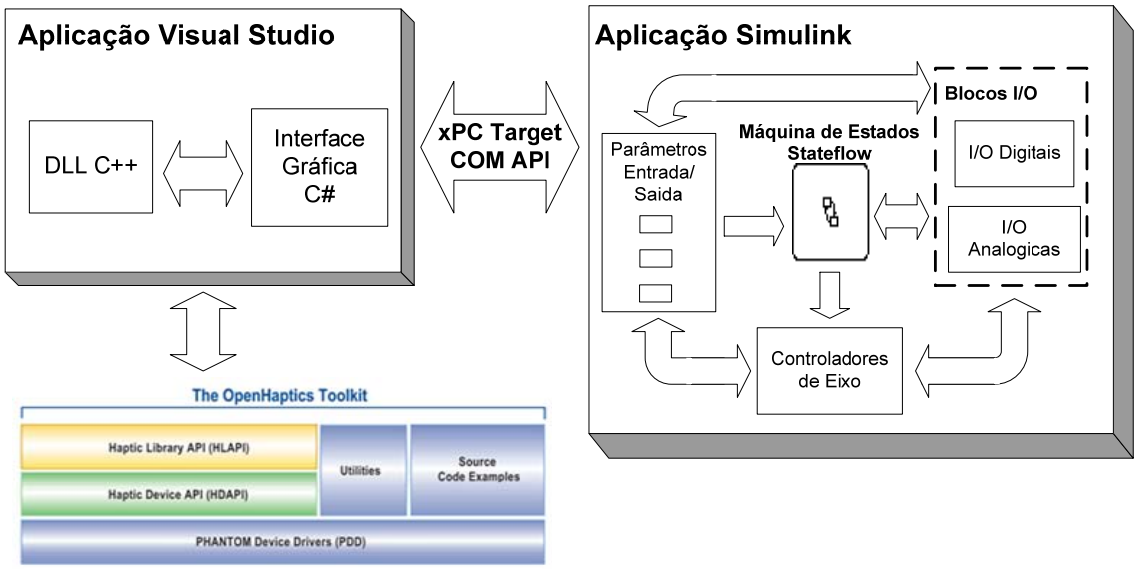


Figura 5. 1 – Arquitectura do *Software* do Sistema Robótico *Master/Slave*

A aplicação de controlo do robô cartesiano é composta por quatro componentes distintas mas que interagem umas com as outras (figura 5.1). Desta forma, uma dessas componentes corresponde a "blocos" de *parâmetros de entrada e saída* de dados, que são transferidos de e para a aplicação de monitorização e comando instalada no *Host PC*. As outras duas componentes essências da aplicação de controlo do robô cartesiano são os *controladores de eixo* e *máquina de estados* do sistema, pois são elas que geram as ordens e acções de comando atribuídas ao robô cartesiano. Sendo que essas a acções são geradas mediante os parâmetros enviados pelo controlador do *Phantom Haptic Device* e os dados obtidos pelos sensores e detectores instalados no robô cartesiano. A quarta componente da aplicação de controlo do robô *slave* corresponde aos *drivers* das cartas digitais e analógicas responsáveis pela interface entre a aplicação de controlo e o meio físico que interage com o robô cartesiano.

5.3 Aplicação de Controlo – *Simulink*

Nesta subsecção serão descritas as componentes da máquina de estados *Stateflow*, bem como os procedimentos inerentes à activação dos estados da mesma. Serão também descritos os subsistemas que compõem a aplicação de controlo e que implementam as diversas acções decorrentes da interacção entre o robô *master* e o robô *slave*, bem como a interacção entre o robô *slave* e o meio ambiente.

5.3.1 Máquina de Estado – *Stateflow*

De seguida são descritos os estados mais importantes que compõem a máquina de estados, para cada um deles é descrito o procedimento inerente a uma dada acção do utilizador ou em resposta à ocorrência de um evento.

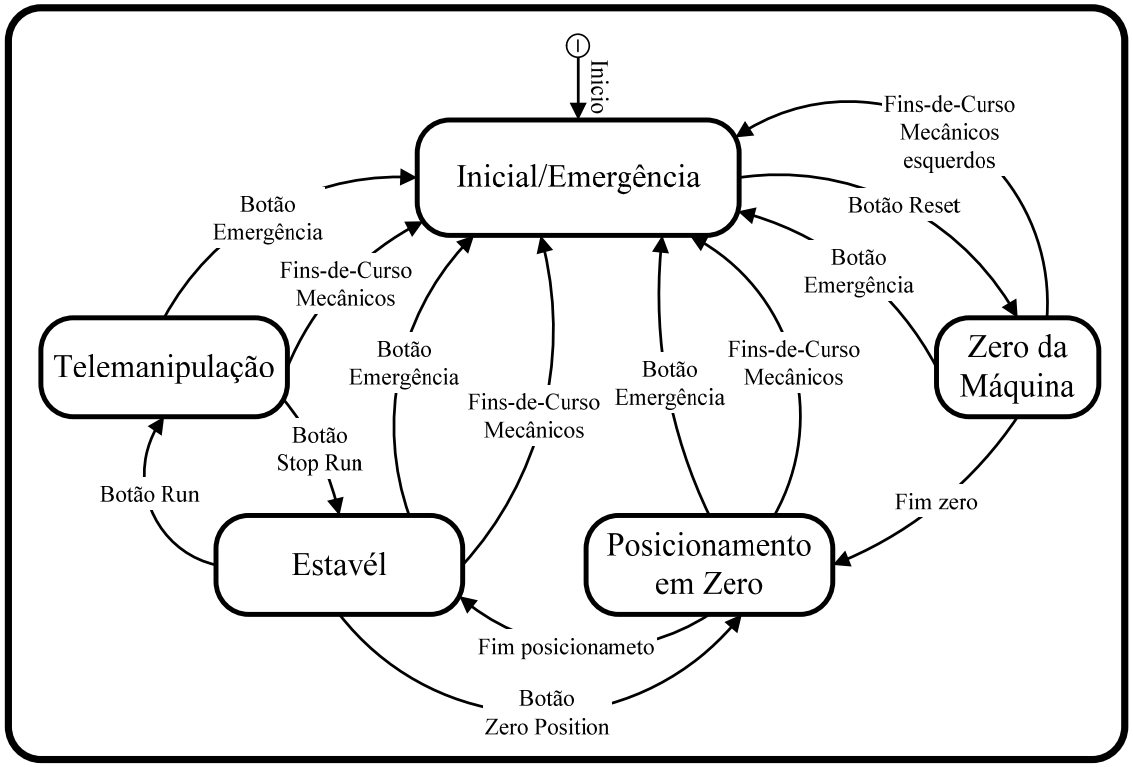


Figura 5. 2 – Diagrama da Máquina de Estado do Controlador

O diagrama de estados da figura 5.2 representa a máquina de estados implementada na construção do controlador do robô cartesiano. Conforme é possível observar no

diagrama de estado existe um estado inicial (*Inicial/Emergência*) correspondente ao ponto de partida, onde os *drivers* de potência são activados sendo-lhes fornecidas referencias nulas, neste estado é também desactivado o travão do eixo vertical do robô cartesiano, desta forma, o controlador está preparado para qualquer ordem de movimento.

Tal como se pode verificar na figura 5.2 o controlador só sai do estado inicial se for dada ordem para se efectuar o procedimento designado *Zero da Máquina*, que corresponde ao posicionamento do robô cartesiano num ponto de referência no espaço cartesiano, a partir desse momento todas os movimentos efectuados têm como referência esse ponto. Sempre que se inicia uma nova sessão de trabalho do sistema de telemanipulação, o primeiro procedimento a ser efectuado é o *Zero da Máquina*. No decorrer deste procedimento, caso o utilizador entenda que por medidas de segurança deva abortar o movimento do robô cartesiano, pode fazê-lo, bastando para isso accionar o botão de emergência na interface gráfica de comando ou pressionar uma das botoneiras que fazem parte da estrutura do robô cartesiano. Quando o utilizador aborta a execução do procedimento *Zero da Máquina*, ocorre uma transição na máquina de estados do estado *Zero da Máquina* para o estado *Inicial/Emergência*. Desta forma, o controlador regressa ao estado inicial com a diferença de os *drivers* de potência ficarem inactivos e o travão do eixo vertical ficar activado.

Após efectuar-se o *Zero da Máquina* o robô cartesiano fica posicionado num determinado ponto no espaço cartesiano que é diferente do (0,0,0), uma vez que para garantir que este procedimento efectue o *reset* dos *encoders* de cada eixo é necessário que o robô cartesiano ultrapasse ligeiramente a posição (0,0,0). Assim, depois de efectuado o procedimento *Zero da Máquina* e antes de se transitar para o estado de *Telemanipulação* é necessário efectuar-se o posicionamento do robô cartesiano na posição (0,0,0), para que desta forma se garanta que tanto o *Haptic Device* como o robô cartesiano se encontrem com a mesma referência de posição quando se inicia a operação de telemanipulação, garantindo que se inicie essa operação de uma forma segura e sincronizada. Desta forma, quando é terminado o procedimento *Zero da Máquina*, a máquina de estados transita para o estado *Posicionamento em Zero*, para que o robô cartesiano fique posicionado na posição (0,0,0).

No fim do posicionamento do robô cartesiano ocorre uma transição do estado *Posicionamento em Zero* para o *Estável*. A transição para o estado *Estável* é necessária para assegurar que antes do início da operação de telemanipulação o robô cartesiano se encontra em controlo de posição, sendo que as referências de posição fornecidas aos controladores de eixo são (0,0,0).

Quando o utilizador ordena o início da operação de telemanipulação ocorre uma transição do estado *Estável* para o *Telemanipulação*. A partir desse momento é executada a operação de telemanipulação, sendo que as referências de posição e velocidade do *Haptic Device* são fornecidas, com um factor de escala associado, aos controladores de eixo do robô cartesiano. Durante a permanência no estado *Telemanipulação*, caso venha a verificar-se alguma situação anómala ao correcto funcionamento da operação de telemanipulação, existem transições de segurança desse estado para o estado *Inicial/Emergência*.

Durante a permanência nos estados que impliquem o movimento do robô cartesiano, no caso de ocorrer um evento associado a uma condição de segurança, nomeadamente, a actuação de um fim-de-curso mecânico, a actuação por parte do utilizador do botão da interface gráfica *Emergency* ou de uma das botoneiras de emergência, ocorre uma transição na máquina de estados para o estado *Inicial/Emergência*.

De seguida são descritos os procedimentos associados aos estados que compõem a máquina de estados (figura 5.2):

- **Inicial/Emergência:**

  Como já foi referido no capítulo 2, este estado está associado a um subsistema *Simulink* que é responsável por fornecer referências nulas aos *drivers* de potência, com excepção do eixo Z que está sujeito à influência da força gravítica, necessitando de uma referência suficiente para se manter equilibrado.

  A partir deste estado, se o utilizador actuar no botão *Reset* da interface gráfica, ocorre um evento que origina a transição para o estado *Zero da Máquina*.

- **Zero da Máquina:**

Este estado já foi descrito no capítulo 2, sendo utilizado para implementar o procedimento do *reset* dos *encoders* do robô *slave*. No fim deste procedimento ocorre um evento associado a uma *flag* que provoca a transição do estado *Zero da Máquina* para o *Posicionamento em Zero*.

- **Posicionamento em Zero:**

Este estado tem associados dois subsistemas *Simulink* que implementam a acção do posicionamento do robô cartesiano na posição (0,0,0). Um dos subsistemas é responsável por calcular as componentes de velocidade média de cada eixo para execução da trajectória. O outro subsistema implementa o gerador de trajectórias (com perfil sinusoidal de velocidade) necessário para fornecer as referências de posição e velocidade aos controladores de eixo, para que seja efectuada a trajectória de modo suave.

No fim do posicionamento do robô cartesiano em (0,0,0) ocorre um evento associado ao termino deste procedimento, o que provoca uma transição do estado *Posicionamento em Zero* para o *Estável*.

- **Estável:**

Este estado está associado a um subsistema *Simulink* que fornece aos controladores de eixo referências fixas de posição. Deste modo, o robô cartesiano permanece fixo numa posição, possibilitando ao utilizador efectuar outras tarefas que não seja a telemanipulação, uma vez que neste estado o robô cartesiano permanece sobre controlo de posição, com as referências de posição que ficaram gravadas no momento da entrada no estado.

Quando este estado está activo pode dar-se uma transição para o estado *Telemanipulação* em consequência do evento associado à actuação, por parte do utilizador, do botão da interface gráfica *Run*. Outro evento que pode originar uma

transição deste estado, mas para o estado *Posicionamento em Zero*, é o facto do utilizador actuar na interface gráfica o botão *Zero Position*.

- **Telemanipulação:**

Este estado está associado a um subsistema que é responsável pelo fornecimento das referências de posição e velocidade aos controladores de eixo do robô cartesiano, estas referências são provenientes do *Phantom Haptic Device* e são multiplicadas por um factor de escala escolhido pelo utilizador.

Enquanto este estado permanecer activo implica que a operação de telemanipulação esteja a ser efectuada. Quando por ocorrência de um evento que corresponda a uma situação de emergência, tal como o accionar por parte do utilizador do botão *Emergency* da interface gráfica ou através do accionamento de uma botoneira de emergência ou no caso de um dos fins-de-curso mecânicos serem activados, dá-se uma transição do estado *Telemanipulação* para o *Inicial/Emergência*. Se o utilizador pretender suspender a tarefa de telemanipulação através do botão da interface gráfica *Stop,* ocorre um evento que provoca a transição da máquina de estados para o estado *Estável*.

5.3.2 Procedimento – Zero da Máquina (Robô *Slave*)

Este procedimento já foi descrito no capítulo 2. É idêntico ao utilizado no funcionamento do robô cartesiano em modo autónomo, tendo como objectivo efectuar o *reset* do *encoder* de cada eixo.

5.3.4 Procedimento – Posicionamento em Zero

A finalidade deste procedimento consiste em sincronizar ambos os sistemas robóticos (*master* e *slave*). Assim, quando se inicia uma tarefa de telemanipulação ambos os sistemas devem possuir as mesmas coordenadas de posição (0,0,0), evitando que ocorra dessincronização entre os dois sistemas.

Para a implementação deste procedimento foram criados dois subsistemas em *Simulink*. Um dos subsistemas é responsável pelo cálculo das velocidades médias que estão associadas à trajectória executada por cada eixo (figura 5.3). O cálculo dessas velocidades médias é efectuado da forma que se segue.

Considerando uma trajectória linear, num referencial cartesiano, entre dois pontos $\mathbf{P_0\,(x_0, y_0, z_0)}$ e $\mathbf{P_1\,(x_1, y_1, z_1)}$, com uma velocidade média, $\mathbf{V_{méd}}$, a distância a percorrer é dada pela expressão que se segue:

$$Distância = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2} \qquad 5.1$$

Sabendo que o período de tempo de execução da trajectória é obtido por:

$$T = \frac{Distância}{V_{méd}} \qquad 5.2$$

obtêm-se as três componentes da velocidade correspondentes a cada eixo:

$$Vx = \frac{|x_0 - x_1|\, V_{méd}}{Distância};\ Vy = \frac{|y_0 - y_1|\, V_{méd}}{Distância};\ Vz = \frac{|z_0 - z_1|\, V_{méd}}{Distância} \qquad 5.3$$

O algoritmo apresentado anteriormente é implementado através de uma *S-Function* (figura 5.3), que tem como entradas as coordenadas de posição do ponto de partida, $\mathbf{P_0\,(x_0, y_0, z_0)}$, as coordenadas de posição do ponto final (0,0,0) e a velocidade média de execução de trajectória, ajustada em 20 mm/s. As saídas dessa *S-Function* são o

resultado do algoritmo implementado, ou seja, as componentes de velocidades médias de cada eixo.

O valor de velocidade média está predefinido, uma vez que este procedimento consiste em executar o posicionamento do robô cartesiano no ponto (0,0,0). Sendo que, a posição inicial do robô pode ser qualquer ponto do espaço de trabalho, o que implica a execução de uma trajectória desconhecida do utilizador, por esta razão foi definido um valor de velocidade média reduzido (20 mm/s, de acordo com os testes efectuados).

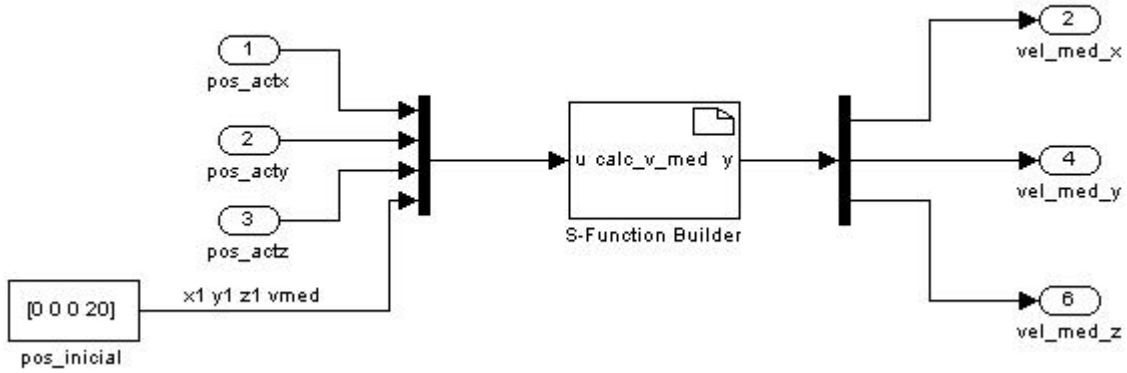


Figura 5. 3 – Subsistema de Cálculo das Componentes de Velocidades Médias

Após obtidas as componentes de velocidades médias de cada eixo, estas são fornecidas ao subsistema (figura 5.4) responsável pela implementação do gerador de trajectórias. Esse subsistema gera a trajectória para cada eixo a partir dos dados de velocidade média, posição actual, posição final (0,0,0) e da variável tempo. Para cada eixo são geradas as referências de posição, velocidade e aceleração distintas.

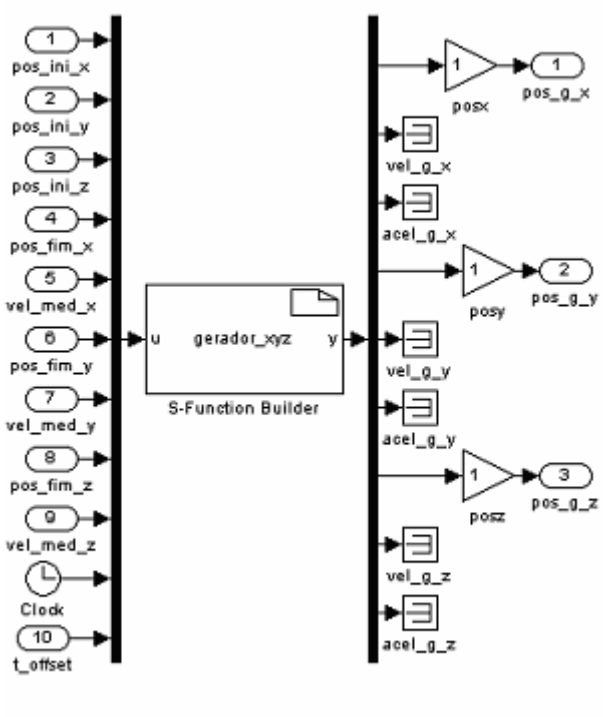


Figura 5. 4 – Gerador de Trajectórias (Posicionamento em Zero)

No caso do robô *master*, a definição do ponto (0,0,0) é efectuada manualmente pelo utilizador, que deve escolher um ponto próximo do centro do espaço de trabalho desse robô.

5.3.5 Telemanipulação

Neste caso o procedimento de telemanipulação consiste em enviar as coordenadas de posição (com ou sem factor de escala) do robô *master* para o controlador do robô *slave*. Essas coordenadas servem de referência para os controladores de eixo (PID) do robô *slave*. Deste modo, é implementado um seguimento de trajectória do robô *master* por parte do robô *slave*.

Este procedimento foi implementado através de um subsistema do *Simulink* (figura 5.5) que recebe as coordenadas de posição e velocidade provenientes do *Haptic Device* (valores obtidos utilizando a biblioteca de funções do dispositivo), essas coordenadas são enviadas como referências para os controladores de eixo. Assim, o robô *master* funciona como o gerador de trajectórias do robô *slave*.

O controlador do robô *slave* processa em tempo-real as acções de controlo, em resposta às referências de posição e velocidade do robô *master*. Essas acções de controlo são enviadas para os *drivers* de potência, que fazem actuar os servomotores de cada eixo do robô *slave*.

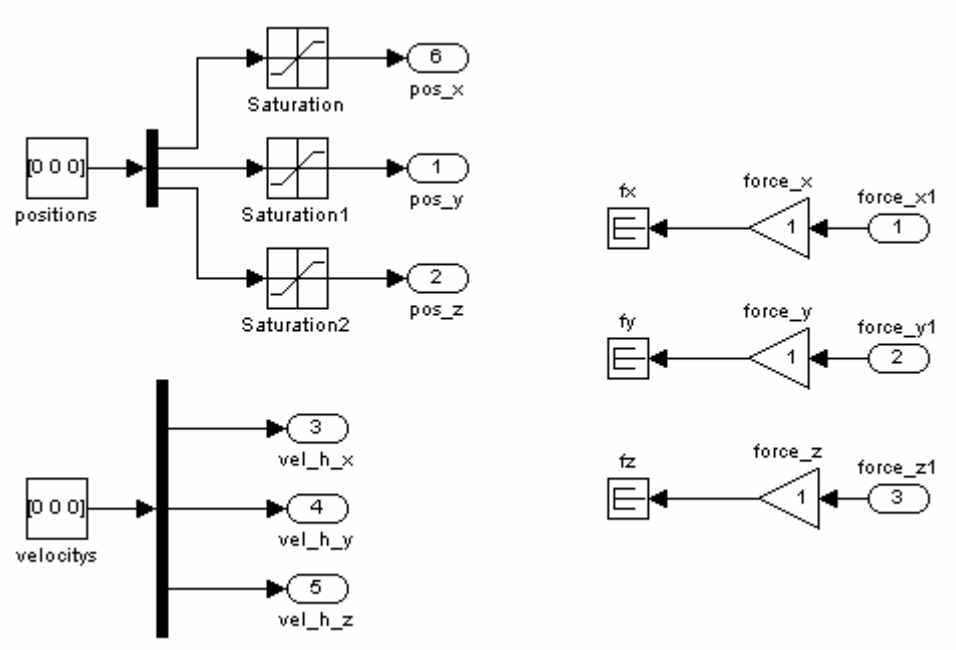


Figura 5. 5 – Subsistema *Simulink* de Telemanipulação

Por uma questão de segurança, as coordenadas de posição do robô *master* (multiplicadas por um factor de escala) fornecidas aos controladores de eixo (PID) do robô *slave*, estão restringidas ao espaço de trabalho deste último, através dos blocos *"Saturation"* (figura 5.5).

5.4 *OpenHaptics Toolkit* – C++ *DLL*

Nesta subsecção será descrito o desenvolvimento da *DLL* construída em C^{++} que permite o acesso à biblioteca de funções cedida pelo fabricante do *Phantom Haptic Device*, que possibilita a interacção entre a interface gráfica de comando e o *Haptic Device*.

Para se ter acesso à biblioteca de funções do *Phantom Haptic Device* é necessário a instalação dos *drivers* do dispositivo (*Phantom Haptic Device*). Através do conjunto de funções que compõem a biblioteca de funções do *Haptic Device* é possível aceder com propriedades de leitura às variáveis de posição e velocidade e com propriedades de escrita à variável força. É também possível aceder a funções que permitem a inicialização do dispositivo, o *reset* dos *encoders* e o lançamento do processo de escalonamento. Todas estas funções são disponibilizadas pelo *software OpenHaptics Toolkit* da *Sensable*, através do uso da *Haptic Device API* (*HDAPI*). Com este pacote de *software* vem incluído o *Phantom Device Driver*.

Em seguida são enumeradas algumas das principais características e funcionalidades da *HDAPI* (figura 5.6):

Características:

- Uma interface optimizada com o dispositivo *Phantom Haptic Device*;
- Independente da plataforma do Sistema Operativo utilizado;
- Controlo por *software* de:
    - Acesso directo aos valores dos *encoders* e controlo da tensão aplicada aos motores;
    - Definição da taxa *servo loop* para obtenção de melhores performances;
    - Funções de tratamento de erros e imposição de limites de segurança.
- A *API* inclui utilidades e exemplos de código fonte.

Funcionalidades:

- Monitorização do estado do dispositivo:

    - Posição e Velocidade;
    - 3 coordenadas no espaço: Cartesiano e das juntas.
- Aquisição das características do dispositivo:
    - Dimensões do espaço de trabalho;
    - Temperatura dos motores;
    - Velocidades e Forças máximas.
- Impor ao dispositivo:
    - Forças/Momentos no espaço cartesiano;
    - Referências de tensão aplicadas aos motores;
- Escalonador:
    - Definição pelo utilizador de *Callbacks* Assíncronas e Síncronas;
    - Escalonamento dessas *Callbacks*;
    - Definição pelo utilizador da taxa de servo loop.
- *Enable/Disable*:
    - Forças: Saída, fixação e rampas;
    - Notificação de erros;
- Interface de calibração (*reset* dos *encoders*) do dispositivo:
    - Calibração Automática;
    - Calibração Manual.
- Utilidades:
    - *Wrapper* para C^{++}.

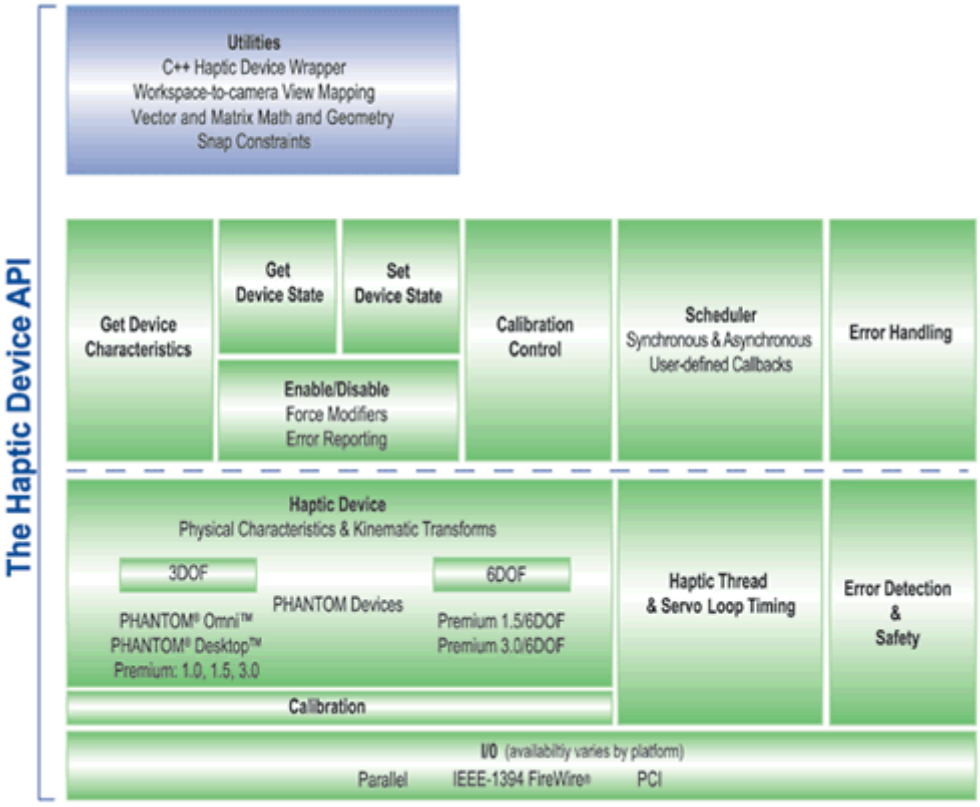


Figura 5. 6 – Diagrama de Funcionalidades da *Haptic Device API* (*HDAPI*)

Através desta *API* foi possível construir uma *DLL* em C^{++} com utilização de funções que possibilitam a interacção com o *Phantom Haptic Device*. A *DLL* foi construída em *Visual Studio .Net*, o que permitiu a criação de um conjunto de estrutura de dados que pode ser facilmente acedida pela interface gráfica desenvolvida em C# (figura 5.7), uma vez que a *DLL* é adicionada como referência ao projecto de desenvolvimento associado à interface gráfica.

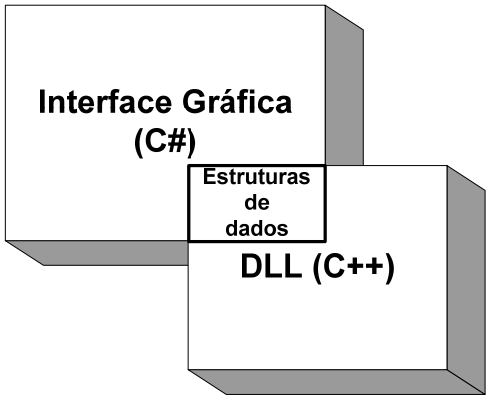


Figura 5. 7 – Estrutura da Aplicação *Visual Studio .Net*

Desta forma, a partir de algumas das funções que compõem a *HDAPI*, foi possível a construção da *DLL* que permite a inicialização do dispositivo, o iniciar do escalonador, a calibração do dispositivo (*reset* dos *encoders*), monitorização do estado do *Haptic Device* (posições e velocidades), impor forças ao *Haptic Device* e o sair do *loop* de controlo (figura 5.8). Todas estas funcionalidades são implementadas com o recurso às seguintes funções e *callbacks* que compõem a *HDAPI*:

- hdInitDevice(HD_DEFAULT_DEVICE) – Inicialização do *Haptic Device*;
- hdEnable(HD_FORCE_OUTPUT) – *Enable* das forças;
- hdStartScheduler() – Iniciar do escalonador;
- hdUpdateCalibration(HD_CALIBRATION_ENCODER_RESET) – Calibração do dispositivo (*reset* dos *encoders*);
- HDCallbackCode HDCALLBACK DevicePositionCallback(void *pUserData) – *Callback* que retorna as coordenadas de posição (x,y,z) do *Haptic Device*;
- HDCallbackCode HDCALLBACK DeviceVelocityCallback(void *pUserData) – Callback que retorna os valores de velocidade (x,y,z) do *Haptic Device*;
- HDCallbackCode HDCALLBACK DeviceForceCallback(void *pUserData) – *Callback* que impõe os valores de força ao *Haptic Device*;
- hdGetDoublev(HD_CURRENT_POSITION, pPosition) – Função que permite obter as coordenadas de posição (x,y,z) do *Haptic Device*;
- hdGetFloatv(HD_CURRENT_VELOCITY, pVelocity) – Função que permite obter o valor de velocidade (x,y,z) do *Haptic Device*;
- hdSetDoublev(HD_CURRENT_FORCE, pForce) – Função que permite impor o valores de força ao *Haptic Device*;
- hdStopScheduler() – Função que permite parar a execução do escalonador, saindo-se desta forma do ciclo *servo loop*;
- hdDisableDevice(hHD) – Função que permite desactivar o *Haptic Device*.

De seguida são apresentadas as estruturas de dados criadas para permitir a troca de informação entre a interface gráfica *C#* e a *DLL C*$^{++}$:

- Dev_State – Estutura de dados que permite o armazenamento dos valores de posição e velocidade (x,y,z) do *Haptic Device*, que é fornecida pela *DLL C*$^{++}$ à interface gráfica *C#*;
- force3D – Estrututra de dados que permite o armazenamento dos valores de força, que são fornecidos pela interface gráfica *C#* a *DLL C*$^{++}$.

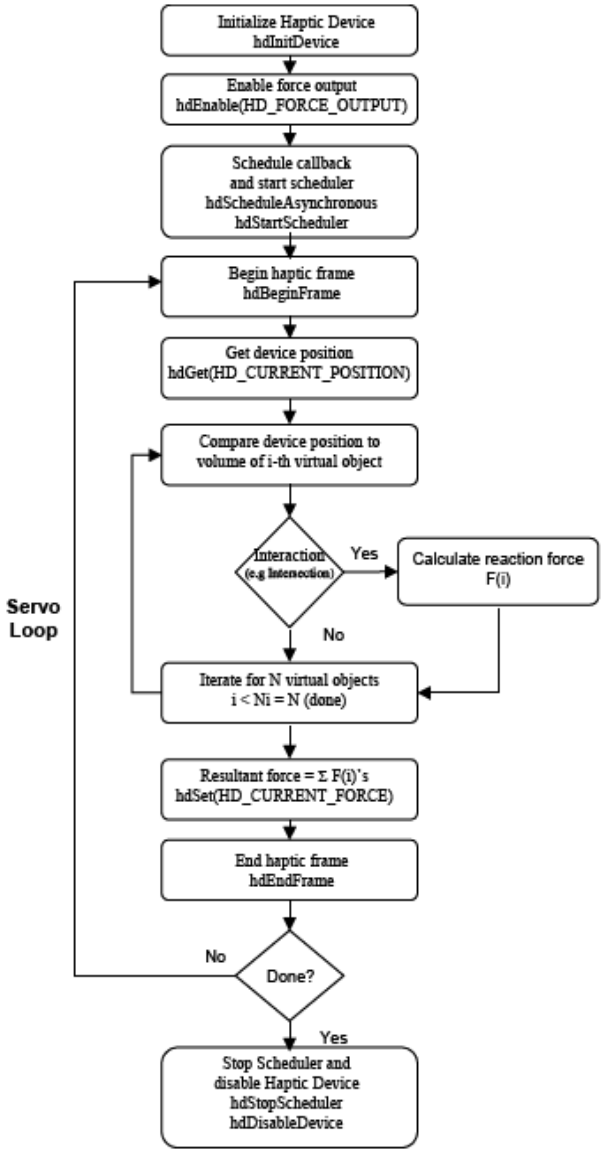


Figura 5. 8 – Fluxograma Típico de um Programa *HDAPI*

5.5 Conclusões

Neste capítulo foi descrita a organização da arquitectura de *software* implementada para efectuar o controlo do sistema robótico de telemanipulação. Deste modo, foram apresentadas as estruturas da aplicação de controlo (*Simulink*) do robô *slave*, bem como a estrutura da aplicação *OpenHaptics Toolkit* que permite a operação e interacção com o *Haptic Device*. Neste capítulo foi também descrita a forma como as várias componentes da arquitectura do *software* interagem entre si.

A possibilidade da integração da *xPC Target COM API* com a aplicação *Visual Studio* (interface gráfica *C#* e a *DLL* C^{++}, que são executadas no *Host PC*) permitiu a fácil troca de dados entre esta e a aplicação de controlo (*Simulink*) do robô *slave* que corre em tempo-real no *Target PC*.

A *DLL* construída em C^{++} possibilita o acesso a funções da *HDAPI* (biblioteca de funções do *Haptic Device*), o que permite aceder às variáveis e características do *Phantom Haptic Device*. A integração da *DLL* C^{++} na aplicação *C#* (interface gráfica) facilita o processo de acesso às variáveis (posição e velocidade) do *Haptic Device*, e posteriormente o envio dessas variáveis para a aplicação de controlo do robô *slave* (controladores de eixo).

O facto de a biblioteca de funções e os *drivers* do *Haptic Device* utilizados não serem desenvolvidos para sistemas operativos de tempo-real trás alguns problemas. O problema mais evidente é a cadência que se apresenta no seguimento de trajectória entre o robô *slave* e o robô *master*. Este e os outros problemas associados a esse facto poderão ser resolvidos futuramente, com a utilização de *drivers* (*Phantom Device Driver*) que possibilitem o uso de sistemas operativos de tempo-real. Nesse caso, será necessário o desenvolvimento de uma nova estratégia de controlo, que permita a integração entre o sistema de controlo de tempo-real (modulo *xPC Target*) do robô *slave* (cartesiano), já existente, e o novo sistema de controlo de tempo-real do robô *master* (*Haptic Device*).

Capítulo 6

Interface Gráfica de Comando do Sistema Robótico de Telemanipulação

6.1 Introdução

A interface gráfica (figura 6.1) foi desenvolvida em *Visual Studio .Net* na linguagem *C#*, sendo que, este tipo de linguagem é caracterizado pela sintaxe muito parecida com a linguagem *C*. A grande vantagem deste linguagem *C#* em relação a outro tipo de linguagens, como é o caso do C^{++}, deve-se ao facto desta se aproximar do *.NET CLR* (*Common Language Runtime*) (Gough, 2001). De salientar ainda a possibilidade de se adicionar como referência ao projecto de desenvolvimento da interface gráfica a *xPC Target COM API*, deste modo, é possível a interacção entre a interface gráfica de comando e a aplicação de controlo (*Simulink*) do robô *slave* (figura 5.1).

A interface gráfica permite ao utilizador a interacção com o sistema robótico de telemanipulação. Desta forma, o utilizador pode monitorizar o estado operacional de ambos os robôs que compõem o sistema robótico, bem como comandar a operacionalidade desse mesmo sistema robótico.

Através da interface gráfica, o utilizador pode estabelecer a comunicação com o controlador (tempo-real) do robô *slave* e com o robô *master*. Na interface gráfica o utilizador pode verificar quais os limites do espaço de trabalho de cada robô, pode também monitorizar as coordenadas de posição instantânea de ambos os robôs. É também possível monitorizar a operacionalidade do robô *slave*, nomeadamente, através dos sinais digitais dos fins-de-curso (mecânicos e indutivos) e dos sinais presentes no armário eléctrico de controlo.

Nas subsecções que se seguem são descritas os vários componentes da interface gráfica, bem como todas as suas funcionalidades.

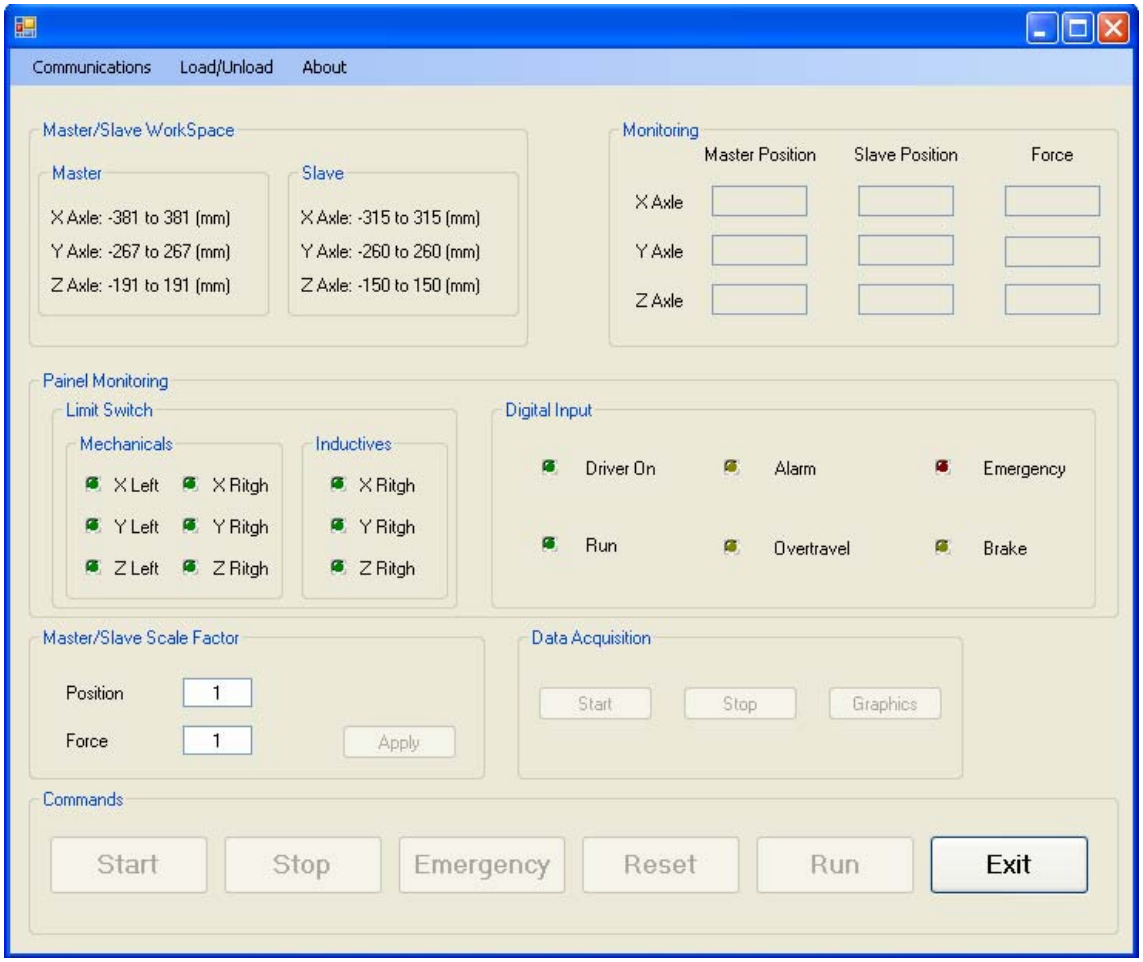


Figura 6. 1 – Interface Gráfica de Comando (*C#*) do Sistema Robótico de Telemanipulação

6.2 Painéis da Interface Gráfica

Nas subsecções que se seguem serão apresentados os painéis que constituem a interface gráfica. A interface gráfica está organizada em painéis, de maneira a melhorar a interacção do utilizador com o sistema robótico de telemanipulação.

6.2.1 Painel *Master/Slave Workspace*

No painel *Master/Slave Workspace* (figura 6.2) o utilizador pode obter informações importantes sobre o sistema robótico, nomeadamente, os espaços de trabalho do robô *master* (*Phantom Haptic Device*) e do robô *slave* (robô cartesiano).

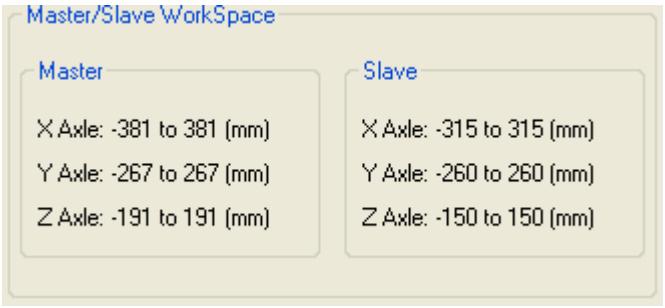


Figura 6. 2 – Painel *Master/Slave Workspace*

6.2.2 Painel *Monitoring*

No painel *Monitoring* o utilizador pode observar variáveis importantes do sistema físico, como as posições (x,y,z) do robô *master* e do robô *slave*. Neste painel também é possível a monitorização de sinais que são parte constituinte do sistema robótico (robô *slave*) através do *Painel Monitoring*, são eles:

- Fins-de-curso mecânicos:
  - Eixo X direito
  - Eixo X esquerdo
  - Eixo Y direito
  - Eixo Y esquerdo
  - Eixo Z direito
  - Eixo Z esquerdo

- Fins-de-curso indutivos:
  - Eixo X
  - Eixo Y
  - Eixo Z

Existem outros sinais que podem ser monitorizados, que fornecem importantes informações sobre o estado operacional do robô *slave*. Assim, como forma de representar o painel presente no armário eléctrico de controlo do robô *slave*, foram escolhidos os seguintes sinais para monitorização:

- *Driver On*
- *Run*
- *Alarm*
- *Overtravel*
- *Emergency*
- *Brake*

6.2.3 Painel *Master/Slave Scale Factor*

Este painel possibilita ao utilizador aplicar um factor de escala no seguimento de trajectória do robô *slave* em relação ao robô *master* (figura 6.3). Este factor de escala pode ser aplicado aos valores de posição e ao *feedback* de força (a implementar futuramente). Deste modo, o utilizador pode a qualquer momento efectuar um movimento com uma amplitude maior ou menor conforme a necessidade. Em relação ao *feedback* de força, este também poderá ser amplificado ou atenuado, possibilitando uma maior percepção por parte do utilizador das forças exercidas no robô *slave*, uma vez que na ausência do *feedback* visual o *feedback* de força se torna extremamente importante. Assim, o factor de escala pode aumentar a segurança do trabalho desenvolvido, o que se revela de extrema importância quando se está a realizar uma tarefa que exija maior sensibilidade por parte do utilizador, como é o caso de tarefas de telemedicina ou operação em ambientes hostis.

A possibilidade de aplicar o factor de escala aos valores de posição do robô *master*, que são enviados para o robô *slave*, permite entre outros aspectos ajustar o espaço de trabalho do robô *master* ao do robô *slave*.

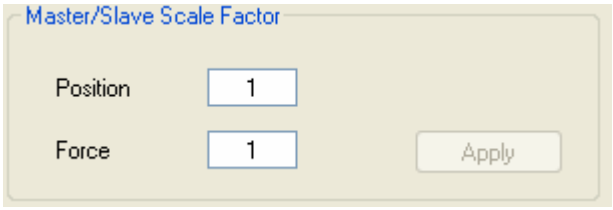


Figura 6. 3 – Painel *Master/Slave Factor*

6.2.4 Painel *Data Acquisition*

Este painel foi criado com o intuito de possibilitar ao utilizador a aquisição e o *log* (armazenamento) dos dados associados à operação de telemanipulação (figura 6.4). Os dados, como as posições e as velocidades do *Haptic Device* e do robô cartesiano, bem como as acções de controlo fornecidas pelos controladores de eixo aos *drivers* de potência dos motores, são guardados em ficheiro (formato *.txt*) no *Host PC* para posterior análise. Durante a execução do movimento estes dados são armazenados no disco do *Target PC*, sendo que através de algumas funções (métodos) da *xPC Target COM API* é possível aceder ao *filesystem* deste *PC*, recolhendo-se e convertendo-se essa informação em ficheiros (formato *.txt*) que são gravados no *Host PC*. A análise dos dados recolhidos pode ser efectuada actuando no botão *Graphics*. Deste modo, é iniciado um executável do *Matlab* que transforma os dados dos ficheiros em gráficos. Após a visualização dos gráficos que contêm a informação das trajectórias executadas, o utilizador pode utilizar as ferramentas e funcionalidades dos gráficos gerados a partir do executável do *Matlab*, uma das quais corresponde à gravação desses gráficos em vários tipos de formatos.

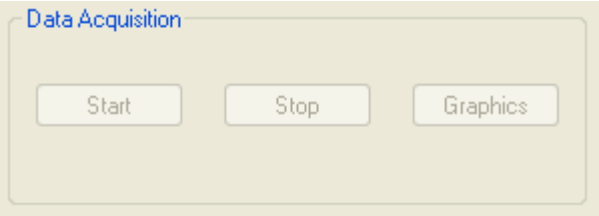


Figura 6. 4 – Painel *Data Acquisition*

Na imagem da figura 6.5 é possível visualizar um dos gráficos de posição gerados após a execução de uma tarefa de telemanipulação. Nesse gráfico é possível analisar o comportamento e o desempenho do sistema de controlo de posição (seguimento de trajectória) do robô *slave* em relação ao robô *master*. Pode verificar-se um pequeno desfasamento entre as posições do robô *master* e do *slave*, justificado pela cadência na obtenção das posições do robô *master*, uma vez que essa tarefa não é efectuada num sistema de tempo-real. Apesar do pequeno desfasamento existente entre as posições de ambos os robôs, pode constatar-se que o objectivo de seguimento de trajectória foi

alcançado, sendo que em regime de repouso ambos os robôs se encontram sincronizados (erro em regime permanente nulo).

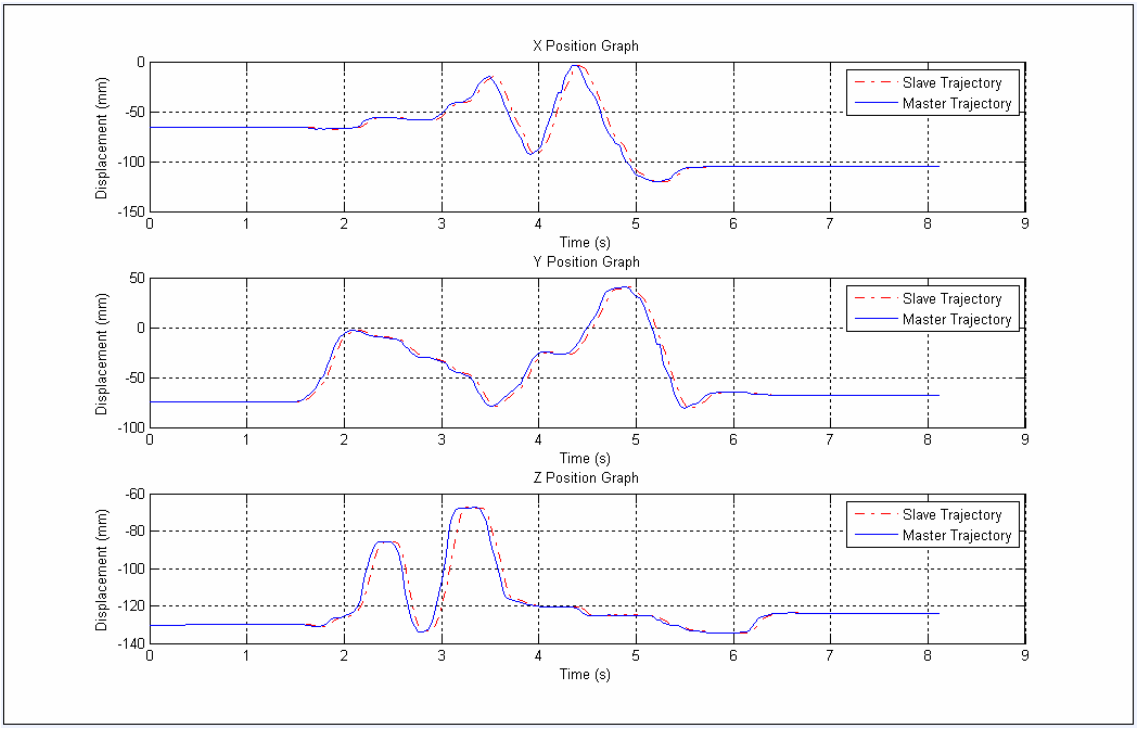


Figura 6. 5 – Gráficos de Posição do Robô *Master* e *Slave* (Factor de Escala Unitário)

6.2.5 Painel *Commands*

Este painel disponibiliza alguns botões com comandos específicos dentro da interface gráfica. A actuação de um destes botões provoca efeitos na interface gráfica, nomeadamente no desbloquear ou bloquear de certas funcionalidades, como também produz acções no sistema de controlo do sistema robótico de telemanipulação. Em seguida serão descritas as propriedades de cada um destes botões de comando.

- ***Start*:**

  Este botão é responsável pelo início de uma sessão. A partir desse momento o sistema está pronto a ser controlado, sendo o *Zero da Máquina* o primeiro procedimento que tem de ser efectuado. Por esse motivo o único botão que fica activo no início da aplicação é o botão *Reset*. Quando é accionado este botão, as rotinas de monitorização da interface gráfica são activadas.

  O accionamento deste botão implica a execução do mesmo procedimento descrito para o botão *Start* da interface gráfica do robô cartesiano (ver capítulo 3).

- ***Stop*:**

  O accionamento deste botão provoca a paragem da aplicação e o sistema é retirado de controlo, ficando inoperante a qualquer ordem de comando executada pelo utilizador através da interface gráfica. São também desactivadas as rotinas de monitorização do sistema. Para retomar a execução da aplicação é necessário accionar de novo o botão *Start*.

  Este botão suspende a execução do controlador, não gera qualquer evento, ou seja, nenhum estado da máquina de estados fica activo, o mesmo acontece com os subsistemas do modelo *Simulink*, uma vez que a aplicação se encontra parada.

- ***Emergency*:**

  Este botão tem a mesma funcionalidade do botão, com o mesmo nome, da interface gráfica de operação do robô cartesiano em modo autónomo (ver capítulo 3). Sendo que a sua principal função é parar a execução da aplicação de uma forma segura. Sempre que o sistema sai de uma situação de emergência, o utilizador está obrigado a executar o procedimento *Zero da Máquina*, accionado para efeito o botão *Reset*, que é o único botão que se encontra activo. De salientar ainda, que a actuação deste botão implica a paragem da operação de telemanipulação.

  O accionamento deste botão implica a execução dos mesmos procedimentos indicados na descrição do botão *Emergency* da interface gráfica do robô cartesiano (ver capítulo 3).

- ***Reset*:**

  Este botão tem a mesma funcionalidade do botão, com o mesmo nome, da interface gráfica de operação do robô cartesiano em modo autónomo (ver capítulo 3). A sua principal função corresponde a activar o procedimento *Zero da Máquina*, ou seja, efectuar o *reset* dos *encoders* do robô cartesiano.

  Assim que o procedimento *Zero da Máquina* esteja concluído ocorre uma sequência de transições de estado, sendo a primeira entre o estado *Zero da Máquina* e o *Posicionamento em Zero*. Quando o procedimento *Zero da Máquina* é concluído a *flag* do *Simulink* que indica o fim deste procedimento é activada, fazendo com que haja mudanças na interface gráfica, nomeadamente o bloqueio e o desbloqueio de certas funcionalidades, sendo uma delas o botão *Run* ficar activo. Uma vez activa a *flag* de fim do procedimento *Zero da Máquina*, na máquina de estados ocorre uma transição do estado *Posicionamento em Zero* para o *Estável*. A máquina de estados permanece no estado *Estável* até que por actuação do botão *Run* haja a transição para o estado *Telemanipulação*.

- ***Run*:**

  Após a conclusão do procedimento *Zero da Máquina* o utilizador possui condições para executar a operação de telemanipulação, nesse sentido, à actuação do botão *Run* estão associados procedimentos que possibilitam a entrada no estado de telemanipulação. Deste modo, ao actuar-se o botão *Run* é inicializado o *Haptic Device*, é efectuada a calibração do mesmo (*reset* dos *encoders*) e é iniciado o escalonador (processo responsável pelo escalonamento das tarefas do *Haptic Device*). Depois de efectuado o procedimento associado à inicialização do *Haptic Device*, é activada a rotina responsável pelo envio das coordenadas de posição e velocidade (x,y,z) do *Haptic Device* para o controlador do robô cartesiano. O accionamento deste botão provoca também o bloqueio e o desbloqueio de certas funcionalidades da interface gráfica.

  O accionamento deste botão origina uma transição no flanco descendente no estado do bloco *Simulink*, que é responsável a ocorrência do evento na máquina de estados *Stateflow*, desta forma dá-se uma transição do estado *Estável* para o *Telemanipulação*. A activação do estado *Telemanipulação* implica a execução do subsistema *Simulink* responsável por fornecer as referências de posição e velocidade do *Phantom Haptic Device* aos controladores de eixo correspondentes do robô cartesiano.

- ***Stop Run*:**

  Este botão permite ao utilizador suspender a tarefa de telemanipulação. Assim, o utilizador tem mais liberdade para executar outras operações que impliquem menor atenção na execução da operação de telemanipulação. Tal facto sucede quando o utilizador se encontra analisar os dados sobre uma operação efectuada, ou se estiver a examinar as condições em que o robô cartesiano se encontra ou ainda se estiver a analisar o resultado da operação de telemanipulação. Todas essas situações implicam uma menor atenção por parte do utilizador em relação à tarefa de telemanipulação o que, por descuido do utilizador, pode levar a uma condição crítica quer para os utilizadores quer para os equipamentos.

O accionamento deste botão provoca uma transição no flanco descendente no estado do bloco *Simulink*, o que origina a ocorrência do evento. Este evento é responsável pela transição na máquina de estados do estado *Telemanipulação* para o *Estável*. Assim, o controlador permanece num estado onde são enviadas referências fixas de posição para cada um dos controladores de eixo do robô cartesiano.

- ***Zero Position*:**

Este botão permite ao utilizador executar o procedimento de posicionamento do robô cartesiano no ponto (0,0,0). O utilizador pode efectuar as operações que pretende, executando a tarefa de telemanipulação, com ampliação ou atenuação da amplitude de movimento. Quando pretender regressar às condições normais (sem factor de escala) efectua o posicionamento na posição (0,0,0). Quando o utilizador estiver a executar a operação de telemanipulação e pretender efectuar o posicionamento em zero, necessita primeiro de parar essa operação através do botão *Stop Run*, de seguida accionar o botão *Zero Position* para efectuar o posicionamento do robô cartesiano no ponto (0,0,0).

O accionamento deste botão provoca uma transição no flanco descendente no estado do bloco *Simulink*, que origina a ocorrência de um evento na máquina de estados. Esse evento é responsável pela transição do estado *Estável* para o *Telemanipulação*. A activação do estado *Telemanipulação* faz com que sejam activados os subsistemas responsáveis pela execução do procedimento de posicionamento do robô cartesiano no ponto (0,0,0).

- ***Exit*:**

  Este botão tem como principais funcionalidades o parar da execução da aplicação de controlo e o fechar da sessão da interface gráfica. Quando este botão é accionado todas as tarefas em execução pelo sistema de controlo são abortadas e as rotinas de monitorização são inibidas.

6.3 Menu *Communications*

Este menu disponibiliza dois tipos de comunicações, suportadas pelo *software* de controlo *xPC Target*, que são a comunicação Porta Série e *TCP/IP*. É necessário o utilizador indicar o endereço de *IP* e o *Port Number* do *Target PC* a que se quer ligar (figura 6.6). Neste menu pode ainda fechar-se a comunicação estabelecida, caso se pretenda estabelecer uma nova.

Figura 6. 6 – Menu *Communications* e Configuração de Comunicação *TCP/IP*

6.4 Conclusões

Neste capítulo foram apresentadas as funcionalidades e as características da interface gráfica do sistema robótico de telemanipulação. O desenvolvimento da interface gráfica teve como principal objectivo, possibilitar ao utilizador a operação do sistema de uma forma fácil e intuitiva. Através da interface gráfica o utilizador pode operar o sistema de uma forma metódica e em condições de segurança, podendo monitorizar em tempo-real o estado operacional do sistema a controlar. A interface gráfica dispõe de uma funcionalidade importante, que corresponde ao armazenamento dos dados de uma determinada trajectória e a possibilidade de visualização desses mesmos dados através de gráficos executados em *Matlab*.

A interface gráfica permite a monitorização do estado operacional do sistema robótico, nomeadamente, o estado dos sinais dos fins-de-curso do robô *slave*, bem como os sinais digitais presentes no armário eléctrico de controlo do robô *slave*. É também possível monitorizar em tempo-real as variáveis de posição do robô *master* e do *slave*.

Na interface gráfica o utilizador pode definir o factor de escala a aplicar numa operação de telemanipulação, este facto é importante na medida em que, existem determinadas tarefas de telemanipulação que requerem uma maior ou menor amplitude de movimentos do sistema a controlar.

A interface gráfica de comando tem acesso directo a uma *DLL* desenvolvida em C^{++}, o que permite aceder às funcionalidades e parâmetros da biblioteca de funções do *Phantom Haptic Device*. Na interface gráfica está também integrada a *xPC COM API*,. Esta *API* permite a utilização de uma estrutura de dados, que possibilita a interacção entra a aplicação gráfica (interface gráfica de comando) e a aplicação de controlo do robô *slave*.

Capítulo 7

Conclusões e Desenvolvimentos Futuros

7.1 Conclusões

Este trabalho foi desenvolvido com o intuito de criar plataformas de desenvolvimento e aprendizagem na área de controlo dos sistemas robóticos, mais concretamente nos sistemas de telemanipulação.

Na primeira fase deste trabalho foi desenvolvida uma aplicação de controlo (*Matlab/Simulink/xPC Target*) e uma interface gráfica (*Microsoft Visual Basic 6.0*), que possibilitam a operação do robô cartesiano de uma forma autónoma. A aplicação de controlo é composta por algoritmos de controlo, geradores de trajectória e uma máquina de estados. Os algoritmos de controlo desenvolvidos permitem o controlo de velocidade do robô cartesiano durante o procedimento de posicionamento no ponto de referência do espaço de trabalho (zero do robô), permitem também o controlo de posição do robô durante a execução das diferentes trajectórias definidas pelo utilizador. A implementação das trajectórias definidas pelo utilizador é efectuada pelos geradores de trajectória desenvolvidos para esse efeito, deste modo, as referências de posição e velocidade são atribuídas a cada controlador de eixo responsável pela execução das respectivas trajectórias.

A interface gráfica desenvolvida permite a qualquer utilizador a operação do robô cartesiano de forma autónoma, sendo dotada das principais funcionalidades que uma consola de um robô industrial possui, ou seja, o utilizador pode movimentar em modo de *"Jog "* cada um dos eixos do robô, pode definir e executar trajectórias entre dois pontos ou entre um número arbitrário de pontos. Os dados da execução de trajectórias são sempre guardados em ficheiros, sendo que no final da trajectória podem ser visualizados e analisados em gráficos executados automaticamente em *Matlab*.

Nesta dissertação foram apresentados e comentados alguns resultados experimentais decorrentes da operação do robô cartesiano de forma autónoma. Esses resultados demonstram a implementação dos geradores de trajectória desenvolvidos e o comportamento dos controladores de posição de cada eixo do robô cartesiano.

A utilização do módulo de controlo *xPC Target* (*Matlab*) permitiu o desenvolvimento de uma arquitectura de controlo de tempo-real, que se verificou implementar com sucesso os algoritmos de controlo e capaz de realizar as tarefas de geração e execução de trajectórias. A estratégia de controlo proposta permite o fácil desenvolvimento dos algoritmos de controlo num *PC* (*Host PC*), que funciona como plataforma de desenvolvimento e de simulação dos mesmos, a interface gráfica de comando também é executada neste *PC*. Depois de desenvolvidos os algoritmos, a aplicação de controlo é compilada e transferida para o *Target PC*, onde é executado o controlador de tempo-real do sistema de controlo do robô cartesiano.

Na segunda fase deste trabalho foi desenvolvida uma arquitectura de controlo para o sistema robótico de telemanipulação (*master/slave*), composto por um *Phantom Haptic Device* (*master*) e pelo robô cartesiano (*slave*). Para o efeito foi desenvolvida uma aplicação de controlo (*Matlab/Simulink/xPC Target*), que é composta por controladores de eixo (posição), responsáveis pela execução das trajectórias, e por uma máquina de estado responsável pela gestão do sistema de eventos. Foi também desenvolvida uma interface gráfica de comando (*Microsoft Visual Studio C#*), que permite ao utilizador a operação do sistema robótico de telemanipulação. Através da interface gráfica o utilizador pode efectuar o posicionamento de ambos os sistemas nos pontos de referência do espaço de trabalho, pode também efectuar tarefas de telemanipulação,

cujas trajectórias são gravadas em ficheiro e que podem ser visualizadas e analisadas em gráficos. A interface gráfica faz parte de uma aplicação desenvolvida em *Microsoft Visual Studio*, que também é composta por uma *DLL* correspondente a uma estrutura de dados, que permite a utilização de algumas funções da biblioteca de funções do *Phantom Haptic Device*, possibilitando desta forma o acesso a parâmetros como a posição e velocidade de cada eixo.

De momento o sistema robótico de telemanipulação desenvolvido apresenta uma estratégia de controlo unilateral, cujo principal objectivo corresponde ao seguimento de trajectória do robô *slave* em relação ao robô *master*. Nesse sentido foi desenvolvida uma arquitectura de controlo composta por dois sistemas de controlo (*PCs*), o *Host PC* onde é executada, através do sistema operativo *Windows*, a interface gráfica de comando e os algoritmos de controlo do *Phantom Haptic Device* (*master*), no *Target PC* é executado em tempo-real o controlador do robô cartesiano. Sendo que, a interligação entre os dois sistemas de controlo é efectuada por um canal de comunicação dedicado (*Ethernet* – cabo *crossover*).

7.2 Desenvolvimentos Futuros

O trabalho desenvolvido tem como objectivos a inovação, a aprendizagem e a implementação de novas tecnologias. Nesse sentido várias são as melhorias e desenvolvimentos futuros que podem acrescentar valor ao trabalho efectuado até ao momento.

Assim, na primeira fase do trabalho que consistiu em criar um sistema de comando e controlo do robô cartesiano e que tem como objectivo o desenvolvimento, o teste e a validação de algoritmos de controlo, podem ser efectuadas algumas melhorias, das quais se destacam as seguintes:

- Desenvolvimento de algoritmos de controlo robustos, nomeadamente controladores de ordem fraccionária ou outro tipo de controladores, que permitam melhorar as performances do sistema de controlo existente e que possibilitem o estudo e a compreensão das teorias de controlo a utilizar;

- A migração da interface gráfica de comando para uma plataforma de desenvolvimento, que permita utilizar novas técnicas de programação e que seja compatível com as novas tecnologias. A actual interface gráfica foi desenvolvida na plataforma *Microsoft Visual Basic 6.0*, por esta razão se recomenda a sua reformulação numa plataforma mais actual, como por exemplo a *Microsoft Visual Studio C#.*

Na segunda fase do trabalho, que consistiu no desenvolvimento de uma arquitectura de controlo de um sistema robótico do tipo *master/slave*, podem ser acrescentadas melhorias e funcionalidades ao sistema, tais como:

- Desenvolvimento de um controlador em tempo-real para o *Phantom Haptic Device* de forma a optimizar as funcionalidades do dispositivo, sendo que, deste modo, melhoraria também as performances (determinismo na cadência de

transferência de dados entre os dois sistemas) do sistema robótico do tipo *master/slave*;

- Integração do transdutor de força no sistema robótico (no órgão terminal do robô cartesiano) e consequente integração do *driver* de tempo-real para *Matlab* no controlador de tempo-real do sistema. Após a integração do *feedback* de força como uma componente do sistema de controlo, será necessário o desenvolvimento de uma estratégia de controlo que permita implementar um controlador de força/impedância;

- A interligação dos dois sistemas robóticos, *master*/*slave* e correspondentes controladores, por meio de comunicação não dedicada (ex. Internet). Essa tarefa implicará o estudo da melhor solução, uma vez que a cadência, atrasos e perdas de informação características de uma comunicação não dedicada, poderão comprometer o objectivo global do sistema robótico de telemanipulação, que corresponde a efectuar um controlo de posição e força com a precisão necessária para o desenvolvimento de tarefas complexas e minuciosas;

- Depois de implementada uma interligação não dedicada entre os dois sistemas de controlo, torna-se necessário o desenvolvimento de controladores e estratégias de controlo robustas de forma a atenuar as perturbações associadas ao meio de comunicação utilizado.

Referências

Andreu, D., Fraisse, P., Segovia De Los Rios, J.A. (2004). "Teleoperation over an IP network: from control to architectural considerations", Control, Automation, Robotics and Vision Conference (ICARCV 2004 8th), China, Volume 1, pp. 765 - 770.

Aström, K. J. and Wittenmark, B. (1997). "Computer-controlled systems theory and design", Upper Saddle River, NJ, Prentice-Hall.

Baumann, R. and Clavel, R. (1998). "Haptic interface for virtual reality based minimally invasive surgery simulation", Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Louvain, Belgium, Volume 1, pp. 381-386.

Bisták, P. and Žáková, K. (2003). "Organising Tele-Experiments for Control Education", 11th Mediterranean Conference on Control and Automation, Rhodes, Grécia.

Cavusoglu, M. C., *et al.* (2002). "Design of bilateral teleoperation controllers for haptic exploration and telemanipulation of soft environments", IEEE Transactions on Robotics and Automation, Volume 18, pp. 641- 647.

Filippi, H. (2007). "Wireless teleoperation of robotic arms", Master Thesis, Department of Space Science, Kiruna, Sweden.

Gough, John (2001). "Compiling for the .NET Common Language Runtime (CLR)". Prentice Hall PTR.

Hercog, D., Gergic, B., Matko, V. (2005). "Remote Lab for Electric Drives", IEEE ISIE 2005, Dubrovnik, Croatia, Volume 4, pp. 1685- 1690.

Hirzinger, G., *et al.* (1997). "Teleoperating space robots. Impact for the design of industrial robots", IEEE ISIE 1997, Guimarães, Portugal, Volume 1, pp. 250-256.

K. S. Fu, R. C. Gonzalez, C. S. G. Lee (1987). "Robotics: Control, Sensing, Vision, and Intelligence", McGraw-Hill International Editions.

Kim, C.H., Kim, S.J., Kim, D.K. (2004). "RTAI Based Real-Time Control of Robotic Wheelchair", 6th RTL Workshop.

Lopes, A. M. (1994). "Análise cinemática e planeamento de trajectórias para um robot industrial", Tese de Mestrado, Porto.

Lopes, A.M., Abreu, P., Cardoso, M. (2007). "A Master/Slave Telemanipulation Robotic System", REV Conference, Porto, Portugal.

Low, K.H., Wang, H., Wang, M. Y. (2005). "On the development of a real time control system by using xPC Target: solution to robotic system control", IEEE International Conference on Automation Science and Engineering, Edmonton, Canada, Agosto 2005, pp. 345-350.

Nuño Ortega, E. e Basáñez, L. (2006). "Bilateral haptic guided Robot teleoperation via packet switched networks using wave variables with impedance adaptation". A: 2es Jornades de Recerca en Automàtica, Visió i Robòtica: AVR'06, pp. 211-218.

Ogata, K. (1997). "Modern control engineering", Upper Saddle River, NJ, Prentice Hall International.

Okamura, A.M. (2004). "Methods for haptic feedback in teleoperated robot-assisted surgery", Industrial Robot, Volume 31, Number 6, pp. 499–508.

Reiley, C. E., *et al.* (2008).“Effects of visual force feedback on robot-assisted surgical task performance”, The Journal of Thoracic and Cardiovascular Surgery, Volume 135, Number 1, pp. 196-202.

Richard P. Paul (1982). “Robot Manipulators: Mathematics, Programming, and Control”, The MIT Press series in artificial intelligence”.

Rundqwist, Lars (1991). “Anti-Reset Windup for PID Controllers,” Ph.D. Thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Sitti, M. and Hashimoto, H. (2003). “Teleoperated touch feedback from the surfaces at the nanoscale: modeling and experiments”, IEEE/ASME Transactions on Mechatronics, Volume 8, Number 2, pp.287-298.

Tavakoli, M., *et al.* (2008). “Haptics for Teleoperated Surgical Robotic Systems”, New Frontiers in Robotics, Volume 1.

Thompson, R. L. (2001).“Integration of visual and haptic feedback for teleoperation”, PhD. Thesis, Trinity College, University of Oxford.

Vukobratovic, M. and Kircanski, M. (1986). “Kinematics and Trajectory Synthesis of Manipulation Robots”, Berlin, Springer – Verlag.

Anexo A

Manual de Utilização do Robô Cartesiano

A interface gráfica de comando do robô cartesiano (aplicação *Visual Basic* – figura 1) pode ser executada no modo de *debbug* através do software *Microsoft Visual Basic*, sendo necessário abrir o ficheiro **Consola_en.vbp** (C:\Documents and Settings\Mario\My Documents\control_eixos), ou então pode ser executada através do executável **3DOF Console.exe** que está no mesmo directório.

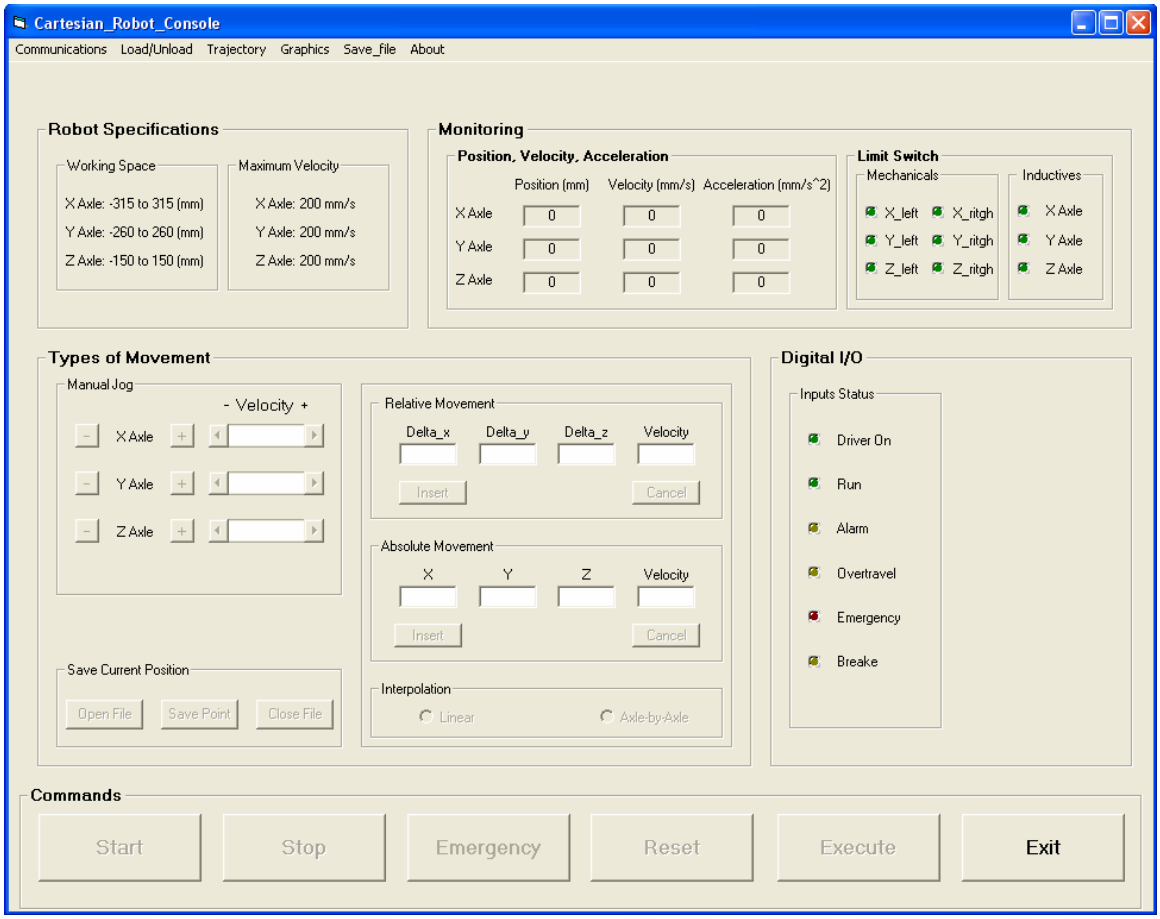


Figura 1 – Interface Gráfica de Comando do Robô Cartesiano

O utilizador dispõe de dois modos de operação do sistema: modo de "*teach*" (ecrã inicial e as suas funcionalidades); e o modo de execução de trajectórias multi-ponto.

Modo *Teach*

Neste modo o utilizador pode efectuar vários tipos de movimento entre dois pontos no espaço cartesiano, pode também guardar os pontos visitados e analisar os gráficos resultantes dos movimentos efectuados. Este modo permite ao utilizador ambientar-se com as ferramentas da interface gráfica, bem como conhecer as características do robô cartesiano. De seguida são apresentados os procedimentos possíveis de serem executados através da interface gráfica.

1. **Estabelecer ligação:** Menu Communications → Establish Connection → TCP/IP (figura 2). Através deste menu é estabelecida a ligação com o controlador do robô cartesiano (*Target PC*).

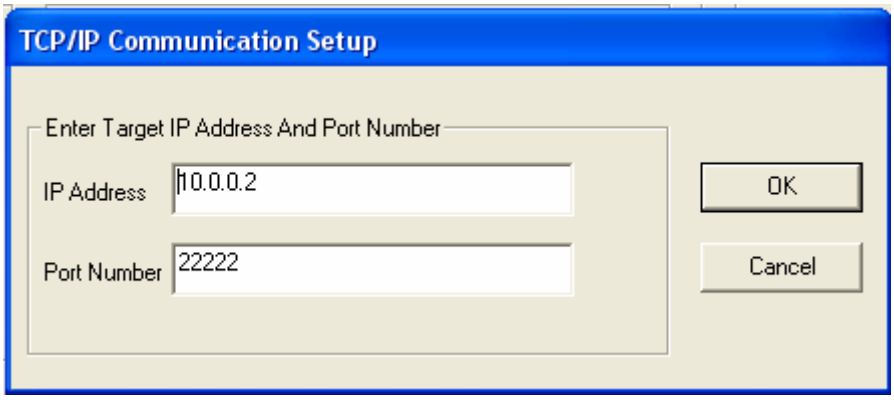


Figura 2 – Configuração da Comunicação

2. **Carregar o ficheiro de aplicação de controlo:** Menu Load →Load (figura 3).

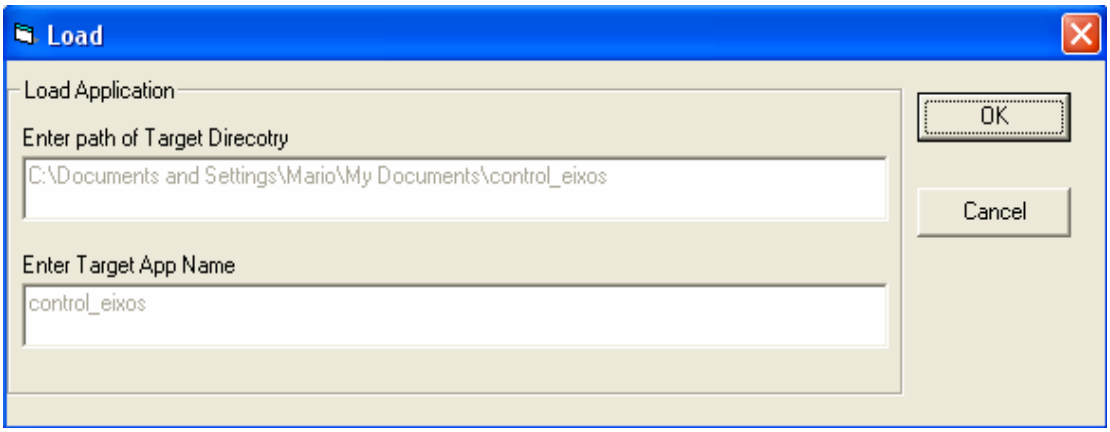


Figura 3 – Load da Aplicação de Controlo

3. **Accionar o botão *Start*:** Sempre que se inicia uma sessão, o botão *Start* do painel *Commands* (figura 4) fica activo. **Antes de accionar este botão é necessário que no armário eléctrico se accione a botoneira *"Driver On"* e o correspondente sinal fique activo**. Quando o sinal *"Driver On"* estiver activo, deve carregar-se no botão *Start* para se iniciar a sessão.

Figura 4 – Botões de Comando da Interface Gráfica

Em consequência do accionamento do botão *Start*, uma mensagem de aviso (figura 5) aparece no ecrã, tendo como objectivo alertar o utilizador para o facto de o primeiro procedimento a ser efectuado ser o zero da máquina.

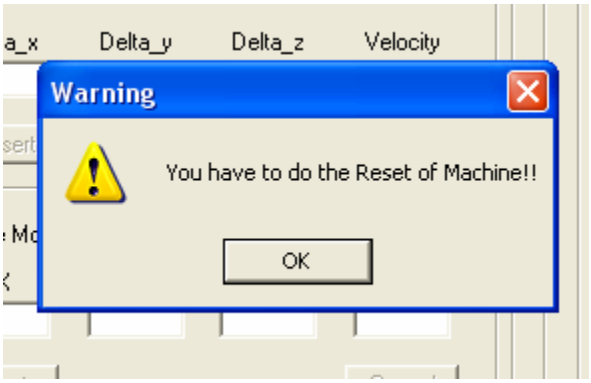


Figura 5 – Aviso no Início da Aplicação

4. **Accionar o botão *Reset*:** Ao ser accionado este botão o procedimento de zero da máquina é executado. Durante este procedimento as restantes funcionalidades ficam desactivadas, somente o botão *Emergency* pode ser actuado (figura 4). No fim deste procedimento são activadas as restantes funcionalidades correspondentes à definição dos diversos tipos de movimento.

5. **Tipos de Movimento**

Existem três tipos de movimento: *Manual Jog*, *Relative Movement* e *Absolute Movement*.

1. ***Manual Jog*:** O utilizador pode efectuar o movimento manual de um eixo de cada vez. Para isso, antes de iniciar o movimento, necessita de definir um patamar de velocidade através dos *slider bars* (Vmax = 20 mm/s e Vmin = 5mm/s). Após definir a velocidade, o utilizador pode movimentar os eixos em ambas as direcções, accionando os botões (+ ou **–**) associados a cada eixo (figura 6).

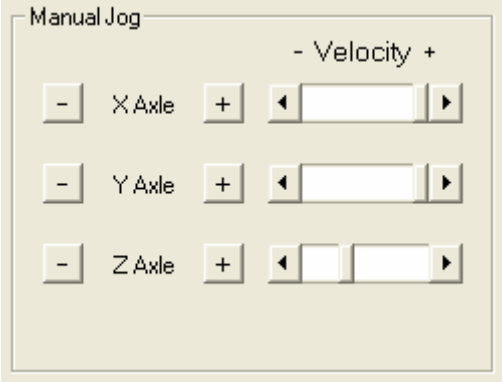


Figura 6 – Painel Movimento Manual

2. *Relative Movement*: O utilizador pode efectuar um movimento linear relativo entre dois pontos. Para isso é necessário especificar os incrementos de posição que vão ser somados à posição efectiva, bem como a velocidade média com que o robô cartesiano se vai deslocar. Para inserir estes dados é necessário carregar no botão *Insert*, uma vez que por uma questão de segurança os seus campos se encontram bloqueados (figura 7).

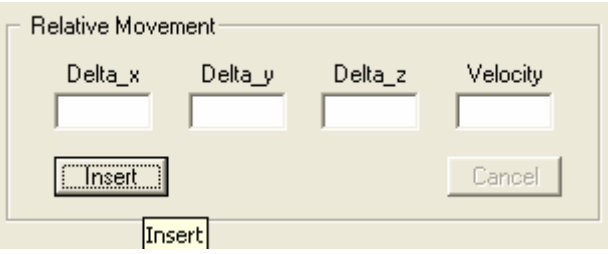


Figura 7 – Painel de Definição de Coordenadas Relativas

3. *Absolute Movement*: Para o robô cartesiano efectuar um movimento linear absoluto entre dois pontos é necessário a especificação das coordenadas de posição absoluta e a velocidade com que o robô vai efectuar o movimento. Tal como acontece com o painel de movimento relativo, é necessário carregar no botão *Insert* para que os campos de inserção de dados fiquem activos (figura 8).

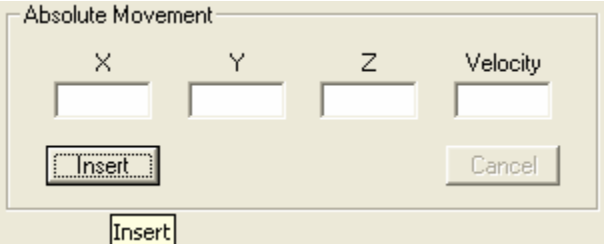


Figura 8 – Painel de Definição de Coordenadas Absolutas

6. **Tipos de Interpolação:** Na execução do movimento entre dois pontos, tanto no movimento relativo como no absoluto, é necessária a indicação de qual o tipo de interpolação que se pretende utilizar (figura 9). Antes de dar ordem de execução de movimento através do botão *Execute* (painel *Commands* – figura 4), é necessário especificar o tipo de interpolação a ser utilizado na execução da trajectória, caso contrário ocorrerá uma mensagem de erro.

Figura 9 – Painel de Definição do Tipo de Interpolação

1. **Interpolação Linear:** Ao ser seleccionado este tipo de interpolação, o movimento resultante será um movimento rectilíneo no espaço tridimensional (3D), ou seja, ambos os eixos se movem ao mesmo tempo.

2. **Interpolação *Axle-by-Axle*:** Ao ser seleccionado este tipo de interpolação o movimento resultante será um movimento rectilíneo de cada eixo, sendo que neste caso o movimento é efectuado por um eixo de cada vez.

7. **Executar Trajectória:** Para se executar uma trajectória definida é necessário accionar o botão *Execute* (painel *Commands* – figura 10). Antes de executar a trajectória o utilizador deve certificar-se que especificou correctamente as coordenadas de posição e velocidade, bem como o tipo de interpolação.

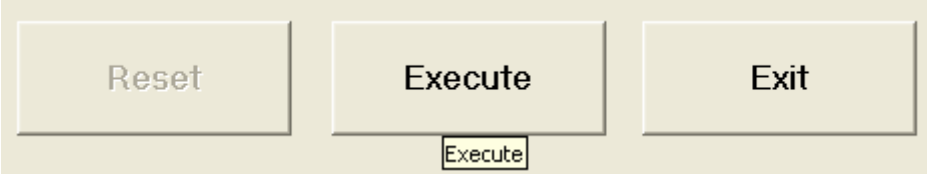


Figura 10 – Botão de Execução de Trajectória

8. **Casos de Emergência:** Durante a execução de um movimento, caso o utilizador entenda que está perante uma situação de emergência e pretenda suspender o movimento, tem duas possibilidades de o fazer, através do botão *Emergency* (painel *Commands*) da interface gráfica ou através da botoneira de emergência presente no armário eléctrico.

   No caso de o utilizador accionar o botão *Emergency*, o movimento que estava a ser executado é automaticamente cancelado. As funcionalidades da interface gráfica ficam desactivas, sendo o utilizador avisado que esta perante uma paragem de emergência e que deve solucionar a causa que lhe deu origem. Quando o utilizador se certificar que todas as questões de segurança estão asseguradas, deve accionar o botão *Emergency Out* (painel *Commands*), esse botão surge no lugar do botão *Emergency*. Após a confirmação do estado operacional do sistema (actuação do botão *Emergency Out*), é necessário efectuar o procedimento zero da máquina accionado o botão *Reset* (o único botão activo da interface gráfica).

   No caso de ser accionada a botoneira de emergência as funcionalidades da interface gráfica são desactivas, o utilizador é avisado que ocorreu uma situação de emergência por actuação da botoneira de emergência, sendo obrigado a fechar a aplicação e retomar a sua execução somente quando estiverem asseguradas todas as condições de segurança.

   No caso de, durante um determinado movimento, um dos fins-de-curso ser actuado o sistema pára de imediato. As funcionalidades da interface ficam desactivas, o utilizador é avisado de que o sistema parou devido à actuação de um dos fins-de-curso, sendo obrigado a sair da aplicação. Após sair da aplicação o utilizador deve,

através dos *drivers* de potência (armário eléctrico) entrar no modo de *Jog* e mover o robô cartesiano para que os fins-de-curso que estejam actuados fiquem desactivados. De seguida deve, no armário eléctrico, efectuar o *reset* ao *Overtravel*. Depois do procedimento descrito anteriormente, o utilizador pode iniciar novamente a aplicação.

9. **Visualização de gráficos:** No fim da execução de um movimento linear entre dois pontos (relativo ou absoluto), o utilizador tem a possibilidade de visualizar os gráficos de posição, velocidade e acção de controlo, para cada um dos eixos. Para visualizar estes gráficos o utilizador deve aceder ao menu *Graphics* (figura 11).

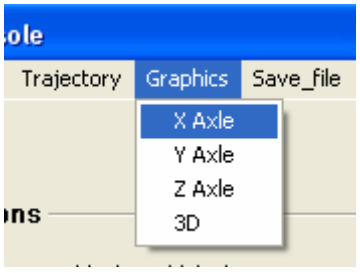


Figura 11 – Menu Graphics

Quando o utilizador carregar numa das opções que aparecem na figura 11, é executado um ficheiro executável do *Matlab*. Por esta razão os gráficos demorarão algum tempo a aparecer no ecrã.

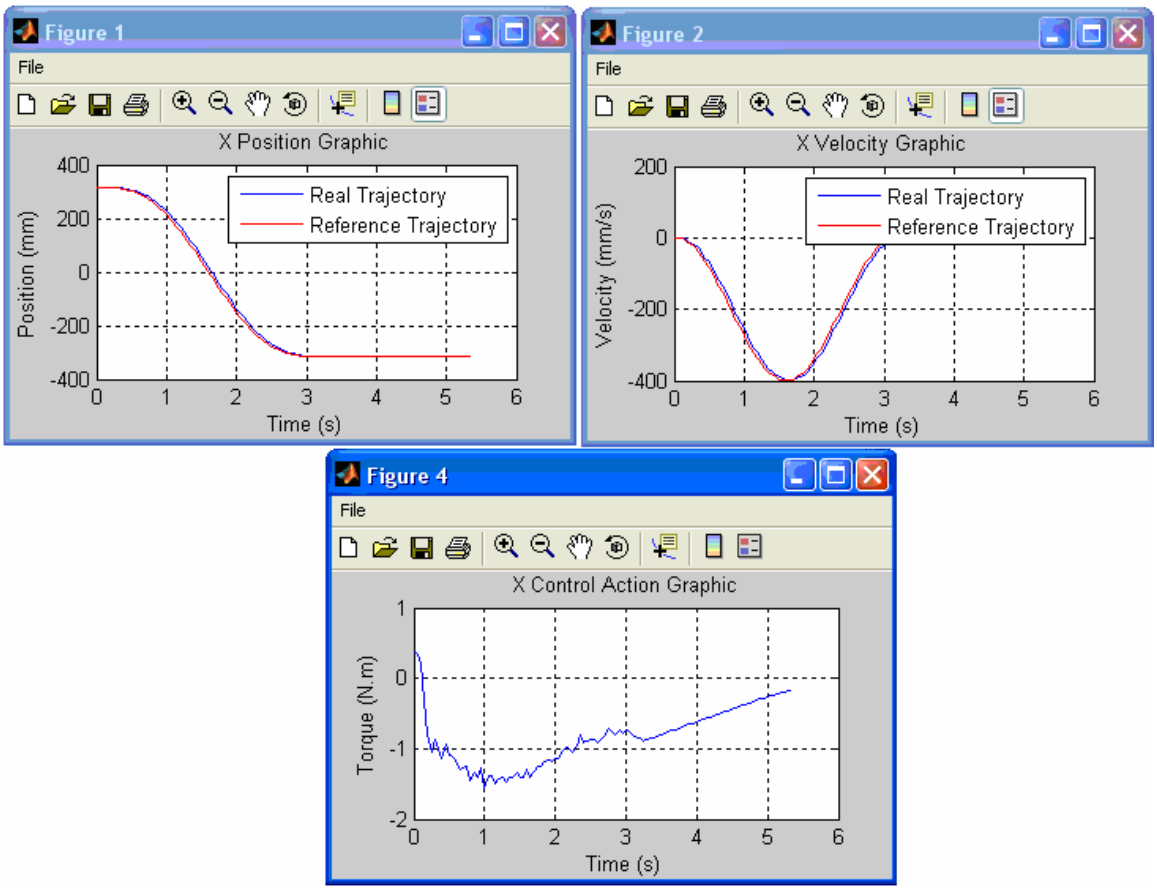


Figura 12 – Gráficos Resultantes da Execução de Trajectória

Após a visualização dos gráficos (figura 12) o utilizador pode gravar em ficheiros de *backup* a informação visualizada, basta para isso aceder ao menu *Save_file* (figura 11). O utilizador deve efectuar este procedimento sempre que pretenda guardar os dados de determinada trajectória, caso contrário, quando uma nova trajectória é executada os ficheiros que armazenam a informação são automaticamente reescritos. Os novos ficheiros gerados são armazenados no directório onde se encontra a aplicação, na subdirectoria *backup_files*.

10. **Guardar pontos visitados:** Durante a execução em modo de *Teach* o utilizador pode guardar os pontos visitados. Através do painel *Save Current Position* (figura 13), o utilizador pode abrir um ficheiro utilizando o botão *Open File*, sendo que à medida que vai executando determinadas trajectórias pode guardar os pontos visitados através do botão *Save Point*. Sempre que o utilizador pretenda gravar um ponto tem de

definir a velocidade que deve ficar associada a esse ponto (figura 14). Deste modo, quando o utilizador carrega no botão *Save File* uma nova janela aparece no ecrã, onde é especificado o valor de velocidade pretendido. Quando o utilizador pretende fechar o ficheiro, para que este possa ser utilizado, tem obrigatoriamente de actuar o botão *Close File*.

Caso o utilizador pretenda abrir um ficheiro já existente, a gravação de novos pontos é feita (pontos acrescentados) no fim desse ficheiro, sendo que os restantes continuam armazenados. Isto permite dar continuidade a ficheiros de pontos utilizados anteriormente. Caso o utilizador pretenda criar um novo ficheiro terá de abrir um ficheiro com um nome que não exista nesse directório.

Os ficheiros que contêm os pontos visitados podem ser utilizados na execução de trajectórias multi-ponto.

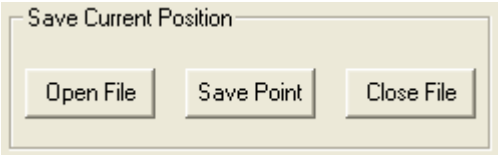


Figura 13 – Painel para guardar os Pontos Visitados

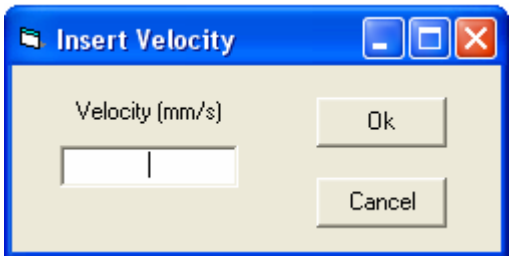


Figura 14 – Janela de Definição de Velocidade

11. **Accionar o botão *Stop*:** Ao accionar este botão o utilizador pára a execução da aplicação, sendo que o controlador do sistema deixa de ser executado.

Trajectórias multi-ponto

O utilizador pode aceder, através do menu *Trajectory* (figura 1), às funcionalidades que lhe permitem executar um conjunto de trajectórias multi-ponto. Sendo que, o utilizador dispõe de dois modos de definição de trajectória multi-ponto: *Insert Points* e o *Load From File*.

1. ***Insert Points*:** Se o utilizador seleccionar esta opção, uma nova janela aparece no ecrã (figura 15).

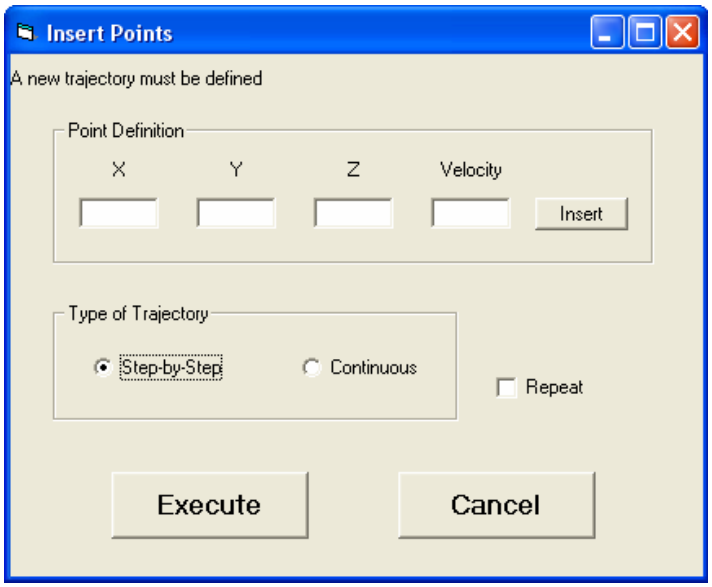


Figura 15 – Janela de Definição da Trajectória Multi-ponto (*Insert Points*)

Através desta funcionalidade o utilizador pode definir um conjunto de pontos no painel *Insert Points*, sendo estes gravados para um ficheiro com o nome *default* sempre que o utilizador carrega no botão *Insert*. Após o utilizador definir todos os pontos (no mínimo está obrigado a inserir três pontos), deve seleccionar o tipo de trajectória que pretende efectuar, sendo que existem duas opções: *Step-by-Step* e a *Continuous*. Se o utilizador escolher a opção *Step-by-Step* o movimento efectuado corresponde à execução da trajectória ponto a ponto, o que significa que durante a execução o robô pára nos pontos definidos. Se o utilizador optar pela opção *Continuous*, a trajectória é executada num modo contínuo, ou seja o robô executa a trajectória sem parar nos pontos.

Neste menu, o utilizador tem ainda a possibilidade de optar por executar a trajectória multi-ponto em modo repetitivo. Para isso deve seleccionar a opção *Repeat* antes da execução da trajectória.

Após fornecer todos dados necessários à execução da trajectória multi-ponto, basta carregar no botão *Execute* para que seja dada ordem de execução de movimento. Assim, o robô é automaticamente posicionado no primeiro ponto definido pelo utilizador, de seguida toda a trajectória é executada, sendo que no final o robô retorna automaticamente ao ponto original.

2. ***Load From File*:** Se o utilizador optar por definir uma trajectória multi-ponto através deste método, uma nova janela aparece no ecrã (figura 16).

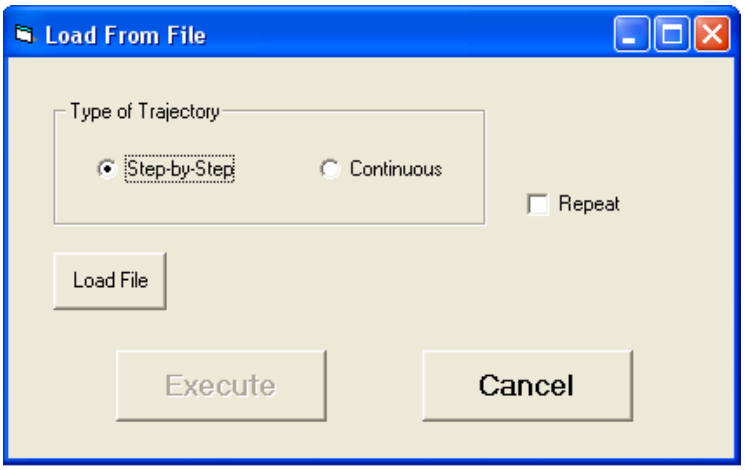


Figura 16 – Janela de Definição da Trajectória Multi-ponto (*Load From File*)

Esta funcionalidade permite ao utilizador carregar um ficheiro onde estejam definidos os pontos que compõem a trajectória a ser executada. Basta para isso carregar no botão *Load File* e escolher o ficheiro pretendido. Após carregar o ficheiro o utilizador tem de seleccionar o tipo de trajectória que pretende executar. Tal como acontecia na opção anterior existem duas opções: *Step-by-Step* e *Continuous*. O utilizador pode ainda optar por executar a trajectória em modo repetitivo. Quando o utilizador fornecer todos os dados necessários pode executar a trajectória accionado o botão *Execute*. O procedimento de execução de trajectória é o mesmo que o descrito para opção anterior.

Anexo B

Manual de Utilização do Sistema Robótico de Telemanipulação

A interface gráfica de comando do sistema robótico de telemanipulação (figura 1) pode ser executado em modo de *debbug* através do software *Microsoft Visual Studio* (*C#*), sendo necessário abrir o ficheiro **Phantom_Interface.sln** (C:\Documents and Settings\Owner\My Documents\C#\Phantom_Interface), ou então pode ser executada através do executável **Phantom_Interface.exe** que esta no mesmo directório.

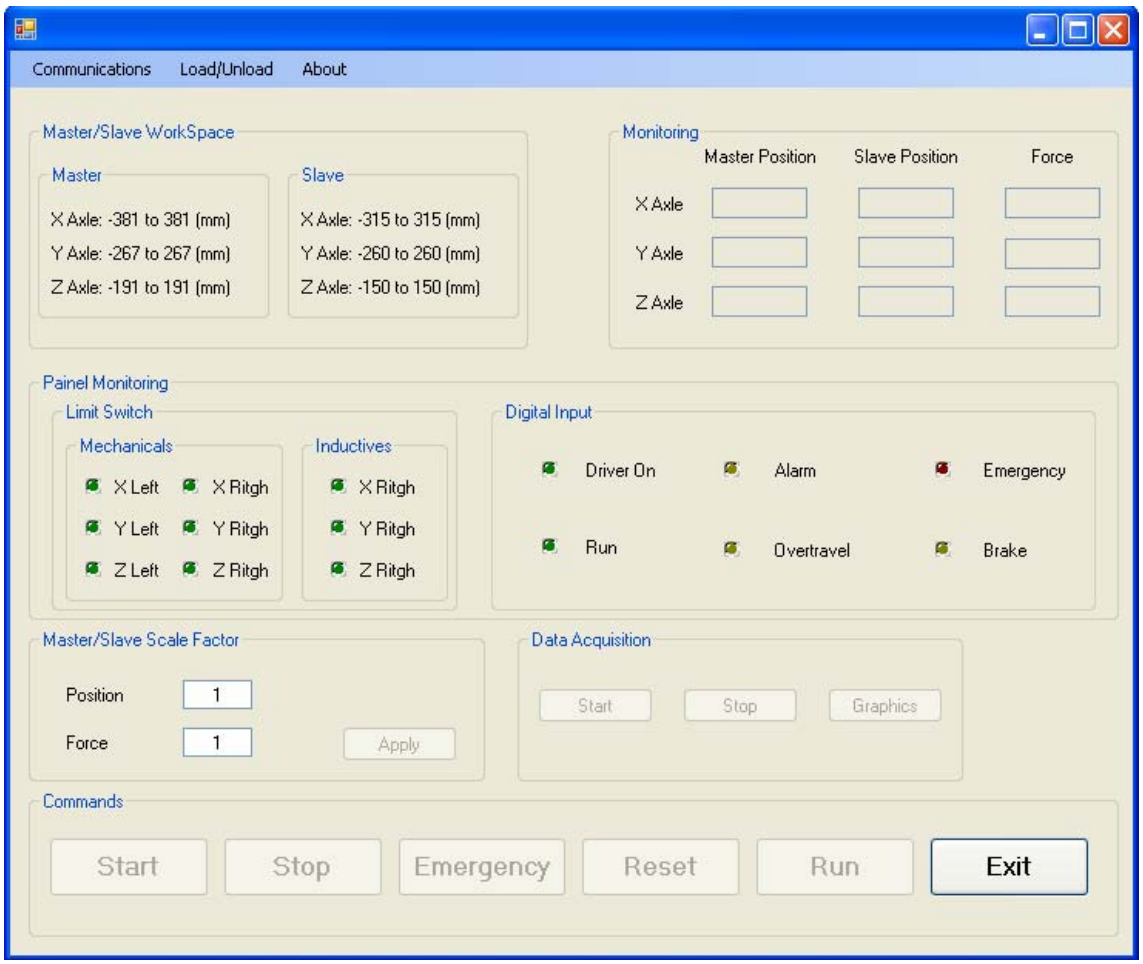


Figura 1 – Interface Gráfica de Comando do Sistema Robótico de Telemanipulação

Através desta interface gráfica o utilizador pode operar o sistema robótico de telemanipulação. Dispõe de um conjunto de botões para executar comandos, um conjunto de sinalizadores com informação do robô *slave* (cartesiano), informação do espaço de trabalho dos robôs *master* e *slave* e um conjunto de caixas de texto onde são apresentadas as coordenadas de posição de ambos os robôs.

De seguida, são apresentados os passos necessários para a operação da interface gráfica.

1. **Estabelecer uma ligação:** Menu Communications → Establish Connection → TCP/IP (figura 2). Através deste menu é estabelecida a ligação com o *Target PC*.

Figura 2 – Menu Communications

2. **Carregar o ficheiro de aplicação de controlo:** Menu Load →Load (figura 3). O ficheiro correspondente à aplicação de controlo é automaticamente carregado e é efectuado o *download* do mesmo para o *Target PC*.

3. **Accionamento do botão *Start*:** Para dar início a uma sessão o utilizador deve carregar no botão *Start*. Ao ser accionado este botão certas funcionalidades da interface gráfica ficam operacionais, tais como os sinalizadores do robô cartesiano e as caixas de texto que apresentam as coordenadas de posição dos robôs *master* e *slave*. O único botão que fica activo é o *Reset*.

4. **Accionamento do botão *Reset*:** Ao ser accionado este botão é executado o procedimento correspondente ao *reset* dos *encoders* do robô cartesiano (*slave*) e do *Phantom Haptic Device* (*master*), seguido do posicionamento do *slave* no ponto (0,0,0). Desta forma, sempre que é iniciada ou reiniciada uma sessão é efectuado esse procedimento, com o intuito de posicionar ambos os robôs nos pontos de referência do seu espaço de trabalho.

Quando este botão é accionado as funcionalidades da interface gráfica ficam bloqueadas até que o procedimento referido anteriormente seja concluído. Depois de terminado esse procedimento, as restantes funcionalidades da interface gráfica ficam desbloqueadas.

5. **Accionamento do botão *Run*:** Para iniciar uma tarefa de telemanipulação, o utilizador tem sempre de carregar no botão *Run*. Assim, quando este botão é accionado o sistema entra no modo de controlo por telemanipulação.

   Antes de carregar no botão *Run* para iniciar uma tarefa de telemanipulação, o utilizador deve introduzir e confirmar os factores de escala (posição e força) no painel *Master/Slave Scale Factor* (figura 3). Deste modo, sempre que se inicia uma sessão de telemanipulação os factores de escala estão definidos, evitando assim que por esquecimento se utilizem factores de escala desajustados à tarefa a realizar.

6. **Definição do factor de escala:** Se o utilizador pretender alterar o factor de escala de posição, deve definir o valor no painel *Master/Slave Scale Factor* (figura 3), sendo que de seguida deve confirmar a introdução desse parâmetro através do botão *Apply*. Assim, é possível ajustar o espaço de trabalho do robô *slave* em relação ao robô *master*, bem como efectuar tarefas de telemanipulação mais precisas.

   O factor de escala de força pode ser definido mas não é usado, uma vez que ainda não se encontra implementado o *feedback* de força.

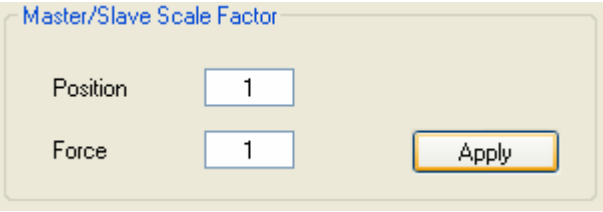


Figura 3 – Definição do Factor de Escala

7. **Casos de Emergência (Accionamento do botão *Emergency*):** Durante a execução de um movimento, caso o utilizador entenda que está perante uma situação de emergência e pretenda suspender o movimento, tem duas possibilidades de o fazer:

através do botão *Emergency* (painel *Commands*) da interface gráfica ou através da botoneira de emergência presente no armário eléctrico.

No caso do utilizador accionar o botão *Emergency*, o movimento que estava a ser executado é automaticamente cancelado, as funcionalidades da interface gráfica ficam desactivas, sendo o utilizador avisado que esta perante uma paragem de emergência e que deve solucionar a causa que lhe deu origem. Quando o utilizador se certificar que todas as questões de segurança estão asseguradas, deve accionar o botão *Emergency Out* (painel *Commands*), esse botão surge no lugar do botão *Emergency*. Após a confirmação do estado operacional do sistema (actuação do botão *Emergency Out*), é necessário efectuar o procedimento zero da máquina accionado o botão *Reset* (o único botão activo da interface gráfica).

No caso de ser accionada a botoneira de emergência as funcionalidades da interface gráfica são desactivas, o utilizador é avisado que ocorreu uma situação de emergência por actuação da botoneira de emergência, sendo obrigado a fechar a aplicação e retomar a sua execução somente quando estiverem asseguradas todas as condições de segurança.

No caso de durante um determinado movimento um dos fins-de-curso ser actuado, o sistema pára de imediato. As funcionalidades da interface ficam desactivadas, o utilizador é avisado de que o sistema parou devido à actuação de um dos fins-de-curso, sendo obrigado a sair da aplicação. Após sair da aplicação o utilizador deve, através dos *drivers* de potência (armário eléctrico) entrar no modo de *Jog* e mover o robô cartesiano para que os fins-de-curso que estejam actuados fiquem desactivados. De seguida deve, no armário eléctrico, efectuar o *reset* ao *Overtravel*. Depois do procedimento descrito anteriormente, o utilizador pode iniciar novamente a aplicação.

8. **Aquisição e visualização de dados:** Através do painel *Data Acquisition* (figura 4), o utilizador pode iniciar o processo de gravação dos dados referentes às trajectórias executadas por ambos os robôs (*master* e *salve*) durante a execução de uma tarefa de telemanipulação.

Para iniciar a aquisição de dados o utilizador deve carregar no botão *Start* do painel *Data Acquisition*. Deste modo, os dados são gravados durante a execução da trajectória, para ficheiros no *Target PC*. Quando o utilizador pretender terminar o processo de aquisição de dados deve carregar no botão *Stop*, o que implica o fim do *log* de dados e a automática transferência desses dados para ficheiros no *Host PC* (o processo de transferência de dados entre os dois *PCs* pode demorar algum tempo, dependendo do tamanho dos ficheiros).

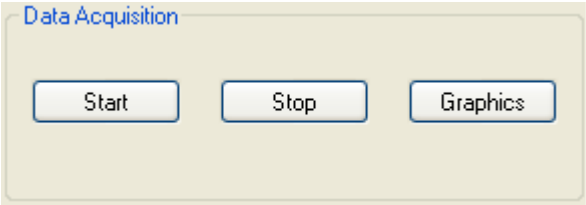


Figura 4 – Painel de Aquisição de Dados

Depois de adquiridos os dados relativos à execução de uma tarefa de telemanipulação, o utilizador pode visualizar esses dados através de gráficos executados automaticamente em *Matlab* (figura 5). Basta para isso carregar no botão *Graphics*. O processo de execução dos gráficos pode ser lento, pelo que é aconselhável que o utilizador faça uma pausa na execução da tarefa de telemanipulação.

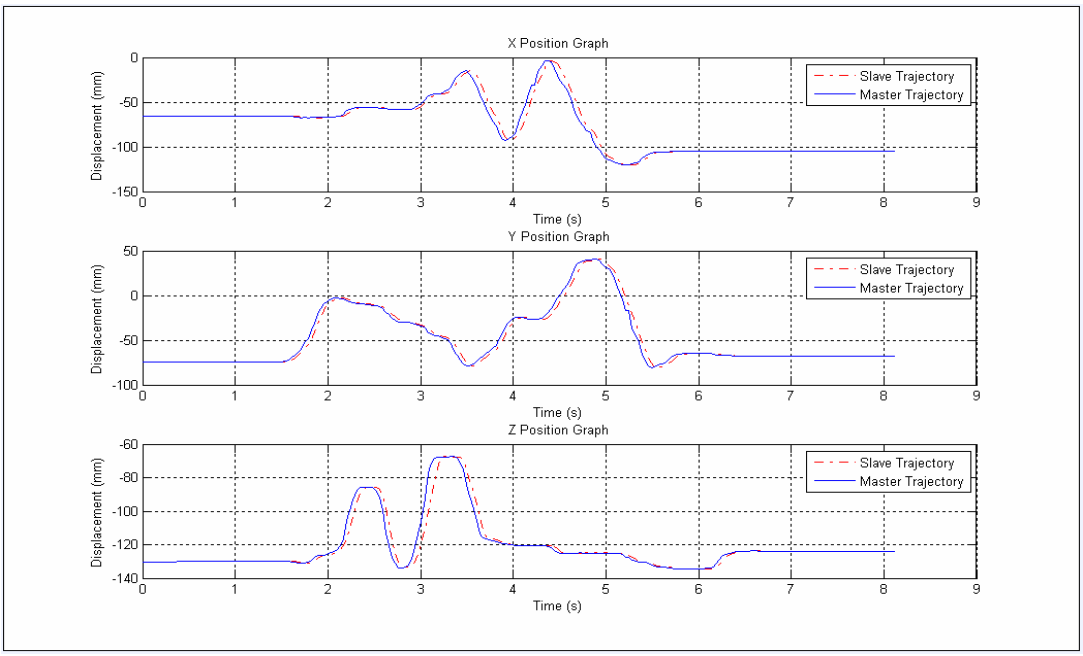


Figura 5 – Gráficos de Posição dos Robôs *Master* e *Slave*

9. **Accionamento do botão *Stop*:** Ao accionar este botão o utilizador pára a execução da aplicação, sendo que o controlador do sistema deixa de ser executado.