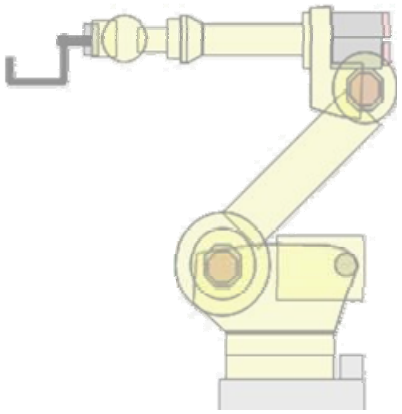
Universidade de Aveiro
2007

Departamento de Engenharia
Mecânica

Rui Cancela

Extensão e flexibilização da interface de controlo de um manipulador robótico FANUC



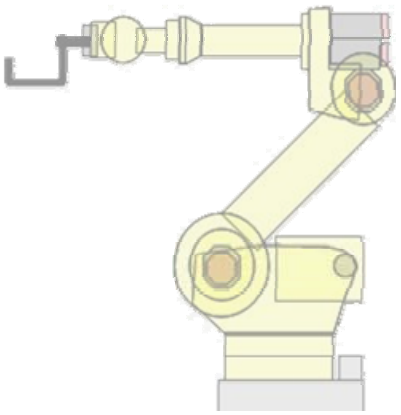
Universidade de Aveiro
2007

Departamento de Engenharia Mecânica

Rui Cancela

Extensão e flexibilização da interface de controlo de um manipulador robótico FANUC

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Prof. Dr. Vítor Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro



júri

presidente

Prof. Dr. José Joaquim de Almeida Grácio
Professor Catedrático | Dept. de Engenharia Mecânica | Universidade de Aveiro

Prof. Dr. Fernando Macedo Ribeiro
Professor Associado | Dept. de Electrónica Industrial | Universidade do Minho

Prof. Dr. Vítor Manuel Ferreira dos Santos
Professor Associado | Dept. de Engenharia Mecânica | Universidade de Aveiro

agradecimentos

Ao Prof. Vítor Santos pela orientação, apoio e inspiração não só neste trabalho mas nos últimos anos ligados à robótica.
Ao Miguel e ao David por todo o apoio e imprescindível ajuda nas longas noites.
Ao Germano por todas as horas passadas a conversar sobre robótica.
À Carla pela imensa paciência e compreensão.
E aos meus pais por terem tornado possível ter chegado até aqui.

À Motofil Robotics, S.A. na pessoa do Sr. João Carlos Novo, pela disponibilização do equipamento sobre o qual foi desenvolvido este trabalho.

Rui Cancela
Setembro | 2007

Robô industrial

“ Máquina manipuladora com vários graus de liberdade controlada automaticamente, reprogramável em posição, velocidade e aceleração, que pode estar fixa ou móvel e é destinada a tarefas de automatização industrial.”

In ISO 8373:1994 [19]

palavras-chave

Células flexíveis de fabrico, manipulador industrial, programação, comunicação

resumo

É ainda hoje relativamente pouco explorado na prática o desenvolvimento de soluções que impliquem conectividade, interface homem-máquina e gestão de células de fabrico utilizando equipamentos robotizados, nomeadamente quando utilizados em conjunção com outros equipamentos produtivos. Uma maior e melhor interactividade e conectividade com os controladores é bastante interessante e, por isso, pertinente de explorar e desenvolver. Os robôs industriais ainda não são utilizados em diversas aplicações tipicamente encontradas em *PMEs*, dadas as complexas configurações das células de fabrico, restrições do ambiente operativo e dificuldades na programação dinâmica de tarefas. A flexibilidade é, sem dúvida, o mote da robótica industrial actual.

Assim, este trabalho adopta o caso particular do controlador *FANUC*, com grande expressão no parque industrial Ibérico, e pretendeu explorar e desenvolver interfaces para a conectividade com outros equipamentos, incluindo computadores ou outras máquinas industriais, utilizando as ferramentas e hardware disponibilizados pelo fabricante.

No âmbito deste trabalho foram desenvolvidas ferramentas de software para o controlador *FANUC RJ3iC* actuando como entidade servidora de uma arquitectura cliente-servidor baseada em *ethernet* e *sockets TCP/IP* e implementando uma linguagem, também ela desenvolvida no âmbito deste trabalho, de suporte à programação e comunicação entre dispositivos clientes e o servidor.

Foi também desenvolvida uma aplicação cliente para *PC* actuando como dispositivo cliente remoto de forma a validar a aplicação e demonstrar a robustez, versatilidade e eficiência do processo de comunicação e programação.

Os resultados obtidos demonstram que a abordagem e a estratégia utilizada na implementação da ferramenta desenvolvida são adequadas e contribuem para melhorar a comunicação entre dispositivos, disponibilizando um interface de programação de robôs industriais a dispositivos remotos baseados em PC com total independência de plataforma e utilizando redes de comunicação convencionais *ethernet*.

key words

Flexible work cells, industrial manipulator, programming, communication

abstract

The development of solutions that imply connectivity, man machine interface and work cell management using robotics equipments is still relatively unexplored in practice, namely when used in conjunction with other production equipments. More and better interactivity and connectivity with the robots is interesting and, thus, pertinent to explore and develop. Industrial robots are still not used in several applications typically found in the *SME's* given the complex work cell configurations, operating environment restrictions and difficulties in dynamic task programming. Flexibility is, undoubtedly, the keyword of current industrial robotics.

Thus this work adopts the particular case of the *FANUC* controller, which has great expression in the Iberian industrial park and aims to explore and develop interfaces for the connectivity with other equipments, including computers or other industrial machines, using the tools and hardware made available by the manufacturer.

Software tools for the *FANUC RJ3iC* controller were developed, this acting as server entity of ethernet client-server architecture and TCP/IP sockets and implementing a language, it too developed along this work, that supports programming and communication between client entities and the server.

A client PC application was also developed, acting as a remote client, in order to validate the application and demonstrate the robustness, versatility and efficiency of the communication and programming process

The results obtained show that the adopted concept and strategy for the developed tool contributes to enhance the communication between equipments, making available an industrial robots programming interface to remote PC based equipments, with full platform independence and using ethernet conventional communication networks.

índice de figuras

índice de tabelas

índice

acrónimos e siglas

AP – Access point
CAD - Computer-aided design
CAM - Computer-aided manufacturing
CIM - Computer-integrated manufacturing
DeviceNet – Rede de campo
DHCP - Dynamic Host Configuration Protocol
DOF – Degrees Of Freedom
EtherNet/IP – Ethernet Industrial Protocol
FTP – File Transfer protocol
HTTP – Hypertext Transfer Protocol
I&DT – Investigação e desenvolvimento tecnológico
I/O – Inputs / Outputs
IANA - Internet Assigned Numbers Authority
Interbus – Phoenix Contact bus
IP – Internet Protocol
ISO - International Standard Organization
LAN – Local Área Network
OSI - Open System Interconnect
PC – Personal computer
PCMCIA – Personal Computer Memory Card International Association
PLC – Programmable Logic Controller
PME – Pequenas e médias empresas
Profibus – Process Field Bus
RISC - Reduced instruction set computer
SME - Small and medium enterprise
SSID – Service set identifier
TCP – Tool Center Point
TCP – Transfer Control Protocol
UDP – User Datagram Protocol
UNECE - United Nations Economic Commission for Europe
USB – Universal Serial Bus
WiFi – IEEE 802.11 - Wi-Fi Alliance - Redes sem fios

marcas

ABB® – Asea Brown Boveri Robotics - Fabricante de manipuladores industriais com marca registada
Chrysler® - Chrysler Corporation – Construtor automóvel norte-americano
DS® DELMIA® – Dassault Systemes DELMIA – Software de programação offline paramétrico com aplicações CAD do mesmo fabricante
FANUC® - Fabricante de manipuladores industriais com marca registada
FlexPendant® - Consola de programação de manipuladores industriais ABB
Inform® - Linguagem de programação de manipuladores industriais YASKAWA
Interbus – Rede de campo
iPendant® - Consola de programação de manipuladores industriais ABB
ISO - International Standard Organization
KCP® - Consola de programação de manipuladores industriais KUKA
KRL® – Linguagem de programação de manipuladores industriais KUKA
KUKA® – Fabricante de manipuladores industriais com marca registada
Rapid® - Linguagem de programação de manipuladores industriais ABB
TPE®, KAREL® - Linguagem de programação de manipuladores industriais FANUC
XRC® - Consola de programação de manipuladores industriais YASKAWA
YASKAWA® – Fabricante de manipuladores industriais com marca registada

1. Introdução

Verifica-se que a robótica industrial é, de facto, um mercado em expansão. Em consequência, existem largos recursos junto dos integradores disponíveis para alocar a *I&D*, especialmente em aplicações industriais por explorar, criando uma oportunidade valiosa para quem pretenda enveredar pela robótica industrial.

1.1. Objectivos genéricos

O desenvolvimento deste trabalho tem por objectivo principal contribuir, de alguma forma, para aumentar a quantidade e diversidade de tarefas industriais robotizáveis, apresentando a robótica como cada vez mais uma alternativa a outro tipo de soluções. Para isto, pretendem-se desenvolver ferramentas, essencialmente de software, que aproximem o mundo dos controladores de manipuladores industriais, proprietários e fechados, ao mundo dos *PCs*, resolvendo uma questão preponderante, a comunicação entre ambos.

1.2. Método

O método utilizado neste trabalho poderá ser resumido em 5 partes com a seguinte ordem cronológica:

a. Levantamento do estado da arte de sistemas de interface e comunicação com o controlador de robôs industriais *FANUC*.

b. Levantamento das soluções para comunicação e controlo remoto de robôs industriais, e respectivas extensões desenvolvidas para problemas similares de outros fabricantes.

c. Desenvolvimento de ferramentas de comunicação entre um computador pessoal e o controlador *FANUC RJ-3iC* sobre *TCP/IP*

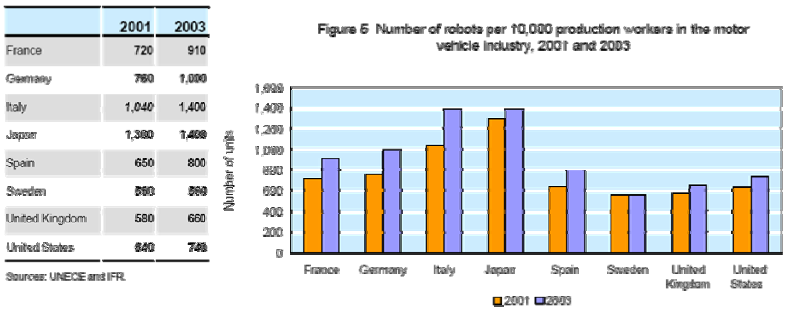
d. Implementação de uma biblioteca de funções que permita o desenvolvimento de aplicações integradas entre manipuladores *FANUC* e sistemas periféricos baseados em *PC* ou similares.

e. Desenvolvimento de uma aplicação ilustrativa das funcionalidades da biblioteca utilizando um robô real e uma aplicação concreta integrável num processo de fabrico.

2. Enquadramento

2.1. I&D em robótica industrial

O investimento em *I&DT* em robótica industrial é extremamente interessante na medida em que é um mercado enorme e em contínua expansão. Segundo a *World Robotics 2004*, publicada pela *UNECE - United Nations Economic Commission for Europe* [2], calcula-se que estejam em operação pelo menos 800.000 robôs industriais em todo o mundo, dos quais cerca 350.000 apenas no Japão, 112.000 nos Estados Unidos e 250.000 na Europa [Figura 1].



	2001	2003
France	720	910
Germany	760	1,000
Italy	1,040	1,400
Japan	1,380	1,400
Spain	650	800
Sweden	590	660
United Kingdom	580	660
United States	640	740

Sources: UNECE and IFR

Figura 1 – Instalações de robôs industriais por pais

Prevê-se que este número cresça para cerca de 1 milhão de unidades até finais de 2007, sendo que o maior incremento será na Europa.

Na última década verificou-se uma melhoria extraordinária nas características técnicas e desempenho dos manipuladores industriais e um decréscimo no seu preço comercial [Figura 3]. Um manipulador médio em 2003 custava já um quarto do seu equivalente vendido em 1990. Pensa-se, no entanto, que o preço deste tipo de equipamentos atingirá o seu equilibro em breve dado que, em condições típicas, é possível atingir o *break-even* do seu custo entre o primeiro e segundo ano de operação.

A eficiência e o custo competitivo de uma unidade robotizada face aos custos escaláveis da mão-de-obra em paralelo com a racionalização constante nos meios industriais explicam a forte penetração no mercado.

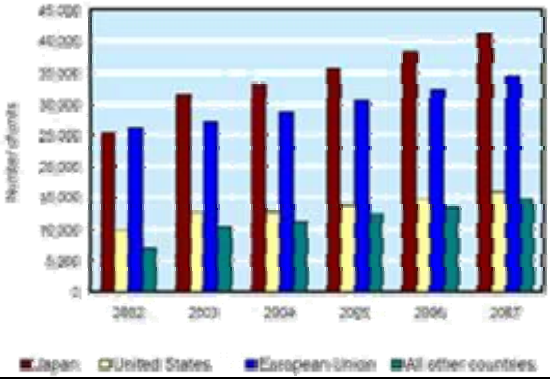


Figura 2 – Instalações de robôs industriais 2002-2003 e previsão 2004-2007

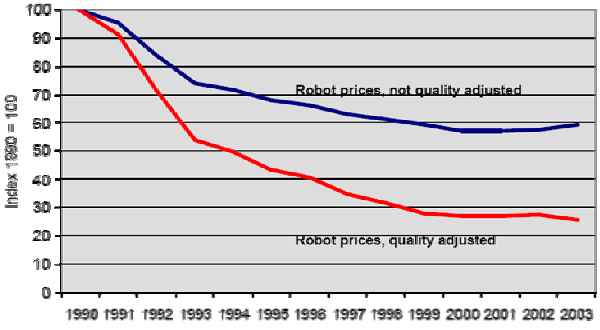


Figura 3 – Evolução do preço de robôs industriais 1990-2003

2.2. Robotização de tarefas industriais

Muito embora exista aquela imagem romântica de um robô industrial [Figura 4], nem sempre é possível utilizar um robô industrial isolado, ou seja, sem fazer uso em paralelo de outras tecnologias que possibilitem a resolução da tarefa. A título de exemplo, para manipular e paletizar peças à saída de um equipamento produtivo, utilizando um manipulador, poderá ser necessário utilizar um sistema de visão artificial para localizar a peça a manipular, o que já terá um custo tecnológico bastante mais elevado tanto na aplicação das tecnologias como na sua interligação.

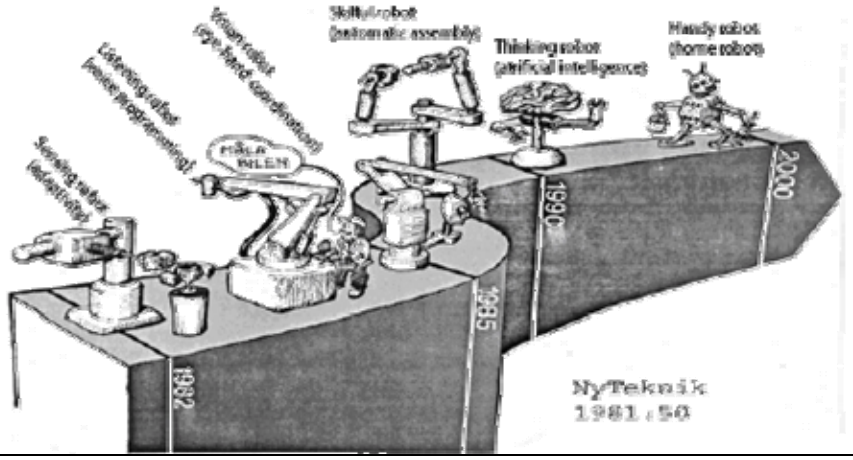


Figura 4 – O futuro da robótica industrial em 1981

Estas questões são exactamente aquelas que motivam o desafio de perseguir e reinventar a utilização de manipuladores industriais que são, hoje em dia, utilizados em tarefas de baixo teor tecnológico, mas que detêm por si um potencial enorme aplicável em industrias tão díspares como a alimentar e a metalomecânica. Estas podem também ser vistas como as oportunidades futuras no mercado da robótica industrial dado que os mercados convencionais, tal como a soldadura, começam a ficar saturados.

2.3. A robótica na industria

A robótica enquanto segmento industrial poderá ser segmentada em 3 grupos distintos: fabricantes, integradores e utilizadores.
Os primeiros são, tipicamente, grandes empresas multinacionais produtoras dos elementos mecânicos, electrónicos e de software, e que têm por missão comercializar produtos robotizados para integração em células de fabrico e cujo canal de distribuição termina, normalmente, nos integradores.

O segundo grupo, os integradores, reporta-se normalmente a parceiros dos fabricantes, integrando apenas elementos mono marca, especializando o seu quadro de técnicos na tecnologia desse fabricante. Têm por missão desenvolver, produzir e comercializar equipamentos produtivos mais ou menos estandardizados dedicados a soluções para problemas industriais.

O terceiro grupo, caracteriza-se por utilizadores normalmente ligados às equipas de manutenção das organizações onde se encontram as células que integram manipuladores. São, tipicamente, operadores das consolas de programação, dedicando-se apenas a ajustes na programação de trajectórias e

afinações de processo. Não dispõem, normalmente, de conhecimentos de robótica teórica e tendem a encarar os manipuladores como equipamentos para resolver problemas repetitivos e sem grande interacção tanto com o utilizador como com o processo produtivo.

O verdadeiro campo de desenvolvimento de soluções industrias, sob a forma de células de fabrico está, essencialmente, na mão dos integradores na medida em que são estes que procuram o mercado, conhecem as necessidades, criam, projectam, fabricam e integram todos os componentes.

O grande desafio na robótica industrial é o desenvolvimento de soluções que impliquem conectividade, interface homem-máquina e gestão de células de fabrico utilizando equipamentos robotizados, nomeadamente quando utilizados em conjunção com outros equipamentos produtivos e, ainda hoje, mais ou menos revolucionários.

As economias modernas dependem cada vez mais da sua capacidade de fabricar produtos competitivos e inovadores o que implica uma enorme flexibilidade das unidades fabris e, por conseguinte, nas células de fabrico. É exactamente aqui que os manipuladores industriais desempenham um papel muito importante na medida em que são, por definição, máquinas altamente versáteis e eficientes e que conseguem desempenhar tarefas tipicamente desempenhadas por humanos.

A maioria dos resultados de I&D na área da robótica industrial foi obtida durante a década de 80. Alguns exemplos são o controlo de força e o planeamento de trajectórias. Agora, cerca de 20 anos depois, muitas destas tecnologias acompanhadas pelo desenvolvimento e consequente redução de custo do hardware, a utilização e aplicabilidade industrial. A questão agora coloca-se em saber se esta situação poderá ser melhorada tecnologicamente e em termos da sua aplicação prática.

Os robôs industriais ainda não são utilizados em diversas aplicações tipicamente encontradas em *PMEs* dadas as complexas configurações das células de fabrico, restrições do ambiente operativo e dificuldades na programação dinâmica de tarefas. A flexibilidade é, sem dúvida, o mote da robótica industrial dos nossos dias. É necessário modificar de forma simples, rápida e eficaz, parâmetros do software de controlo do equipamento com vista à eficaz adaptação ás tarefas.

2.4. Manipuladores industriais em células flexíveis de fabrico

Uma configuração típica de uma célula de fabrico envolve a utilização de um *PLC* como componente central. É este equipamento que normalmente controla todos os elementos escravos tais como sensores, actuadores mecânicos ou pneumáticos, ou mesmo um manipulador através dos seus barramentos de *I/O* digitais ou analógicas. Dado que os controladores dos manipuladores se encontram optimizados para o movimento e respectivo planeamento de trajectórias e as respectivas unidades mecânicas são utilizadas apenas para realizar os movimentos repetitivos, não é efectuado grande investimento no desenvolvimento em interface e sistemas de comunicação. Esta solução é extremamente comum e, muito embora robusta e estável, limitada sempre que existir necessidade de conectividade e ligação a sistemas baseados em *PC*.

Dadas as necessidades dos dias de hoje, em especial relativamente à conectividade, seria interessante colocar no centro nevrálgico da célula de fabrico, em alternativa ou mesmo em paralelo com um *PLC*, um *PC*. O *PC* poderia elevar a fasquia tecnológica das aplicações e potenciar a resolução de muitos problemas

que são hoje de difícil resolução implicando muitas vezes o abandono do projecto ou o contornar das dificuldades de forma tecnicamente incorrecta.

As ultimas gerações dos controladores dos fabricantes líderes de mercado já dispõem da possibilidade de manipulação de redes de campo (*DeviceNet, Interbus, Profibus*, etc.), sistemas de visão ou mesmo sensores de força. No entanto, volta-se a cair no problema da arquitectura fechada e de todos os problemas que dai advêm.

Uma célula de fabrico baseada num *PC* poderá facilmente, e com custos reduzidos, não só fazer tudo aquilo que um controlador proprietário e fechado faz, mas muito mais. Poderá ser utilizado para comunicar com um controlador proprietário de um manipulador, e fazer uso das funcionalidades ligadas ao movimento das unidades mecânicas de que dispõe, mas acima de tudo, poderá ser utilizado para cumprir outro tipo de tarefas, que não seriam possíveis cumprir com um controlador, por não dispor de tais funcionalidades ou que se tornaram de tal forma complicadas de instalar, configurar e operar que não justifica sequer o investimento necessário.

É hoje em dia consensual que grande parte das aplicações industriais que incluem manipuladores industriais requerem alguma inteligência, ou seja, nem sempre é suficiente incluir um manipulador pré-programado que executa uma qualquer tarefa de posicionamento e que trabalha isolado do resto da célula onde se encontra instalado. Em muitas aplicações industriais torna-se necessário recorrer a algum controlo em malha fechada seja incluindo visão artificial para, por exemplo, detectar a posição e orientação do objecto a manipular, seja controlo de força para operações de polimento onde é necessário corrigir a trajectória do robô em função de um feedback de um sensor externo.

No passado, este tipo de problema era resolvido de formas engenhosas, muitas vezes utilizando dispositivos mecânicos complexos para, por exemplo, garantir que o objecto a manipular se encontra sempre com a posição e orientação preestabelecida, eliminando a necessidade de inteligência. Este tipo de aproximação além de muito pouco flexível, exigindo um enorme investimento de tempo de *set-up* sempre que o objecto, por força das necessidades produtivas, muda de forma, torna-se muito caro para as exigências da indústria dos dias de hoje.

3. Programação e interface de robôs industriais

3.1. Linguagens e técnicas de programação: estado da arte

A programação de robôs industriais, forma de instruir um robô a executar uma dada tarefa, começou quase no mesmo instante em que estes apareceram.

Existem várias abordagens à programação de robôs industriais. *Lozano Perez* [20] dividiu as linguagens de programação de robôs industriais em 3 categorias: sistemas de *guiamento*, programação ao nível do robô e programação ao nível da tarefa.

A mais difundida de todas é denominada de *guiamento* ou ensino por demonstração. Consiste em levar manualmente o robô a vários pontos, nos quais a configuração de juntas é gravada. Assim, um programa desta linguagem não é mais do que uma movimentação do robô por uma sequência de vários pontos gravados no espaço. Adicionalmente, em cada ponto, podem ser enviadas instruções para equipamento externo, tais como fechar a garra, começar a soldar, etc. Esta técnica é muito simples de implementar tendo no entanto algumas limitações que derivam do facto de não possibilitar uma movimentação condicionada por informação vinda de sensores externos.

Existem algumas linguagens que não só disponibilizam o acesso a sensores como permitem especificar as movimentações do robô com base nesse feedback externo. São denominadas linguagens ao nível do robô e têm a vantagem de poderem lidar com tarefas mais complicadas onde, por exemplo, existe incerteza em relação ao posicionamento dos objectos a manipular. Como desvantagens têm o facto de necessitarem de peritos que estejam aptos a desenvolver estratégias de movimentação baseadas em sensores, competências raramente encontradas num normal operário fabril.

Uma das abordagens que procuram obviar esta limitação é a programação ao nível de tarefas, onde se definem objectivos para o posicionamento de objectos, em vez de se especificarem as movimentações adequadas para cumprir esse objectivo. Este tipo de linguagem pretende ser independente do tipo de robô, pois não é necessário lidar com trajectórias, cinemática e geometria do robô, entre outras. [20].

Hoje em dia, a categoria mais utilizada é a de programação ao nível do robô utilizando linguagens proprietárias de alto-nível

Quanto à técnica de programação utilizada os robôs industriais são, hoje em dia, programados utilizando uma de duas técnicas possíveis: programação *online* ou *offline*.

A programação *online* [Figura 5] pressupõe criar programas robô directamente na consola ligada ao controlador do robô, movimentando-o para as posições desejadas através de teclas de direcção ou *joysticks* e memorizando-as com a vantagem da independência da potencial falta de precisão absoluta do equipamento mas sendo um processo moroso, implicando a paragem produtiva do equipamento.

Com técnicas de programação *offline* [Figura 6], os programas-robô são gerados utilizando aplicações para PC, não implicando a utilização do equipamento real mas levantando problemas de calibração entre o mundo real e o virtual.

Ao longo deste capítulo serão abordadas ambas as técnicas referidas evidenciando as suas vantagens e desvantagens.

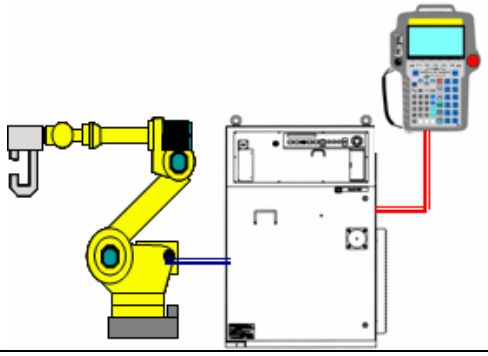


Figura 5 – Programação "on-line"

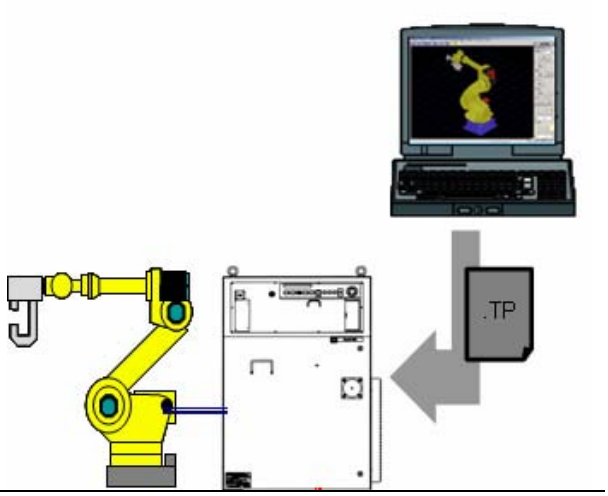


Figura 6 – Programação "offline"

3.1.1. Programação online: Linguagens proprietárias

A programação de manipuladores industriais recorrendo a linguagens proprietárias é o método convencional e o mais utilizado para a programação de manipuladores industriais.

Estas linguagens possuem, normalmente, uma sintaxe muito simples permitindo uma rápida aprendizagem e, na esmagadora maioria das aplicações de *touch-up*, apresenta versatilidade suficiente para cumprir os objectivos. [Figura 7 e Figura 8]

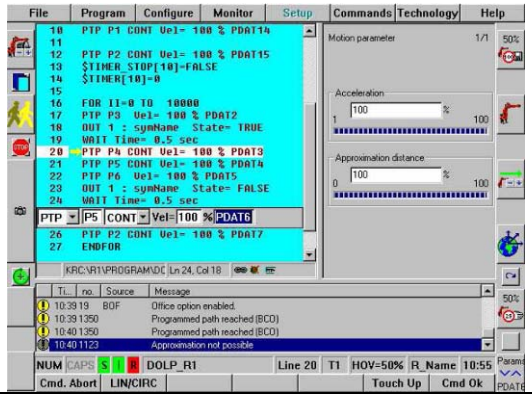


Figura 7 – Linguagem KUKA® KRL®

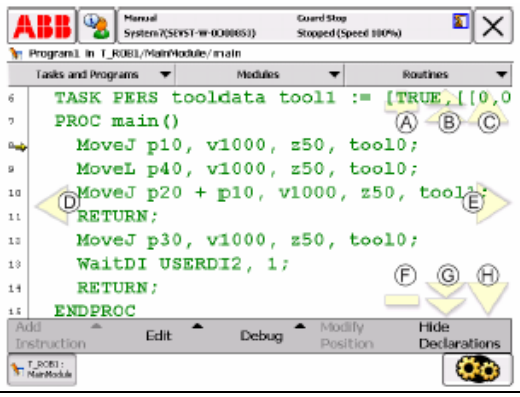


Figura 8 – Linguagem ABB® RAPID®

Estas linguagens, além de pequenas alterações, melhorias e adição de funcionalidade extra, pouco evoluíram desde o seu desenvolvimento original.

Um bom exemplo deste tipo de linguagens é a utilizada pela *FANUC* para a programação em consola dos seus sistemas, a linguagem *TP* [Figura 9]. Os programas escritos nesta linguagem podem correr em controladores ou em ferramentas de simulação *offline* deste fabricante.

Figura 9 – Linguagem FANUC® TPE®

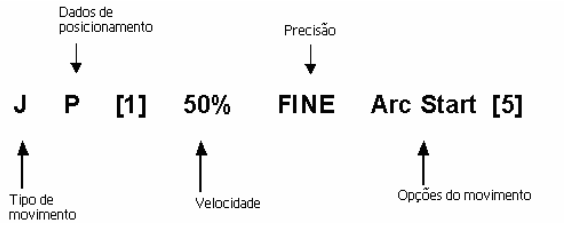


Figura 10 – Exemplo de linha de programação FANUC® TP

Este tipo de linguagens proprietárias e específicas de cada fabricante possui algumas desvantagens entre as quais as mais relevantes são a ausência absoluta de universalidade entre linguagens com a consequente necessidade de *know-how* específico e alguma limitação na potencialidade da linguagem que, para simplificar a sua utilização é, muitas vezes, orientada em função da aplicação do equipamento complicando outro tipo de tarefas menos próximas de acções ligadas ao movimento.

Uma linha de instrução deste tipo de linguagens é, tipicamente, uma composição engenhosa de comandos seleccionados de listas, incluindo instruções de muito alto nível relativas à aplicação a que se destina [Figura 10].

Alguns fabricantes dispõem, também, de linguagens de médio nível. Além da linguagem base utilizada em consola, estas linguagens, tipicamente compiladas, pretendem estender o potencial das linguagens de consola, acrescentando, por exemplo, funcionalidades ligadas à comunicação. Um bom exemplo é a linguagem *KAREL* da *FANUC Robotics* que, em paralelo com a linguagem *TP*, disponibilizam ao programador um maior controlo sobre o equipamento.

3.1.2. Programação offline

Programação *offline* pode definir-se como um conjunto de técnicas para a programação de robôs sem a utilização de um robô real.

Estas aplicações contam também com ferramentas avançadas que, entre muitas outras funcionalidades, permitem modelar quer a unidade robô quer os componentes periféricos da célula de fabrico, permitem estimar tempos de ciclo, calcular alcances e volumes de trabalho, optimizar trajectórias, detectar colisões, com claros benefícios em aplicações industriais, tornando possível, por exemplo, dimensionar o manipulador necessário para uma dada aplicação, ajudar a projectar ferramentas e, até, calcular a estrutura de custos de uma instalação fabril.

Em células complexas, seja pela utilização de vários manipuladores [Figura 11] e posicionadores, seja porque o processo tecnológico obriga a movimentos complexos [Figura 12], a programação com consola de programação seria além de muito morosa, bastante mais difícil. Experimentar estratégias, optimizar trajectórias e calcular tempos de ciclo são tarefas mais fáceis e rápidas de executar numa aplicação *PC* em ambiente calmo e onde os erros não são penalizados com custos reais.

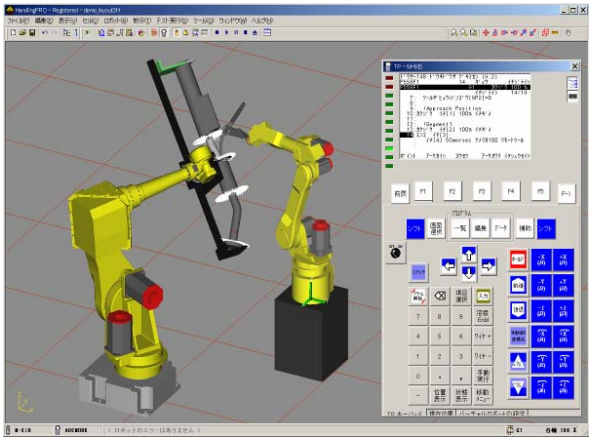


Figura 11 – Programação offline de trajectórias – FANUC® Roboguide®

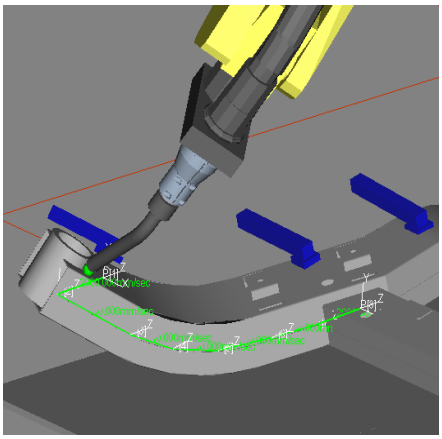


Figura 12 – Detecção automática de arestas - FANUC® Roboguide®

Os ambientes de simulação oferecem uma emulação quase perfeita do controlador real. Assim, é possível utilizar a mesma linguagem disponível na consola de programação tornando-os até numa ferramenta formativa bastante ponderosa. Treinar operadores numa aplicação deste tipo tem um custo e uma rentabilidade muito mais elevada e diminui a necessidade de disponibilidade de equipamentos reais [Figura 13 e Figura 14].

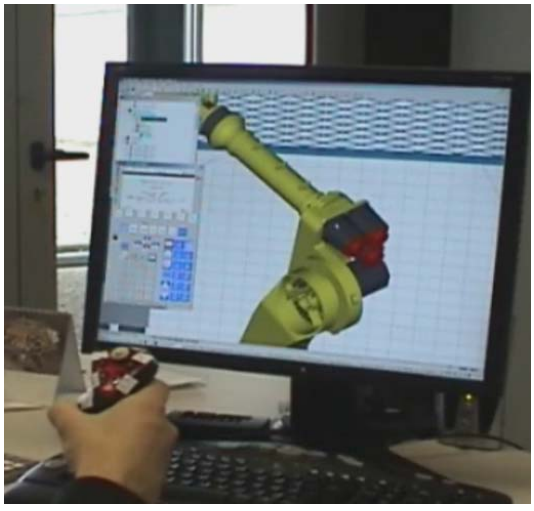


Figura 13 – Programação offline com dispositivos periféricos de interface

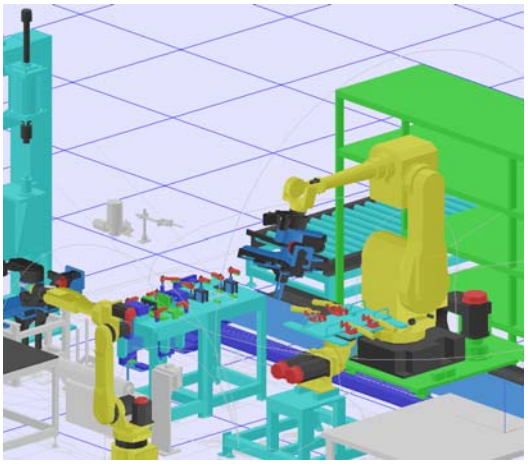


Figura 14 – Simulação de célula flexível de fabrico

Hoje em dia, todos os fabricantes de manipuladores dispõem de aplicações de programação *offline* e existem até aplicações *CAD* que já dispõe de ambientes de simulação de robôs integrados de forma paramétrica no sistema *CAD*, este últimos com a vantagem de alguma independência das linguagens proprietárias dos fabricantes, pois os programas robô são gerados através de um pós-processador semelhante aos disponíveis em aplicação de *CAM* no mundo das máquinas-ferramenta [Figura 15].

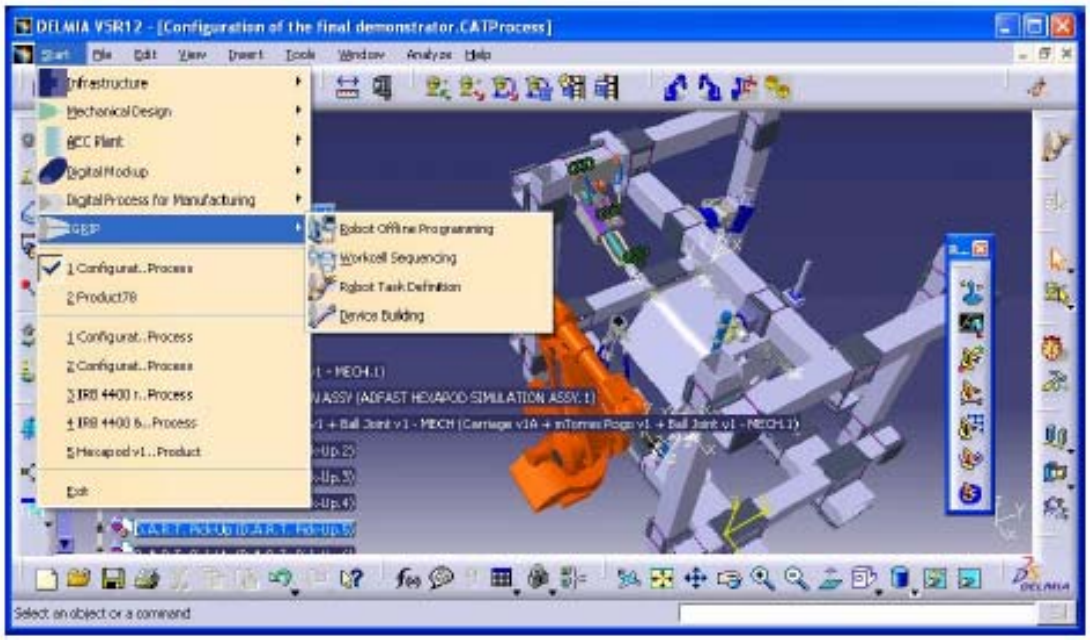


Figura 15 – Programação offline – DS DELMIA

Fazendo um paralelo com o mundo das máquinas-ferramenta, onde uma aplicação de *CAD* gera o desenho-peça e, normalmente, uma outra aplicação *CAM* gera a trajectória a cumprir para obter o contorno criado.

Com as tradicionais consolas de programação os equipamentos têm que ser retirados de produção, implicando percas directas de produtividade.

Em estudos efectuados verifica-se que a utilização de aplicações de programação *offline* podem reduzir o tempo em que o equipamento está fora de produção até 85%. Uma demonstração nas instalações fabris da *Chrysler*, mostrou reduções reais entre os 50 e os 60% [4].

No entanto, este tipo de programação possui um conjunto de desvantagens. Ao programar utilizando uma tradicional consola de programação e técnicas de *touch-up*, ou seja, conduzir o manipulador até aos pontos desejados e só então guardá-los, apenas se faz uso do potencial de repetitibilidade, ou seja, da capacidade que o equipamento tem de se posicionar num mesmo ponto repetidamente. Programar utilizando técnicas *offline*, torna necessário recorrer também ao potencial de precisão, ou seja, da capacidade que o manipulador tem de posicionamento absoluto. O posicionamento de um manipulador é, no limite, fruto do cálculo matemático que posiciona as suas juntas de forma a obter uma dada posição cartesiana do seu *end-effector*.

Factores como as tolerâncias construtivas da unidade mecânica, as propriedades do material utilizado no braço, a resolução e precisão dos codificadores utilizados, folgas e desgastes mecânicos e a dificuldade de repetitibilidade dos métodos de calibração, impõem a cada manipulador industrial características únicas mesmo entre pares [4].

Todas estas questões são muito difíceis senão impossíveis de modelar numa aplicação *PC*. Assim e dado que por um lado o modelo cinemático do robô não consegue, no limite, proporcionar um posicionamento absolutamente correcto e, por outro lado, o modelo cinemático utilizado na aplicação não consegue modelar com exactidão o equipamento real, um programa-robô gerado desta forma poderá revelar diferenças mais ou menos significativas face ao programado.

Reduzir o impacto da necessidade de precisão absoluta, implica calibrar os programas gerados em *offline* na célula real ou mesmo retocar os pontos do programa gerado.

Desta forma, verifica-se que muitas vezes as ferramentas de programação *offline* são muito mais úteis ou como ferramenta de engenharia no desenvolvimento da célula de fabrico ou utilizadas como forma de estruturar o programa robô em termos de estratégia e menos como uma ferramenta capaz de gerar programas chave–na-mão.

3.2. Controladores e dispositivos de interface

3.2.1. Controladores industriais: Estado da arte

Os controladores de manipuladores industriais são, hoje em dia e na essência, dispositivos de controlo de posição, fechados e algo limitados quando comparados com os computadores actuais [1].

No passado popularizaram-se sistemas completamente abertos em que a malha de controlo do hardware bem como o acesso a todas as variáveis era acessível. No entanto, este tipo de acesso de baixo nível deixou de ser potenciado pelos fabricantes dos robôs industriais actuais. A maioria dos fabricantes deste tipo de equipamentos (*FANUC, ABB, Kuka, MotoMan*) dispõe nos seus controladores de interfaces para controlo de movimento.

Muito embora já existam fabricantes que dispõem de controladores baseados em *PC*, a maioria ainda continua a utilizar controladores proprietários, ou seja, controladores baseados em componentes de hardware não *standard* e de que o fabricante detém todo o *know-how* e tecnologia de fabrico.

Por oposição, controladores baseados em *PCs* são baseados em componentes de hardware disponíveis no mercado, existindo vantagens óbvias tais como a disponibilidade e facilidade de integração e substituição de componentes, baixo custo, escalabilidade, etc.

Pode mesmo fazer-se uma analogia com meios de transporte. Enquanto que controladores proprietários se assemelham a um comboio, em que o utilizador tem um número limitado de opções e tem que respeitar horários predeterminados e estanques, controladores baseados em *PC* assemelham-se a automóveis em que o mesmo utilizador por um lado terá que aprender a conduzir mas que, em contrapartida, passa a dispor de liberdade de destino e horário, podendo mesmo escolher funcionalidades extras como sistemas de áudio e navegação *GPS* de marcas que não da do fabricante do automóvel.

Com controladores abertos, de facto, existe uma questão preponderante que fica mais próxima de ficar resolvida: a comunicação.

No futuro, todos os fabricantes deverão possuir controladores baseados em PC mas até lá, e por forma a diminuir a limitação do que é um controlador proprietário, é importante desenvolver ferramentas que possibilitem a conectividade de controladores com outros equipamentos, nomeadamente *PCs*.

3.2.2. Consolas de programação

As consolas de programação são os dispositivos mais utilizados para programar manipuladores industriais. Estes dispositivos são portáteis, leves e ergonómicos e permitem operar, programar e configurar o robô [Figura 16] [Figura 17].

Tipicamente dispõem de teclados alfanuméricos e de teclas de função que permitem aceder a menus e digitar linhas de programação, bem como dispositivos de segurança tais como botoneiras e interruptores de emergência.

Dispõem também de ecrãs, em alguns casos coloridos e de alta resolução e até tácteis, aumentando o potencial de interacção com o utilizador.

Permitem de uma forma fácil e intuitiva aceder a todas as funcionalidades ligadas ao movimento de um robô, incluindo tarefas de programação de trajectórias bem como parametrização de todos os elementos internos ou periféricos ao equipamento, sem comprometer a portabilidade necessária à sua utilização em ambiente real. Actualmente as consolas de programação dispõem até de ferramentas gráficas e exploradores *web* que permitem desenvolver aplicações de software para, por exemplo, interagir com o utilizador ou monitorizar o processo em tempo real.

Figura 16 – Consola FANUC® *iPendant*® [12]

Figura 17 – Consola YASKAWA® XRC® [12]

É um dispositivo de extrema utilidade, permitindo que o programador se movimente sem restrições e em segurança ao executar tarefas de programação e configuração.

Estudos realizados em finais da década de 90 nos Estados Unidos, mostram que cerca de 70% da programação de robôs industriais é feita recorrendo única e exclusivamente a este tipo de dispositivos [Figura 18].

A utilização deste tipo de dispositivo é, no entanto, transversal numa organização, ou seja, tipicamente a utilização destes dispositivos para operações de configuração e programação está reservada a pessoas tecnicamente melhor formadas, enquanto operadores tecnicamente inferiores utilizam-nas apenas para efectuar movimentos simples em modos manuais e para rearmes após paragens motivadas por problemas de processo [Figura 19].

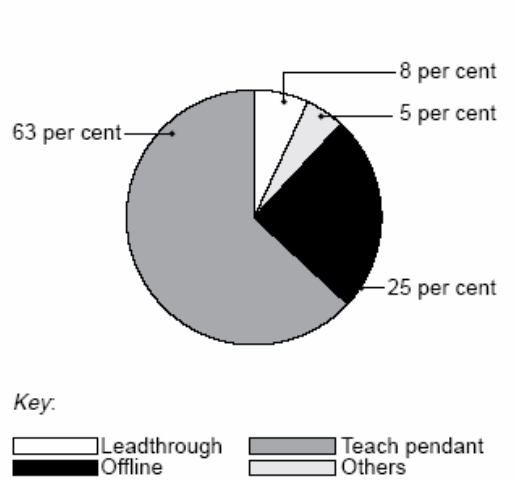


Figura 18 – Distribuição de tipos de programação de robôs na indústria [12]

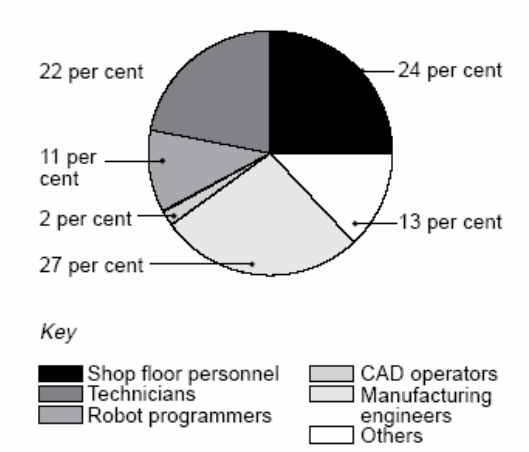


Figura 19 – Distribuição de utilização de robôs por tipo de operador [12]

Este tipo de dispositivos possui, no entanto, algumas desvantagens. No estudo referido, faz-se notar o peso ainda assim elevado para utilização intensiva, o *design* que, por vezes, não é ergonomicamente correcto, contribuindo para doenças profissionais e, muito embora alguns fabricantes já disponham de soluções sem fios, o cabo que liga a consola ao controlador além de incomodo pode causar alguns acidentes.

3.3. Comunicação

Uma grande componente do desafio de utilizar controladores convencionais para aplicações de elevado teor tecnológico passa pela comunicação controlador / *PC*.

A conectividade com os controladores de manipuladores industriais poderá ser segmentada em 3 categorias. Na primeira categoria poderia ser agrupada a conectividade com equipamentos periféricos tais como equipamentos de soldadura. Este tipo de conectividade é, normalmente conseguida recorrendo a redes de campo e *I/O* digitais e analógicas convencionais [Figura 20].

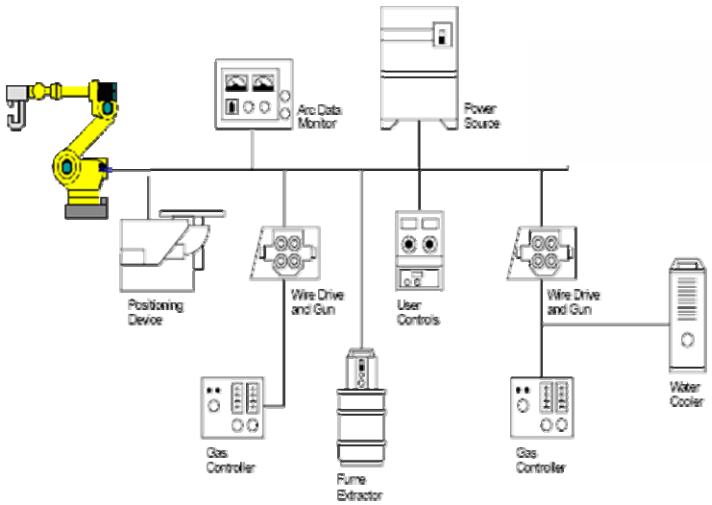


Figura 20 – Dispositivos típicos numa célula de soldadura

Numa segunda categoria poderia agrupar-se todo o hardware de comunicação que tipicamente é utilizado apenas para carregamento ou descarregamento de programas efectuados na consola de programação: portas série, *USB*, *slots PCMCIA*, etc.

Para uma terceira categoria resta a comunicação *ethernet*, sobre a qual podem ser implementados servidores *Http e FTP* mas também pode servir de meio de comunicação de alto débito com um *PC* externo [Figura 21].

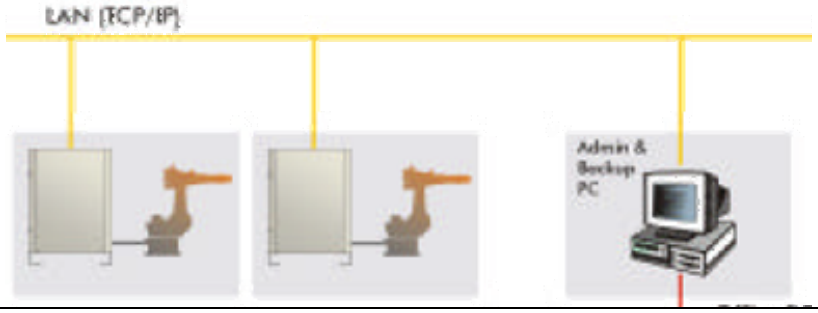


Figura 21 – Robôs industriais em redes de comunicação

Alguns dos maiores desenvolvimentos na tecnologia *ethernet* tais como o *switching*, e o *full-duplex* aumentaram muito o potencial desta tecnologia em redes de comunicação utilizadas em aplicações e para dispositivos industriais.

Entre as vantagens da utilização de *ethernet* para comunicações em rede contam-se a disponibilidade de componentes de *hardware* de elevadas performances a preços muito baixos, possibilidade de utilização de redes estruturadas já implementadas e a robustez de uma tecnologia completamente validada nas mais diversas aplicações, nas mais diversas condições e suportada pelos mais diversos equipamentos.

Ao utilizar *ethernet* passa a ser possível a utilização de protocolos *internet* tais como *FTP* e *HTTP*, o que, dada a facilidade de implementação em software adquire uma vantagem muito interessante face a outras soluções para a interligação de dispositivos.

A performance e a largura de banda comercialmente disponível hoje para *ethernet* é, também, claramente superior a outras redes de campo comercialmente disponíveis [Tabela 1].

Name	*Developer/Year*	*Physical Media*	*Max Nodes*	*Transmission Rate*
Profibus DP/DA	Siemens/94-95	TP or fibre.	127	DP: 9.6kbps to 12Mbps DA: 31.25kbps
Interbus S	Phoenix Contact, Interbus Club/84	TP or fibre.	256	500 kbps
Foundation Fieldbus H1	Fieldbus Foundation/95	TP, fibre or slipring.	240/segment 65000segments	31.25 kbps
DeviceNet	Allen-Bradley/94	TP for signal & power	64	125 to 500 kbps
ControlNet	Allen-Bradley/96	Coax or fibre	99	5 Mbps
CANopen	CAN in automation/95	TP	127	10 kbps to 1 Mbps
Ethernet	DEC, Intel and Xerox /76	TP, coax or fibre	1024+	10 Mbps to 1Gbps

Tabela 1 – Comparação entre redes de comunicação [8]

Existem, no entanto, algumas desvantagens tais como a necessidade de energia para a sua operação, alguma inadaptação a ambientes industriais e, essencialmente, a característica não determinística desta tecnologia, na medida em que não é possível determinar se os pacotes de dados chegaram ou não com atraso, nem medir esse mesmo atraso. Este comportamento é provocado pela própria natureza da tecnologia em que a quantidade de colisões pode atrasar a entrega de pacotes de dados. Assim, e muito embora o problema das colisões possa ser reduzido aumentando a largura de banda da ligação, a utilização de *ethernet* para aplicações em tempo-real fica comprometida, sendo necessário adoptar variações ao protocolo original, como o *ethernet/IP*, para contornar este tipo de problema [13].

3.3.1. Modelo OSI e protocolos de comunicação

O modelo *OSI* é um modelo conceptual composto por sete camadas em que cada uma especifica uma funcionalidade de rede e descreve a forma como a informação é transportada de um aplicação de software de um qualquer dispositivo para uma outra aplicação noutro dispositivo remoto [Figura 22].

Este modelo foi desenvolvido em 1984 *pela International Organization for Standardization, ISO,* e é sobre este modelo que a maioria das redes de comunicação são baseadas, sendo considerado como a arquitectura de eleição para a comunicação entre computadores e computação distribuída.

O modelo *OSI* segmenta a tarefa genérica de enviar dados entre dispositivos em sete subtarefas, atribuindo a cada uma delas uma das sete camadas disponíveis [22] [23].

Cada camada é de tal forma estanque que permite que a sua implementação seja independente das outras [Figura 23].

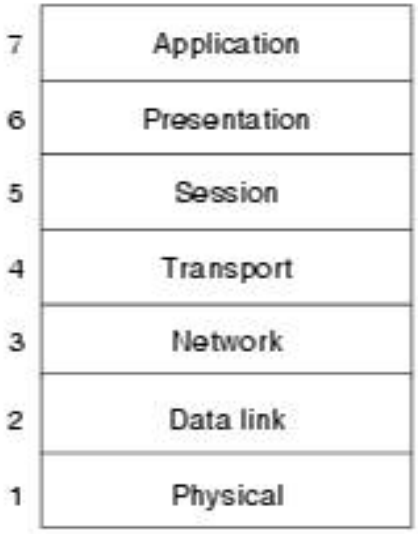


Figura 22 – Modelo OSI – Sete camadas [22]

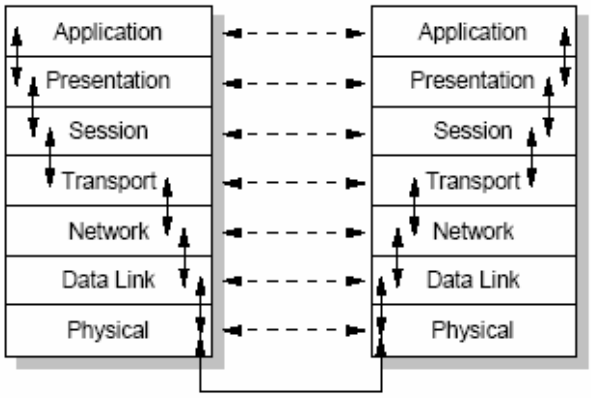


Figura 23 – Modelo OSI – Conectividade entre camadas [12]

Enquanto que as camadas inferiores são responsáveis por disponibilizar comunicação ponto a ponto, garantindo a ausência de erros, as camadas superiores são responsáveis por tarefas mais complexas e geralmente

implementadas apenas em software. A camada mais elevada, a da aplicação, é a mais próxima do utilizador e responsável pela interface.

O modelo *OSI* é, tal como discutido, um modelo conceptual para a interligação de computadores mas não um método de comunicação em si.

Assim foi necessário desenvolver um conjunto de regras, convenções e formalismos que especificassem a forma como os computadores trocam informação sobre uma rede física. Estes conjuntos de regras são chamados de protocolos de rede e podem implementar uma ou mais camadas do modelo *OSI*.

3.3.2. Modelo cliente-servidor

O modelo cliente-servidor descreve a relação entre dois dispositivos em que um dos dispositivos, o cliente, é um consumidor de serviços, fazendo pedidos de serviço a um outro dispositivo, o servidor, este fornecedor de serviços [Figura 24].

A aplicação servidora deverá estar sempre disponível para aceitar pedidos de serviço por parte de clientes, clientes esses que a contacta sempre que necessitam de um serviço por ele disponibilizado.

Entre as vantagens deste tipo de modelo, contam-se a disponibilidade de protocolos assimétricos em que existe uma relação de *muitos-para-um* entre clientes e um dado servidor, onde é o cliente que inicia a ligação com um servidor passivamente à escuta, o encapsulamento de serviços em que o servidor é sempre o dispositivo responsável por executar um dado serviço em função da mensagem recebida, integridade do servidor face à individualização dos clientes.

Muito embora o processo relativo à execução de um servidor possa coexistir num mesmo dispositivo com a aplicação cliente, este conceito é realmente interessante para comunicações sobre redes de dados em que os dispositivos estão disponíveis através de um par endereço/porta.

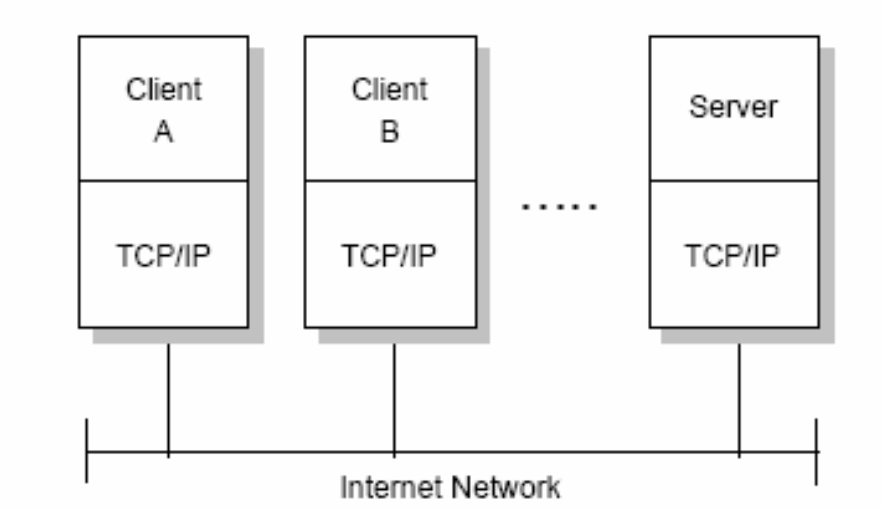


Figura 24 – O modelo cliente servidor para aplicações [12]

Os sistemas baseados numa lógica cliente-servidor são, normalmente, tolerantes à falha na medida em que o servidor pode falhar e recuperar sem que seja posta em causa toda a integridade do sistema.

Uma outra vantagem deste tipo de aproximação à comunicação entre dispositivos é a potencial independência de plataformas. As aplicações cliente e servidor poderão ser desenvolvidas em plataformas diferentes e operar em hardware distinto. Este tipo de modelo é, também, altamente escalável seja nos dispositivos cliente, quantidade e tipo de plataforma, seja nos servidores, relativamente aos serviços disponibilizados e plataforma utilizada.

A especificação da conversação terá que ser baseada num qualquer protocolo entendido por ambos, o servidor e o cliente. Para comunicar com sucesso ambas as aplicações terão que falar a minha linguagem, seguir o mesmo protocolo, ou seja, conhecer as regras e o conjunto de mensagens a esperar um do

outro de forma a agir em concordância. Além disso ambas as aplicações terão que saber quando podem ou devem proceder ao envio ou recepção de dados.

Assim, um servidor *web* escrito numa dada linguagem e para uma dada plataforma poderá comunicar com um cliente *web* escrito numa outra qualquer linguagem a correr numa outra plataforma. Basta apenas que ambos utilizem o mesmo protocolo de transporte, por exemplo TCP/IP, e o mesmo protocolo de aplicação, como por exemplo HTTP:

3.3.3. Sockets TCP/IP

O conceito de *socket* de *Berkeley* é uma abstracção utilizada para representar o elemento terminal de uma ligação entre dois dispositivos. Como que um cabo ligado a cada *socket* mas em que o aspecto físico e de hardware não é relevante e onde os elementos terminais em ambos os extremos importa conhecer.

O objectivo é criar uma camada de abstracção em que não seja necessário conhecer pormenores da ligação física entre dois dispositivos. Basta apenas implementar os tais elementos terminais e deixar que a infra-estrutura de rede, camadas físicas e de transporte, se encarreguem de conduzir a informação a trocar entre dispositivos.

Os *sockets*, de *Berkeley*, devem o seu nome ao facto de terem sido desenvolvidos na Universidade de Berkeley, Estados Unidos da América, nos meados da década de 80. O conceito foi inicialmente desenvolvido como mais um método de comunicação mas cedo evoluiu para o que viria a ser a tecnologia mais utilizada para a comunicação em rede.

Este tipo de aproximação à comunicação em rede pressupõe que os processos de comunicação decorram entre dois extremos ou *sockets*, que podem existir em distintos dispositivos anfitriões ou mesmo entre processos a correr num mesmo dispositivo.

Todos os sistemas operativos e linguagens de programação modernas dispõem de algum tipo de implementação deste tipo de tecnologia dado que se tornou no método *standard* para a ligação de computadores pessoais à Internet.

3.3.4. Protocolos e endereçamento

Os protocolos de comunicação [Figura 25] encontram-se divididos em duas classes diferentes: os protocolos sem ligação e os orientados por ligação.

Enquanto que os protocolos sem ligação, dos quais o *UDP* é um bom exemplo, apenas passam a informação não dispondo de funcionalidades de reconstrução de pacotes nem de retransmissão de pacotes extraviados, implicando que os pacotes de dados poderão chegar desordenados ou mesmo nem sequer chegar.

Esta questão é de grande importância dado que se o protocolo de transporte não assegurar a correcta entrega de informação, as camadas OSI superiores, nomeadamente as de aplicação, terão de garantir que as mensagens chegam inteiras e na sequência correcta. Em regra, os protocolos deste tipo acabam por impor um baixo nível de performance e robustez à aplicação.

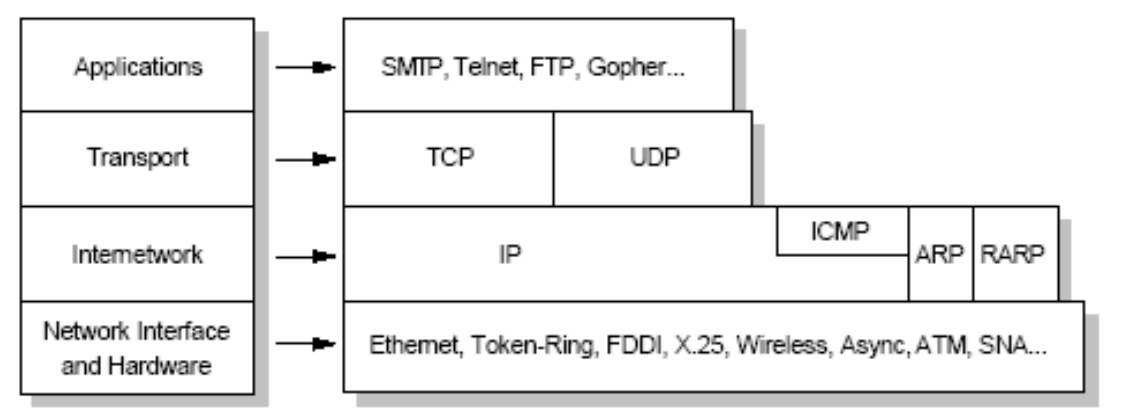


Figura 25 – Protocolo TCP/IP [12]

Num protocolo orientado por ligação, como o *TCP*, a ligação de dados possui estados associados a ambos os extremos da ligação, implicando um procedimento de *handshaking* para o estabelecimento da comunicação e que poderá ser mantido enquanto a ligação se mantiver.

Este tipo de protocolo resolve internamente os procedimentos de *handshaking* e possuiu funcionalidades de correcção de erros e ordenação da informação. Divide as mensagens em pacotes numerados que serão reordenados na sequência correcta no seu destino e consegue gerir pedidos de retransmissão de pacotes que por alguma razão não chegaram ao destino. O *TCP* adiciona os cabeçalhos apropriados para garantir a entrega eficiente dos pacotes de informação no seu destino.

O protocolo *TCP* é, assim, aquele que uma maior estabilidade na troca de infirmação entre dois sistemas assegura, oferecendo robustez e velocidade.

Dadas as vantagens descritas, será com base neste protocolo que serão desenvolvidas as ferramentas no decorrer deste trabalho.

3.3.5. Endereçamento

O endereçamento deste tipo de comunicação é efectuado tendo em conta que os dispositivos intervenientes possuem endereços *IP* únicos e que a cada serviço corresponderá uma e uma só porta. As portas são números de 16 *bits* independentes para *UDP* e *TCP*, sendo que alguns destes números estão já reservados para alguns serviços. Esta *standardização*, regulada pela *IANA* [28], reserva as portas de 1024 a 49151 para utilização de conveniência dos utilizadores.

A implementação da etapa de ligação (*connect()*) sobre *TCP* utiliza o chamado *three-way handshake* em que o cliente envia um bit de controlo do tipo *SYN* (sincronizar números de sequência) ao servidor que, por sua vez, responde com um *ACK* (*Acknowledgement)* indicando que recebeu correctamente o pedido,e envia o seu *SYN*. De seguida o cliente devolve um *ACK* ao *SYN* do servidor e a ligação é estabelecida.

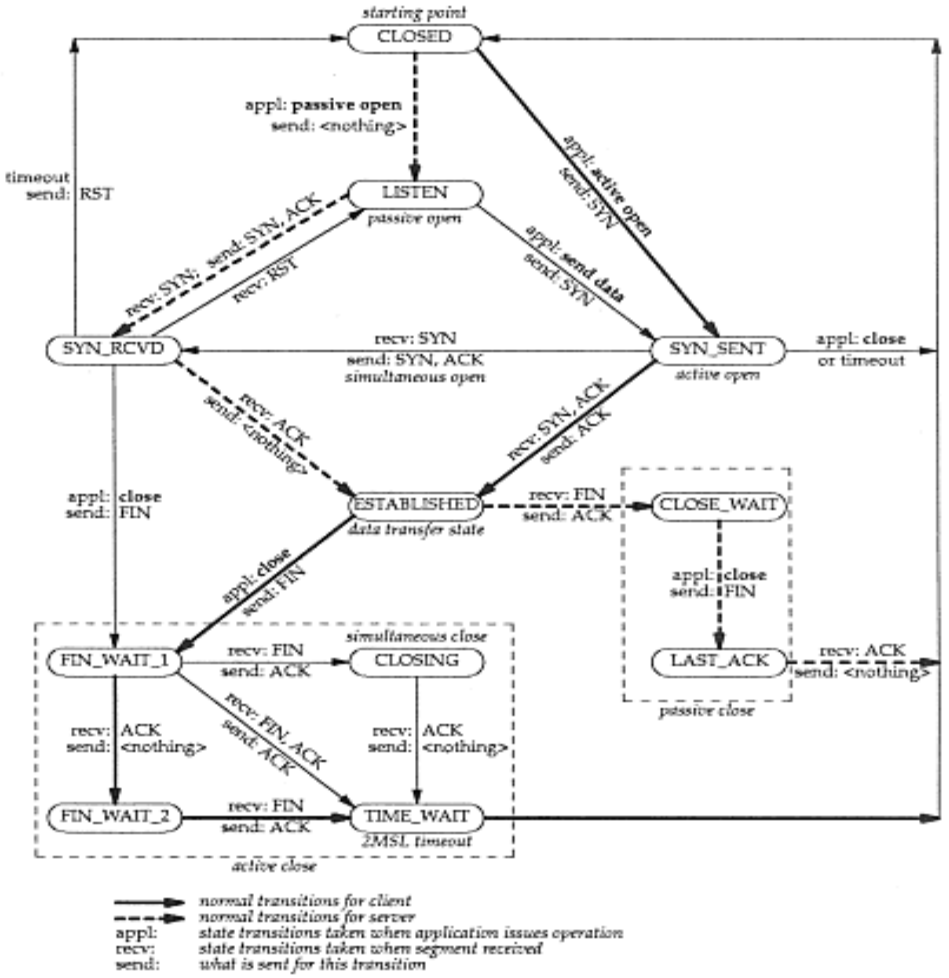


Figura 26 – Diagrama de estados do protocolo TCP

Para a terminação da ligação, o dispositivo que termina a ligação envia um *FIN* ao outro dispositivo que devolve um *ACK* indicando que recebeu correctamente o pedido de finalização e, assim que possível, responderá com um *FIN*, terminando então e definitivamente a ligação, o que será confirmado com um ultimo *ACK* do dispositivo que tomou a iniciativa.

3.3.6. Modos de operação

Na utilização de *sockets* para comunicação, é possível a implementação de dois cenários distintos: bloqueante e não bloqueante.

O primeiro cenário é chamado de operação em modo bloqueante, pois o programa ficará bloqueado até que sejam recebidos dados, ou seja, a tarefa de leitura do *socket* não será terminada até que sejam enviados dados do sistema remoto.

O segundo cenário é chamado de operação em modo não bloqueante dado que a aplicação consegue reconhecer a ausência de informação a ser lida e gerir adequadamente a situação.

Em modo não bloqueante poder-se-á utilizar uma de duas estratégias possíveis: *polling* e notificação assíncrona. Enquanto que a primeira estratégia

passa por tentar consecutivamente ler ou escrever no *socket*, tipicamente recorrendo a um temporizador, a segunda estratégia, e mais utilizada, implica o conceito de evento e a utilização de multi-tarefa, ou seja, existirá uma tarefa que está constantemente a monitorizar o estado do *socket*, mas que se encontra em execução paralela ao programa principal. Se o dispositivo remoto escrever dados no *socket* monitorizado, será gerado um evento que despoletará a execução de uma dada tarefa, como por exemplo, uma rotina de interpretação da trama de dados recebida.

3.3.7. Implementação de servidor

O diagrama de fluxo da [Figura 27] representa a implementação de um típico servidor de *sockets* sobre *TCP/IP* em que o servidor instância um *socket*, ligado a uma dada porta e fica à escuta da mesma até que um cliente remoto a ele se ligue. De seguida processa-se a troca de dados servidor-cliente e, por fim, é encerrada a ligação, em princípio, por iniciativa do dispositivo cliente.

A implementação de um servidor *TCP* envolve, então, as seguintes etapas [Figura 28] :

. Criação do *socket* (*socket()*)
. Ligação do *socket* a uma porta (*bind()*)
. Escutar a porta e aguardar por ligações (*listen()*)
. Aceitar uma ligação de um cliente (*accept()*)
. Troca de dados com o cliente (*write()* / *read()*)
. Terminar o servidor encerrando o *socket* (*close()*)

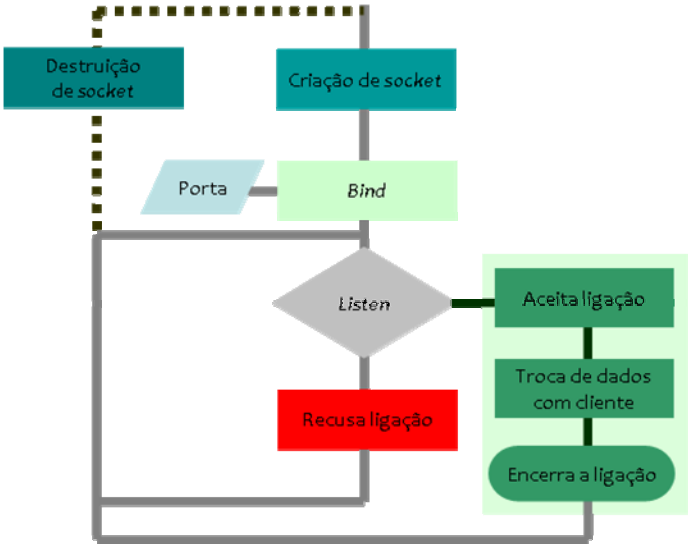


Figura 27 – Etapas e procedimentos de um servidor de *sockets* genérico

```
socket()
bind()
listen()
while ()
   accept()
   while ()
      read()
      write()
   close()
```

Figura 28 – Algoritmo genérico em pseudo-código

3.3.8. Implementação de cliente

O diagrama de fluxo da [Figura 29] representa a implementação de um típico cliente de *sockets* sobre *TCP/IP* em que inicialmente o cliente instancia um *socket* e tenta uma ligação a um par *IP*/Porta onde se espera que um servidor se encontre à escuta. Se a ligação for estabelecida, processa-se a troca de dados servidor-cliente e, por fim, é encerrada a ligação.

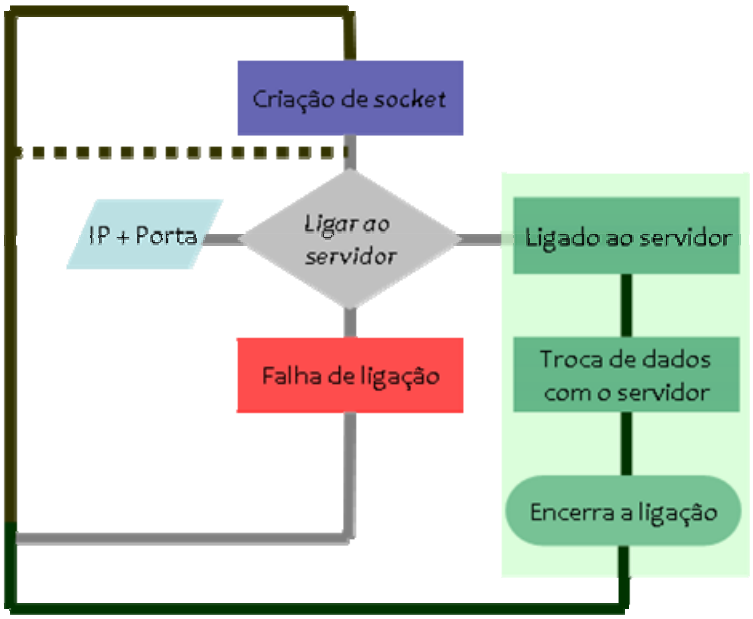


Figura 29 – Etapas e procedimentos de um cliente de *sockets* genérico

```
socket()
connect()
while ()
  write()
  read()
close()
```

Figura 30 – Algoritmo genérico em pseudo-código

A implementação de um cliente TCP envolve, então, as seguintes etapas [Figura 30]:

. Criação de um *socket (socket())*
. Ligar a um servidor à escuta num par *IP* / porta (*connect (IP/port)*)
. Troca de dados com o servidor (*write() / read()*)
. Encerramento da conexão com o servidor (*close()*)

O exemplo descrito resumidamente na Figura 31 e exposto alargadamente no Anexo D a este trabalho desenvolvido sobre a plataforma *.NET*, implementa o cliente em modo síncrono. Foram utilizados os *namespaces System.Net* e *System.Net.Sockets*. São as classes e os métodos disponibilizados por estes *namespaces* que permitem a implementação de *sockets TCP/IP*.

```
Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp)
System.Net.IPEndPoint remEndPto = new IPEndPoint (ipAdd,port)
sClient.Connect (remEndPto)
sClient.Send(dataToSend)
int msgRx = sClient.Receive (msg)
```

Figura 31 – Exemplo de implementação de cliente (C#)

O cliente implementado no âmbito deste trabalho foi, no entanto, desenvolvido numa perspectiva assíncrona. A implementação em modo assíncrono,

bastante mais interessante do ponto de vista funcional, implica acrescentar uma aproximação multi-tarefa ao conceito de implementação descrito.

Fazendo uso das potencialidades de chamadas assíncronas disponíveis na plataforma *.NET* é possível conceber um cliente capaz de gerir mais do que apenas a tarefa de comunicação com o servidor. Para uma aplicação poderá ser importante, por exemplo, a aplicação conseguir gerir a interface com o utilizador e, em simultâneo, as tarefas relativas à comunicação.

4. Abordagem

Neste capítulo será descrita a abordagem utilizada para resolver o problema proposto. Será apresentada a plataforma utilizada, as suas características e as razões da sua selecção bem como a estratégia utilizada e o software desenvolvido.

4.1. Objectivos

O objectivo principal deste trabalho era o de desenvolver uma plataforma que permitisse interligar um controlador de um manipulador industrial a outro tipo de dispositivos utilizando uma tecnologia aberta e relativamente fácil de implementar, disponibilizando serviços representativos de tarefas típicas desempenhadas por um manipulador industrial a dispositivos remotos, tais como um *PC*. Pretendeu-se acrescentar a possibilidade de utilizar dispositivos externos, tais como câmaras ou sensores, para fornecer algum nível de inteligência a um manipulador industrial de forma a ser possível desempenhar tarefas que de outra forma não seriam possíveis.

4.2. Plataforma utilizada

O sistema utilizado neste trabalho consiste nos seguintes componentes:

a. Manipulador industrial - Unidade mecânica e controlador
b. *PC*
c. Ponto de acesso *WiFi*
d. Sistema operativo do manipulador, suas componentes e algumas opções especiais
e. Sistema operativo e aplicações para *PC*

Relativamente ao manipulador industrial, foi utilizada uma plataforma *FANUC*, em particular um controlador *FANUC RJ3iC* e uma unidade mecânica *FANUC M6iB/6S*. A escolha recaiu nesta plataforma por várias razões das quais se destacam o elevado potencial tecnológico e a disponibilidade do equipamento. E é um bom exemplar daquilo que é hoje um manipulador industrial disponível no mercado, com as características do estado da arte nesta área.

O *PC* utilizado é uma plataforma *WININTEL* (*Intel Centrino* e *Windows XP*) e a implementação de rede é baseada num *AP WiFi.*

4.2.1. FANUC M6iB/6S RJ3iC

Foi utilizada uma unidade mecânica *FANUC M6iB/6S* [Figura 32], um manipulador industrial, parte do segmento de manipuladores utilizados preferencialmente em aplicações de manipulação industrial, de 6 *DOF* com capacidade de carga nominal de 6 Kg. É capaz de repetibilidades de 0,08mm e possuiu um alcance de 951mm [Tabela 2].

Esta unidade provou ser uma boa escolha para uma unidade utilizada em laboratório dado que dispõe de características mecânicas e performances ao nível do estado da arte da robótica industrial [Figura 33].

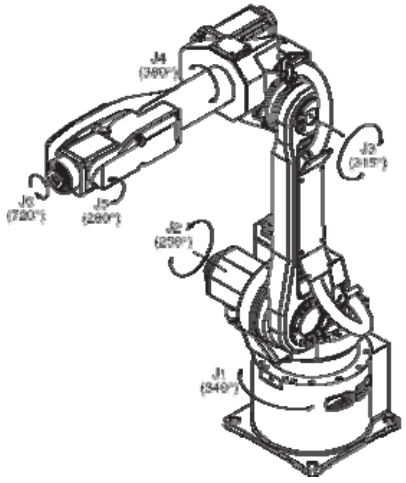


Figura 32 – Unidade mecânica FANUC ®M6iB [10]

M-6*i*B Series Specifications

Items		M-6*i*B	M-6*i*B/6S
Axes		6	6
Payload (kg)		6	6
Reach (mm)		1373	951
Repeatability (mm)		±0.08	±0.08
Interference radius (mm)		273	273
Motion range (degrees)	J1	340	340
	J2	250	250
	J3	315	310
	J4	380	380
	J5	280	280
	J6	720	720
Motion speed (degrees/sec.)	J1	150	200
	J2	160	200
	J3	170	260
	J4	400	400
	J5	400	400
	J6	500	720
Wrist moment (kgfcm)	J4	16.0	16.0
	J5	10.0	10.0
	J6	6.0	6.0
Wrist inertia (kgfcms²)	J4	0.4	0.4
	J5	0.2	0.2
	J6	0.61	0.61

Tabela 2 – Especificações FANUC® M6iB/6S [8]

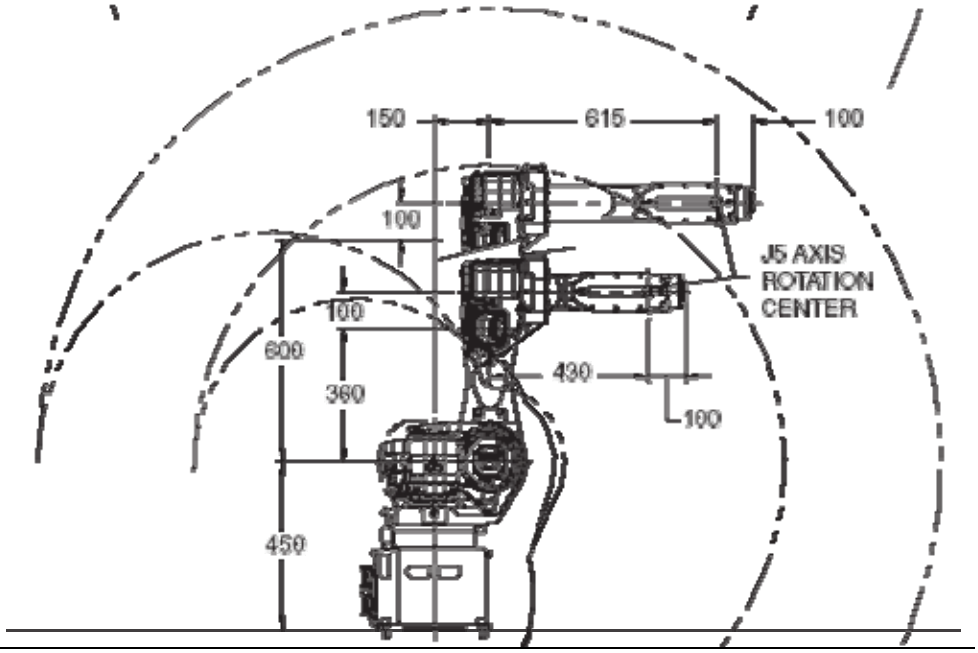


Figura 33 – Volume de trabalho FANUC® M6iB/6S [8]

Quanto ao controlador o *FANUC System RJ3iC*, é a 6ª geração de controladores deste fabricante. É baseado numa arquitectura multi-processador *RISC* de 16 *bits* dispondo de processadores separados para as tarefas de movimento e comunicação. Este controlador é capaz de controlar até 40 eixos divididos em 8 grupos de movimento e dispõe de capacidades avançadas de comunicação e conectividade. Implementa a maioria das redes de campos actuais bem como portas de comunicação série e *ethernet* [Figura 34] [Figura 35] [Figura 36].

Figura 34 – Controlador FANUC® System RJ3iC [8]

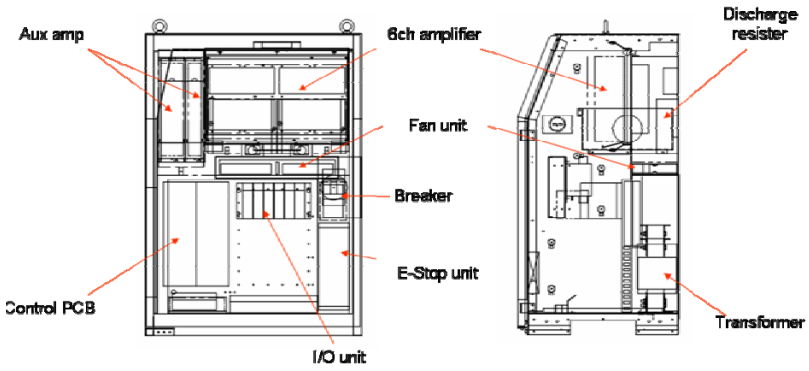


Figura 35 – Aspecto interior do controlador FANUC® RJ3iC [8]

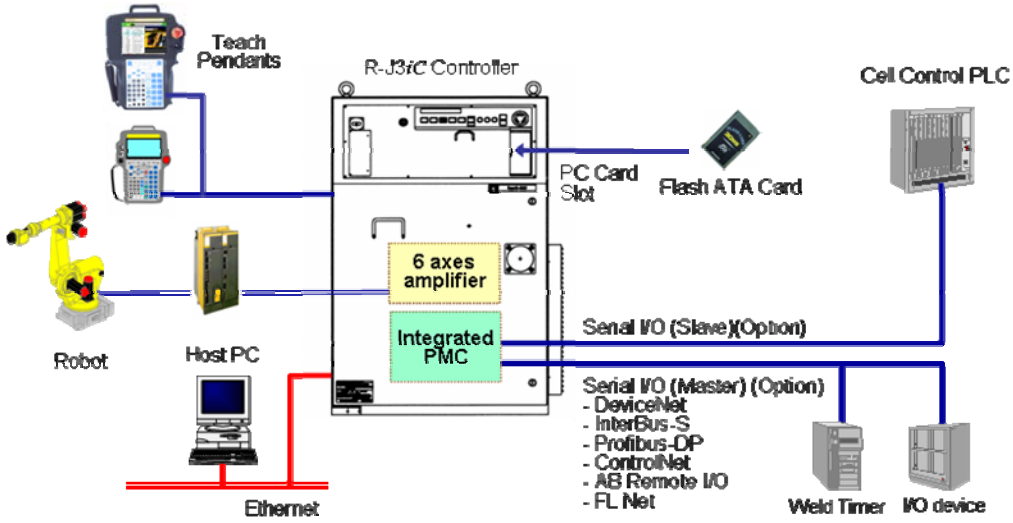


Figura 36 – Conectividade do controlador FANUC RJ3iC [8]

Este controlador dispõe de clientes e servidores *HTTP* e *FTP* implementados de base, disponibilizando serviços de transferência de ficheiros e monitorização de variáveis [Figura 38] assim como um explorador *web* integrado e disponível através da consola *TP* [Figura 37]. O servidor *FTP* foi a funcionalidade utilizada para trocar ficheiros com o controlador no decorrer deste trabalho. Foi também desenvolvida uma página *HTML* específica para monitorização de alguns aspectos do manipulador no decorrer deste trabalho.

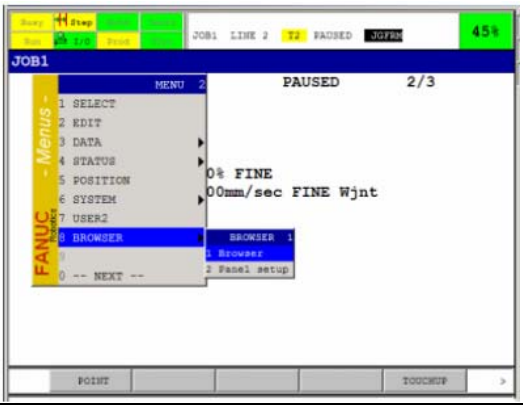


Figura 37 – Explorador HTML integrado

Figura 38 – Servidor HTTP

4.2.3. Sistema operativo - FANUC Handling Tool

O sistema operativo disponível no controlador poder-se-á dividir em 3 grupos: o sistema operativo base, a aplicação relativa ao processo e opções especiais.

O primeiro grupo diz respeito à camada de aplicação que disponibiliza funcionalidades básicas de operação e programação e é inerente ao controlador e respectiva unidade mecânica. Neste caso estava disponível a versão 6.34, a mais recente à data de início deste trabalho.

O segundo grupo diz respeito à camada de aplicação que corre sobre o sistema operativo base e que disponibiliza funcionalidades específicas para a aplicação industrial em questão. Dentro desta família estão disponíveis aplicações especialmente direccionadas para aplicações industriais de soldadura, manipulação, pintura, entre outras. Neste caso estava disponível a aplicação de manipulação, *FANUC Handling Tool*.

O último grupo diz respeito a opções de software instaladas sobre o sistema operativo base que pretendem facilitar a utilização do equipamento e permitem incluir funcionalidades avançadas como, por exemplo, comunicação com dispositivos externos, comandos avançados de movimento, entre outras, numa extensa lista de opções disponíveis. Neste caso foram instaladas algumas opções especiais que implementam algumas funcionalidades necessárias a este trabalho e não disponíveis no sistema operativo base, das quais se destacam:

. KAREL command language and runtimes
    Interpretador de linguagem KAREL
. Ethernet package
    Funcionalidades relativas à utilização da porta ethernet
. DNS, FTP and WEB server
    Servidores DNS, FTP e HTTP

4.2.4. Linguagens de programação

No decorrer deste trabalho foi utilizada a linguagem *KAREL* no desenvolvimento da aplicação servidora sobre TCP/IP para a plataforma *FANUC RJ3iC* e *C#* no desenvolvimento da aplicação cliente sobre *TCP/IP* para *PC*. Foi também utilizada linguagem *FANUC TP* no desenvolvimento de programas *TP* para execução de tarefas de movimento.

A aplicação cliente foi desenvolvida sobre a plataforma *Microsoft® .NET Framework 2.0*, fazendo uso da vasta livraria de classes dedicadas à comunicação *TCP/IP* disponíveis na plataforma.

4.2.5. Implementação em LAN

A implementação do sistema utilizado passou pela sua interligação numa *LAN WiFi* [Figura 39]. Foi utilizado um vulgar *AP / router* ligado ao controlador por uma ligação de cobre a *10/100Mbps* que, por sua vez, disponibiliza uma rede *WiFi*, (*SSID* “fawifi”) protegida. Foi atribuído um *IP* fixo ao controlador e endereçamento dinâmico *DHCP* por parte do *AP* aos dispositivos clientes *WiFi*.

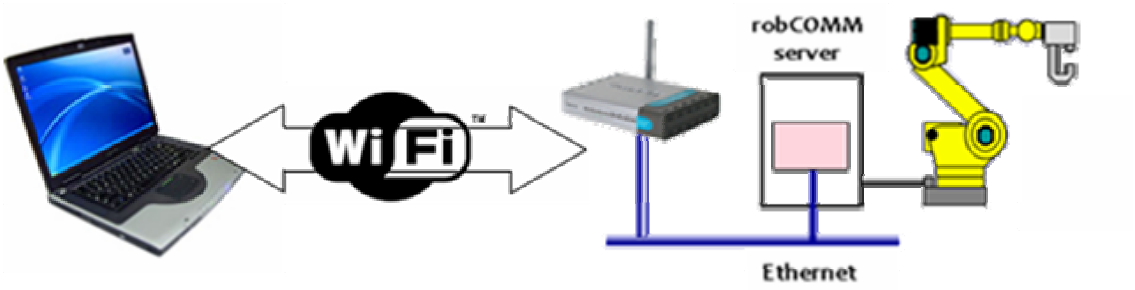


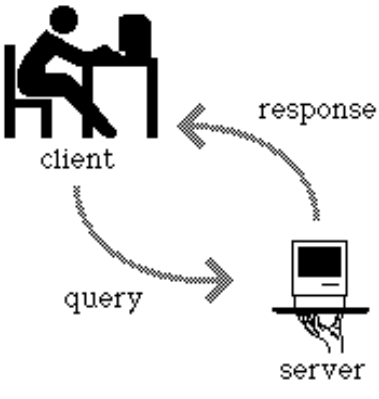
Figura 39 – Implementação da plataforma em LAN

Foi seleccionado este tipo de ligação pela flexibilidade que permite, na medida em que o *WiFi* é hoje uma tecnologia disponível em qualquer dispositivo móvel e que se pretendia permitir o acesso a qualquer dispositivo cliente sem que fosse necessário alterar ou reconfigurar a rede local ou os dispositivos clientes, sendo apenas necessário conhecer o *IP* do controlador servidor.

4.3. Arquitectura e estratégia

Foi seleccionada uma abordagem cliente servidor [Figura 40 / Figura 40] baseada no modelo *OSI*. Pretendeu-se implementar uma plataforma que permitisse algum grau de computação distribuída quando aplicada a casos práticos da utilização de manipuladores industriais em células de fabrico reais, ou seja, pretendeu-se que a aplicação servidora fosse a responsável pela gestão do manipulador, garantindo independência da aplicação cliente cuja implementação deveria estar voltada para o fim a que se destina.

Por outro lado é possível que diferentes clientes sejam implementados para desempenhar diferentes tarefas e que recorram ao servidor apenas e só para fazer uso do tipo de serviços passíveis de serem disponibilizados por um manipulador industrial.



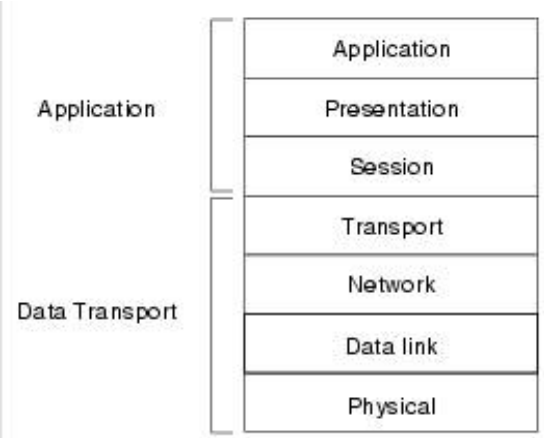


Figura 40 – Abordagem cliente-servidor

Figura 41 – Modelo OSI – Camadas de aplicação e de transporte [22]

Algumas das questões tipicamente apontadas como fragilidades deste tipo de arquitectura, tais como a sobrecarga da largura de banda e dependência

efectiva do servidor provaram, no decorrer deste trabalho, ser de importância menor.

Por um lado, a largura de banda das redes locais actuais e o tipo de situação em que um sistema destes pode operar dificilmente gerará congestionamento.

Dada a impossibilidade física do controlo de um manipulador ser efectuado em simultâneo por mais do que um cliente, implica que não faz sentido que um servidor deste tipo, na essência um servidor de operações de um manipulador industrial, possa, efectivamente, ser acedido por um numero tal de clientes que se gere congestionamento.

A implementação desta arquitectura dividiu-se entre dois grandes grupos das camadas do modelo *OSI*, camadas de aplicação, e, num nível inferior, camadas de transporte [Figura 41].

Para as camadas de aplicação foi desenvolvido um servidor em *KAREL* do lado do controlador e uma aplicação cliente do lado do *PC*. Quanto às camadas de transporte foi seleccionada uma implementação de *sockets* de *Berkeley* sobre *TCP/IP* cujo funcionamento se expõe no diagrama da Figura 42.

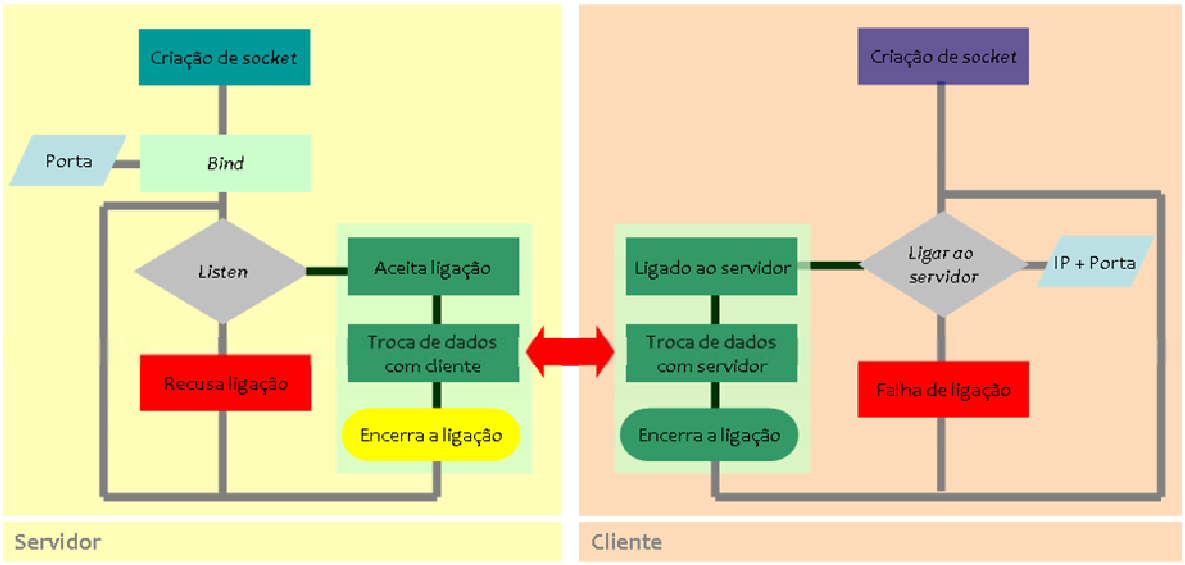


Figura 42 – Etapas e procedimentos da arquitectura cliente servidor utilizada

A escolha recaiu sobre o protocolo *TCP/IP* essencialmente e para além das vantagens de índole técnica apresentadas atrás pela versatilidade que oferece, nomeadamente na portabilidade da implementação do conceito de cliente para diversas plataformas, nomeadamente para plataformas móveis. Este aspecto é discutido no capítulo relativo a trabalho futuro.

4.4. Implementação

Nos pontos seguintes descreve-se, sumariamente, a implementação de software efectuada no decorrer deste trabalho.

O software desenvolvido divide-se em dois grupos, o desenvolvimento da aplicação servidora e da aplicação cliente, sendo que o primeiro é o foco deste trabalho.

4.4.1.Desenvolvimento de um servidor

Foi desenvolvida uma aplicação servidora para o controlador do manipulador, aplicação esta responsável por disponibilizar um conjunto de serviços a dispositivos remotos [Figura 43 / Figura 44].

Esta aplicação é, por um lado, um *parser* de uma linguagem proprietária também desenvolvida no decorrer deste trabalho e por outro um gestor de operações ao nível do controlador, capaz de gerir o fluxo dos pedidos remotos e de responder convenientemente ao dispositivo remoto.

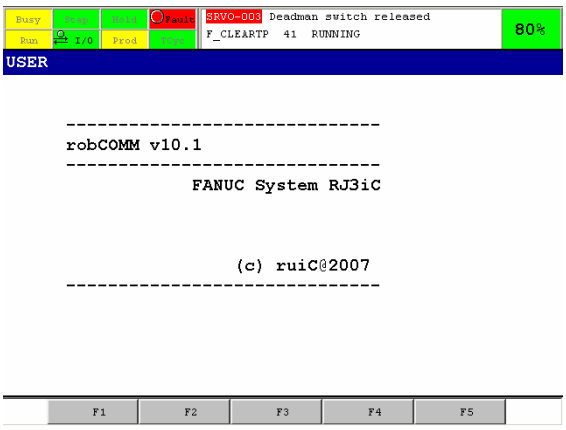


Figura 43 – robCOMM© – Ecran inicial

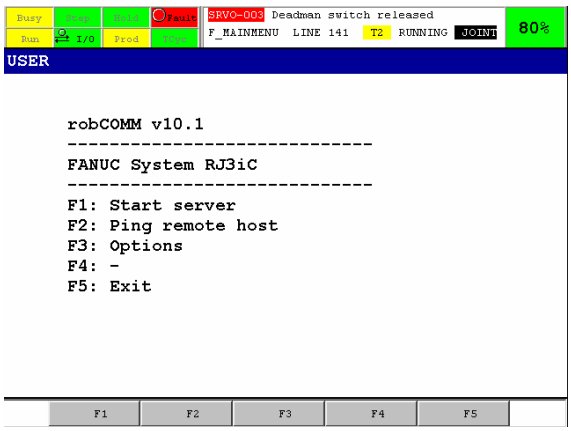


Figura 44 – robCOMM© - Menu inicial

Esta aplicação foi completamente desenvolvida sobre a linguagem *KAREL* e corre sobre o sistema operativo do controlador *FANUC*.

O servidor foi desenvolvido numa perspectiva mono-cliente. Os clientes poder-se-ão ligar e desligar livremente admitindo que apenas um cliente está ligado de cada vez. Esta implementação poderá evoluir para uma plataforma multi-cliente, tal como é discutido mais à frente no capítulo relativo ao trabalho futuro.

A implementação de *sockets* seleccionada para este trabalho, implica que o cliente escute continuamente uma porta até que um cliente remoto a ele se ligue.

4.4.2. Desenvolvimento de um cliente

Foi desenvolvida uma aplicação cliente para *PC* para correr num dispositivo remoto, enviando instruções à aplicação servidora solicitando a execução de tarefas [Figura 45 / Figura 46].

Esta aplicação foi desenvolvida em *C#* sobre a plataforma *Microsoft® .NET Framework.*

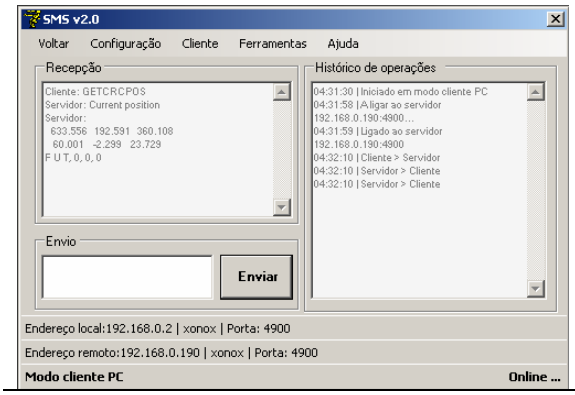


Figura 45 – robCOMM© Cliente – Aspecto da interface gráfica

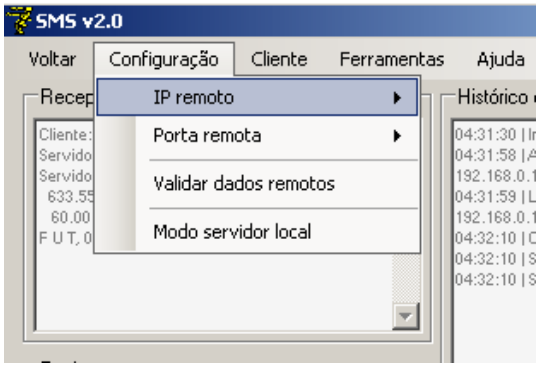


Figura 46 – robCOMM© Cliente – Exemplo de menu de configuração

Dado que os clientes se ligam através de *sockets TCP/IP*, poderão ser implementados e correr sobre qualquer plataforma que suporte *sockets* (*Windows, Linux, Mac*). Na verdade, a aplicação cliente foi implementada como um cliente de *sockets TCP/IP* não como foco e objecto deste trabalho mas apenas como forma de demonstrar o conceito, validar o funcionamento da aplicação servidora e exemplificar a sua implementação numa qualquer linguagem de programação por objectos.

4.4.3. Desenvolvimento de uma linguagem

Para a concretização deste trabalho, tornou-se óbvio que seria necessário desenvolver um protocolo que permitisse que quer o cliente, o dispositivo remoto, quer o servidor, falassem a mesma linguagem, tornando possível a comunicação. Assim, a necessidade do desenvolvimento de um simples protocolo, cedo se tornou na necessidade de uma linguagem dado que se verificou que as instruções remotas teriam que ser caracterizadas e convenientemente parametrizadas.

Foi, portanto, desenvolvida uma linguagem de programação a que se chamou *robCOMM language* no seguimento do nome da aplicação servidora.

Esta linguagem especifica um conjunto de instruções devidamente caracterizadas que indicam ao servidor que tipo de tarefa realizar [Figura 47].

string **MOVTOCPOS** x y z w p r Flip Cfg2 Cfg3 Turn1 Turn2 Turn3 WaitMode

Figura 47 – Exemplo da linguagem robCOMM

5. robCOMM©

5.1. Fundamentos e enquadramento

Tal como descrito no capítulo anterior, foi desenvolvida uma aplicação servidora a que se chamou *robCOMM*, de *robotic communication,* para o controlador do manipulador responsável por disponibilizar um conjunto de serviços a dispositivos remotos.

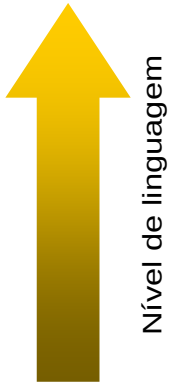
As soluções existentes para resolver este problema implicam, tipicamente, a aquisição de pacotes auxiliares de software quer para o controlador do robot quer para o *PC*. Este tipo de solução, além do custo de aquisição e licenciamento, implica um conjunto de desvantagens tais como a dependência da plataforma do software *PC* e a limitação ás funcionalidades pré-implementadas pelo fabricante. A solução proposta através do *robCOMM* é completamente independente da plataforma *PC*, não exige qualquer módulo avançado de *software* específico para o controlador e é escalável ao nível do utilizador.

O *robCOMM* impõe também uma camada de abstracção interessante na utilização de um manipulador industrial já que o utilizador remoto não precisa de conhecer as especificações do equipamento robotizado nem tão pouco as linguagens de programação nativas para o utilizar, o que confere boas possibilidades de independência de plataforma. É pois possível que o programador de aplicações da célula de fabrico, conhecendo apenas a linguagem utilizada pelo *robCOMM* e o endereçamento do equipamento na rede, possa usufruir do conjunto alargado de funcionalidades implementadas.

5.2. Arquitectura e definições

Esta aplicação servidora foi desenvolvida numa lógica cliente-servidor, e implementa *sockets TCP/IP* para realizar a comunicação.

Para melhor compreensão dos capítulos seguintes, é importante definir as seguintes palavras-chave, sobre as quais assentam conceptualmente a implementação do *robCOMM* [Tabela 3].



Tarefa
Acção ou conjunto de acções a desempenhar pelo sistema
Materialização real de um serviço

Serviço
Funcionalidade disponibilizada pelo servidor a um dispositivo remoto
Implementação em software de tarefas reais

Instrução
Serviço devidamente caracterizado pelos respectivos parâmetros

Comando
Implementação em linguagem de programação de um serviço

Parâmetros
Caracterizadores de um serviço

Tabela 3 – Tarefa, serviço, instrução e parâmetros

É também importante definir o conceito utilizado para serviços síncronos e assíncronos e explicitar a sua diferença. Assim, entende-se por serviço síncrono um serviço que só retorna o resultado da operação que lhe está associada após a execução integral do mesmo, ou seja, a execução do servidor aguardará que a tarefa solicitada seja cumprida antes de continuar a sua execução e retornar o resultado.

Entende-se por serviço assíncrono, um serviço que retorna o resultado da operação logo após a tarefa ter sido iniciada, ou seja, a execução do servidor não aguardará pela completa execução da tarefa para continuar a execução.

A título de exemplo um serviço de movimento básico pode, mediante um dos seus parâmetros, ser executado em modo síncrono ou assíncrono, o que, neste caso, significará que assim que o servidor receba a instrução de movimento ordenará ao controlador que inicie o movimento da unidade mecânica mas, em modo síncrono, aguardará que o manipulador chegue à posição de destino para retornar o resultado da operação, enquanto que em modo assíncrono retornará de imediato o resultado da operação.

Esta funcionalidade é extremamente útil e acrescenta um enorme potencial à aplicação desenvolvida pois permite que o servidor continue a aceitar solicitações, colocando as instruções numa pilha de tarefas a executar em modo FIFO. Para o caso de serviços relacionados com movimento, permite, por exemplo, que um dado movimento em execução seja interrompido antes do seu destino e, por exemplo, reorientado.

5.3. Exemplo de aplicação

De seguida, descreve-se um exemplo de aplicação. Foi iniciada uma ligação com o servidor *robCOMM* [Figura 48] e foi enviada a instrução *GETCRCPOS* a partir da aplicação cliente. Esta instrução foi validada e executada pelo servidor, que devolveu à aplicação cliente a informação solicitada.

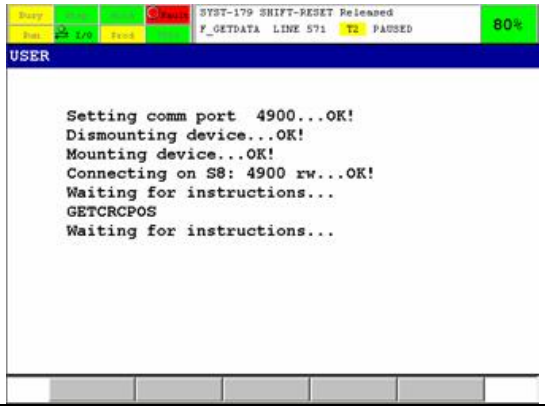


Figura 48 – robCOMM© – Ligação e execução de uma instrução

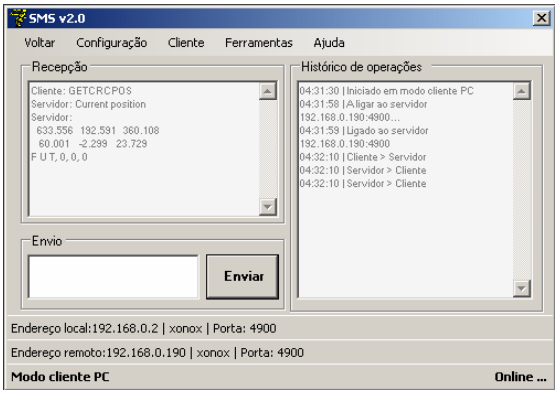


Figura 49 – robCOMM© Cliente - Instrução e recepção de resultados

Neste caso, a instrução em causa, solicita ao servidor a posição actual do *TCP* do robot em coordenadas cartesianas, informação esta que foi recebida com sucesso [Figura 49].

5.4. Serviços disponíveis

Os serviços disponibilizados pelo *robCOMM* agrupam-se em 4 grupos: serviços de movimento, serviços de manipulação de dados, serviços de cálculo e serviços de execução de programas.

De seguida descrevem-se os grupos de serviços e o tipo de funcionalidade que disponibilizam.

5.4.1. Serviços de movimento

Os serviços de movimento são, porventura, os serviços mais relevantes e significativos na utilização de um manipulador industrial. Em última análise, aquilo que se espera de um equipamento deste género é, de facto, a execução de tarefas que incluam movimento e, como tal, este tipo de serviço foi aquele que mais relevância teve no desenvolvimento deste trabalho.

As aplicações reais deste tipo de serviço são inúmeras. A título de exemplo, uma aplicação de manipulação de objectos em que se pretende que a informação relativa ao posicionamento dos objectos a manipular seja obtida a partir de um sistema de visão artificial, faria uso de um destes serviços, enviando para o servidor uma instrução de movimento, fazendo passar como parâmetros a posição cartesiana e orientação do objecto [Figura 50].

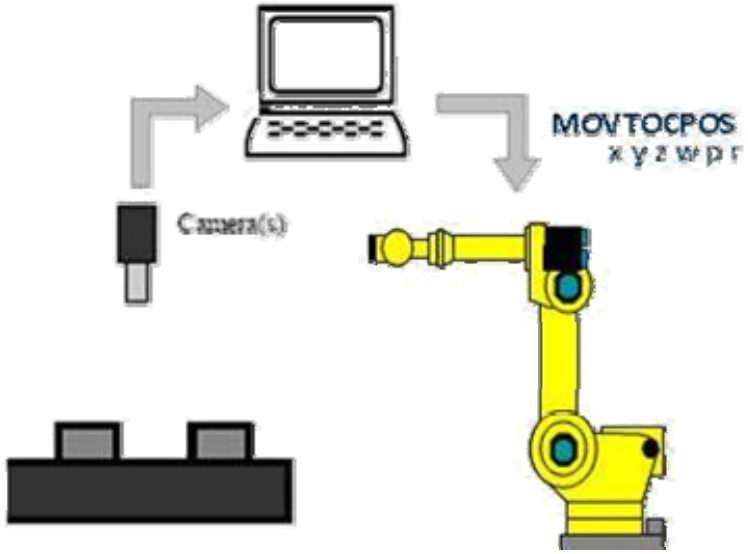


Figura 50 – Implementação industrial típica de um sistema de visão

Um outro exemplo, este mais elaborado, seria uma aplicação de polimento em que se pretende gerar uma trajectória a cumprir pelo manipulador, composta por um qualquer número de nodos, trajectória esta gerada a partir uma aplicação a correr num *PC* cujo *input* poderia ser, por exemplo, um modelo *CAD*.

5.4.2. Serviços de manipulação de dados

A implementação deste tipo de serviços tornou possível, por um lado, a troca de valores de registos, quer numéricos quer de posição, com o controlador e por outro a interacção com os dispositivos de I/O disponíveis no controlador.

5.4.3. Serviços de cálculo

É um exercício recorrente em aulas de robótica industrial, desafiar os alunos a calcular as cinemáticas de um manipulador industrial. Tipicamente são utilizados modelos e algoritmos sistemáticos como forma de descrever a relação cinemática entre os elos do manipulador. Um dos métodos mais utilizados é a notação de *Denavit-Hartenberg* em que a posição e orientação dos corpos rígidos é definida numa matriz 4x4 [Figura 51 / Figura 52].

Esta funcionalidade é intrínseca do próprio manipulador na medida em que é exactamente como base na determinação das cinemáticas directas e inversas que é possível posicionar o manipulador.

Pretendeu-se, portanto, disponibilizar estas funcionalidades a um utilizador remoto como ferramenta de verificação do modelo teórico utilizado, ou seja, poder-se-á utilizar um qualquer método teórico, tal como é a notação de *Denavit-Hartenberg,* para determinar as posições de junta de cada eixo e comparar os resultados com aqueles calculados pelo controlador do manipulador.

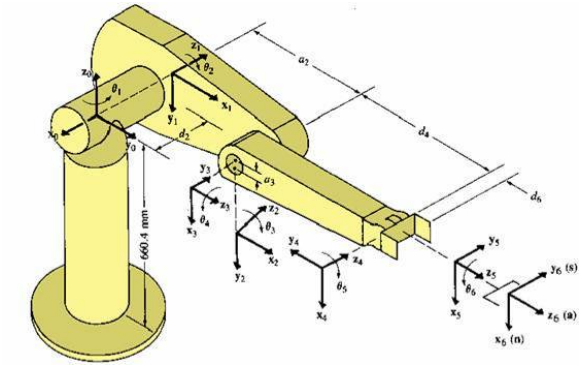


Figura 51 – Convenção D-H [18]

$$ {}^{n-1}T_n = \begin{pmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & d_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & d_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & l_n \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Figura 52 – Mariz transformação de sistema de coordenadas [17]

5.4.4. Serviços de execução de programas

A funcionalidade de execução remota de programas *TP* em conjugação com os serviços de manipulação de registos numéricos e de posição, geram uma enorme versatilidade já que é possível criar programas *TP*-tipo que utilizem registos para obter informação fornecida remotamente.

Por outro lado, as funcionalidades de execução de programas *TP* dão à aplicação uma versatilidade interessante. Existem um sem numero de opções disponibilizadas pelo fabricante, de tal forma que a tarefa de implementar todas as possibilidades no servidor seria, além de extremamente morosa, provavelmente inútil no sentido em seria sempre possível que o fabricante lançasse um nova funcionalidade não disponível à data de criação do servidor.

Assim, torna-se possível construir programas *TP* que poderão fazer uso de todas as funcionalidades disponíveis no controlador e torná-las disponíveis à aplicação servidora de uma forma transparente.

A título de exemplo, imagine-se uma nova opção de movimento não contemplada e portanto não disponível enquanto serviço, mas que, em última análise, faz mover o robô do ponto A para o ponto B com um conjunto de características tornadas possíveis de especificar pela tal opção. Seria apenas necessário criar um simples programa *TP* com 2 linhas de movimento utilizando a tal opção, e cujos pontos fossem quaisquer registos de posição. De seguida seria possível executar o tal programa *TP* criado e cujos pontos seriam registos também eles possíveis de especificar remotamente.

Esta funcionalidade reveste-se, pois, de importância na medida em que potencia o alargamento quase ilimitado das funcionalidades da aplicação desenvolvida.

5.4.5. Gestão de erros

Todos as rotinas implementadas possuem um validação interna que assegura que, na eventualidade da instrução solicitada não puder por qualquer razão ser cumprida, é gerado um código de erro que é passado para o dispositivo cliente e que, dependendo do serviço, pode ou não informar sobre o problema que lhe deu origem.

Esta gestão de erros funciona a dois níveis. Num primeiro nível é validada a disponibilidade do serviço solicitado e é verificado se a sintaxe dos parâmetros do mesmo se encontram na especificação do protocolo.

Num segundo nível e depois de validado o comando, o servidor tenta executar o serviço associado avaliando o seu resultado. Se por qualquer razão não for possível executá-lo com sucesso, é passado um código de erro que é em alguns casos traduzido e, em última análise, devolvido ao dispositivo remoto afim de informar o utilizador do estado de execução da instrução.

A título de exemplo, no caso de uma instrução de movimento simples e além de validado a sintaxe do comando e respectivos parâmetros, é averiguado se a posição destino se encontra dentro do volume de trabalho e se é atingível pelo manipulador. Com isto pretende-se evitar que o utilizador só se aperceba que a posição destino solicitada não é válida depois do comando de movimento já ter sido despoletado, dando origem a um erro de sistema no controlador, cujo rearme implicaria necessariamente operar a consola *TP* pondo em causa a robustez da plataforma.

6. A linguagem *robCOMM*©

6.1. Fundamentos

As linguagens de programação, tal como as linguagens humanas, são definidas através de um conjunto de regras morfológicas, que determinam a sua estrutura, e semânticas que determinam o significado. São utilizadas para facilitar a comunicação homem-máquina, a manipulação de dados e a explicitação de algoritmos.

O objectivo deste trabalho passa, também, por desenvolver uma forma de controlar o comportamento do dispositivo robótico utilizado.

Pretendeu-se criar uma linguagem cujo conjunto de regras pudesse definir comportamentos do equipamento, tornando possível o seu controlo a partir de dispositivos remotos. Assim, a implementação do *robCOMM*, enquanto servidor, implicou também o desenvolvimento de uma linguagem que implementasse os serviços concebidos e possibilitasse a interacção de dispositivos remotos.

Foi, então, desenvolvida a *robCOMM language*, linguagem de implementação do servidor *robCOMM* [Tabela 4].

MOVTOCPOS	Movimento cartesiano
MOVTOJPOS	Movimento em junta
GETCRCPOS	Posição actual do TCP
GETCRJPOS	Configuração actual das juntas
CHECKCPOS	Verifica posições cartesianas
CHECKJPOS	Verifica configurações de junta
GETDIRKIN	Calcula cinemática directa
GETREVKIN	Calcula cinemática inversa
GETREG	Obtêm valores de registos
SETREG	Escreve em registos
SETPATH	Compõe um caminho
MOVTHPTH	Move ao longo de um caminho
LISTTPP	Obtêm a lista de programas TP
RUNTPP	Executa programas TP
GETDIO	Devolve o estado de I/O digitais
GETAIO	Devolve o estado de I/O analógicas
SETDIO	Actua I/O digitais
SETAIO	Actua I/O analógicas
SETTOOLFRM	Especifica uma Tool frame
SETUSERFRM	Especifica um User frame
SETLSPEED	Especifica velocidade linear
MOTIONSTOP	Para o movimento
STOPSERV	Termina o servidor

Tabela 4 – Resumo de intruções robCOMM language

6.2. Instruções disponíveis

6.2.1. MOVTOCPOS e MOVTOJPOS

Sempre que se pensa num manipulador, pensa-se em movimento, fazendo do conjunto de instruções disponíveis, as mais emblemáticas deste trabalho. Permitem instruir o servidor a efectuar movimentos de tipo variado e utilizando como destinos posições cartesianas ou configurações angulares das juntas.

Relativamente à instrução *MOVTOCPOS*, é necessário fornecer os valores para as coordenadas cartesianas e as respectivas orientações [Figura 53].

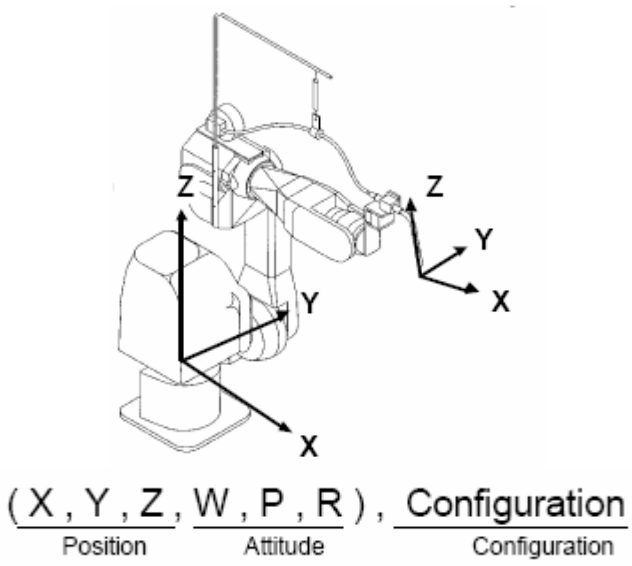


Figura 53 – Composição de ponto cartesiano [7]

No entanto, no espaço cartesiano, é também necessário especificar qual a configuração desejada para uma dada posição [Figura 54], configuração esta que é informação suficiente para ultrapassar qualquer situação de redundância [Figura 55].

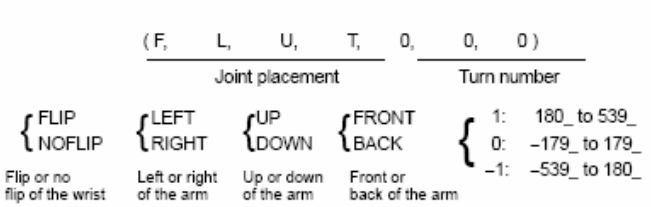


Figura 54 – Configuração [7]

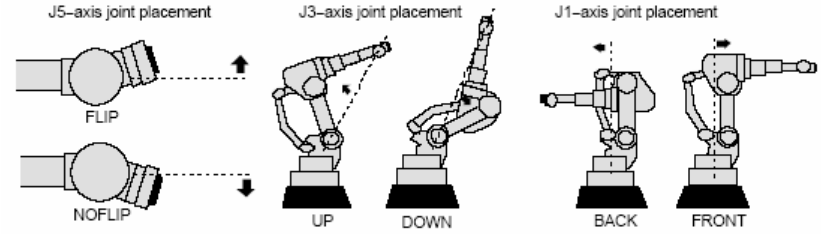


Figura 55 – Redundâncias [7]

Relativamente à instrução *MOVTOJPOS*, é necessário apenas fornecer os valores angulares para cada junta [Figura 56].

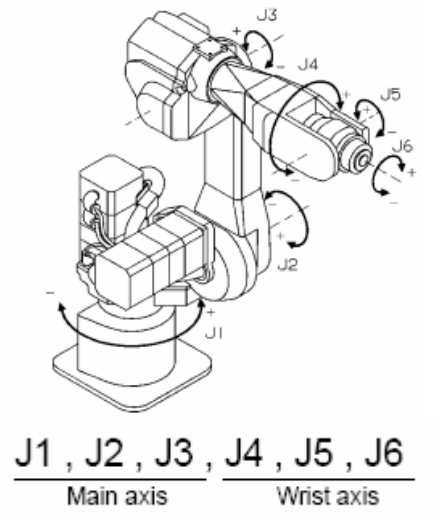


Figura 56 – Composição de configuração de juntas [7]

MOVTOCPOS X Y Z w p r Cfg1 Cfg2 Cfg3 Turn1 Turn2 Turn3

MOVTOJPOS J1 J2 J3 J4 J5 J6

input arg	***descrição***	***tipo***
X Y Z w p r	Posição cartesiana	String
Cfg1 Cfg2 Cfg3	Configuração	String
Turn1 Turn2 Turn3	Voltas	String
J1 J2 J3 J4 J5 J6 J7	Posição angular das juntas	String

output	***Descrição***	***tipo***

Tabela 5 – Instruções movToCPos / movToJPos

Tabela 6 – Parâmetros movToCPos / movToJPos

É, também, possível pré configurar o servidor para efectuar movimentos em junta por oposição ao movimento linear pré definido.

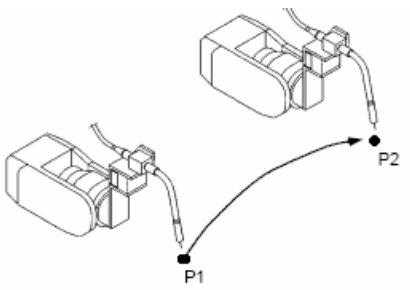


Figura 57 – Movimento em junta [7]

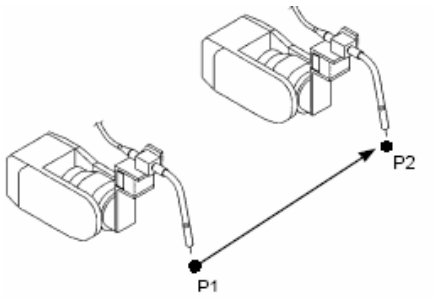


Figura 58 – Movimento linear [7]

6.2.2. GETCRCPOS e GETCRJPOS

Estes serviços permitem obter a posição actual do *TCP* do manipulador quer em coordenadas cartesianas, *GETCRCPOS*, quer em posições angulares de junta, *GETCRJPOS*.

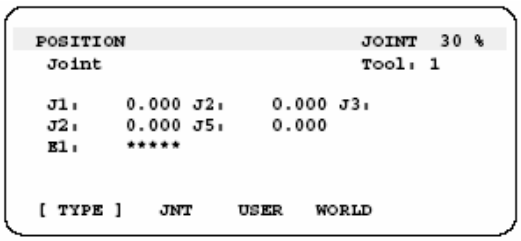


Figura 59 – Ecran TP com posição de juntas [7]

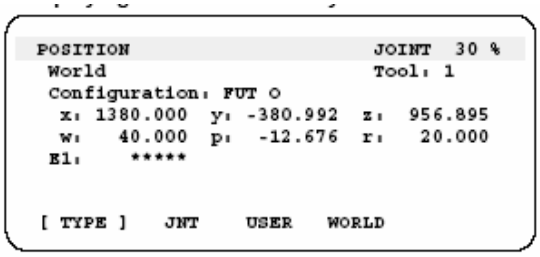


Figura 60 – Ecran TP com posição cartesiana [7]

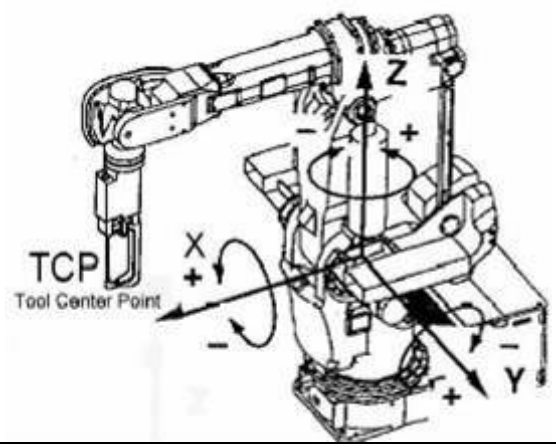


Figura 61 – Posição actual do TCP [7]

De salientar que as coordenadas cartesianas são relativas ao sistema de coordenadas base do manipulador muito embora seja possível configurar o servidor de modo a que as coordenadas cartesianas sejam relativas a qualquer outro sistema de coordenadas.

	input arg	***descrição***	***tipo***
GETCRCPOS **GETCRJPOS**			

output	***Descrição***	***tipo***
X Y Z w p r	Posição cartesiana	String
J1 J2 J3 J4 J5 J6 J7	Posição angular das juntas	String

Tabela 7 – Instruções getCrcPos / getCrjPos

Tabela 8 – Parâmetros getCrcPos / getCrjPos

Para estes comandos não existem argumentos de entrada e o dado de saída é uma cadeia de texto com os valores numéricos da posição actual.

6.2.3. CHECKCPOS e CHECKJPOS

Este serviço implementa a validação de posições ou configurações de juntas sobre o volume de trabalho do manipulador.

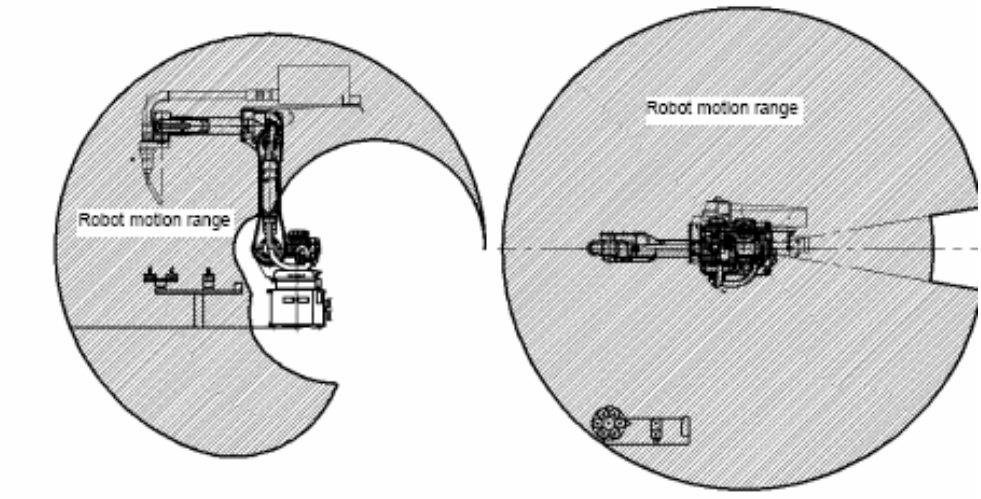


Figura 62 – Exemplo de volume de trabalho de um manipulador industrial [7]

Esta validação é efectuada sempre que se executa um comando de movimento individual, *MOVTOCPOS* ou *MOVTOJPOS*, mas não se o comando de movimento for um movimento composto através de um caminho, *MOVTHPTH*. Assim, dependendo do objectivo ou da necessidade de garantir que uma dada posição destino está dentro do volume de trabalho e é alcançável pelo manipulador, é importante executar este serviço fornecendo as coordenadas cartesianas e orientações da posição a testar, *CHECKCPOS*, ou a configuração de juntas, *CHECKJPOS*.

De salientar que a configuração utilizada para o cálculo é a configuração actual do manipulador, ou seja, dever-se-á mover previamente o robô para o quadrante da configuração sobre a qual se pretende testar a validade de uma dada posição cartesiana ou configuração de juntas.

CHECKCPOS X Y Z w p r
CHECKJPOS J1 J2 J3 J4 J5 J6

input arg	*descrição*	*tipo*
X Y Z	Posição cartesiana a verificar	Real
w p r	Orientação da posição a verificar	Real
J1 J2 J3 J4 J5 J6	Configuração de juntas a verificar	Real

Tabela 9 – Instrução checkCPos /checkJPos

output	*Descrição*	*tipo*
CHECKCPOS	Validação	String (OK / NOK)
CHECKJPOS	Validação	String (OK / NOK)

Tabela 10 – Parâmetros checkCPos / checkJPos

6.2.4. GETDIRKIN e GETREVKIN

Estes serviços permitem obter as soluções de cinemática directa para uma dada configuração de juntas ou de cinemática inversa para uma dada posição cartesiana [Figura 63].

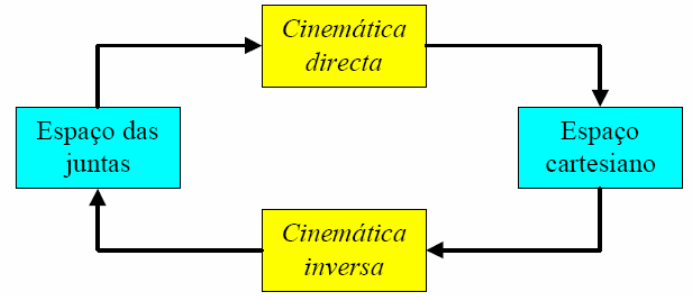


Figura 63 – Cinemática directa e inversa [17]

GETDIRKIN J1 J2 J3 J4 J5
GETREVKIN X Y Z w p r

input arg	*descrição*	*tipo*
X Y Z w p r	Posição e orientação cartesiana	Real
J1 J2 J3 J4 J5 J6	Configuração de juntas	Real
output	***Descrição***	***tipo***
GETDIRKIN	Posição cartesiana	String
GETREVKIN	Configuração de juntas	String

Tabela 11 – Instruções getdirkin / getrevkin

Tabela 12 – Parâmetros getdirkin / getrevkin

6.2.5. GETREG

Este serviço permite obter os valores de registos. É possível obter os valores de registos de posição e de registos numéricos mediante a especificação do tipo e número do registo cujos dados se pretende obter.

GETREG regType
regIndex

input arg	*descrição*	*tipo*
regType	Tipo de registo a obter	Real
regIndex	Numero do registo a obter	Real
output	***Descrição***	***tipo***
GETREG	Valor do registo	String

Tabela 13 – Instrução getReg

Tabela 14 – Parâmetros getReg

6.2.6. SETREG

Este serviço permite atribuir valores a registos. É possível atribuir valores a registos de posição e registos numéricos mediante a especificação do tipo e número do registo cujos dados se pretende obter.

LISTTP
RUNTP tpPrg

input arg	*descrição*	*tipo*
tpPrg	Nome de programa TP a executar	String
output	***Descrição***	***tipo***
LISTTP	Lista de programas	String
RUNTP	Status	String

Tabela 15 – Instruções listTp / runTp

Tabela 16 – Parâmetros listTp / runTp

6.2.7. SETPATH e MOVTHPTH

Estes serviços permitem configurar um dado caminho, *ie* trajectória, através de um conjunto de nodos, *ie* pontos de passagem, e, posteriormente, mover o TCP do manipulador através desse mesmo caminho.

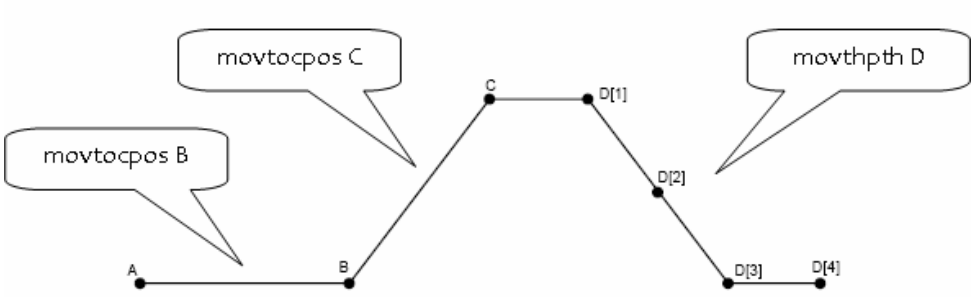


Figura 64 – Instrução movThPth

Relativamente ao *SETPATH*, é possível especificar a posição cartesiana e respectivas orientações de cada nodo, mas é de notar que a configuração utilizada ao longo do movimento é a configuração do manipulador na altura da execução do movimento.

Quanto à instrução *MOVTHPTH*, é possível especificar a velocidade a utilizar para o movimento através do caminho. Este valor deve ser considerado em mm/s com o máximo de 2000 mm/s e é de notar que passa a ser utilizado como o valor de velocidade utilizado para quaisquer tipos de movimentos seguintes.

SETPATH X Y Z w p r
MOVTHPTH pathSpeed

input arg	*descrição*	*tipo*
X Y Z	Posição cartesiana a verificar	Real
w p r	Orientação da posição a	Real

	verificar	
pathSpeed	Velocidade	Real

output	*Descrição*	*tipo*
SETPATH	Status	String
MOVTHPTH	Status	String

Tabela 17 – Instrução setPath / movThPth

Tabela 18 – Parâmetros setPath / movThPth

6.2.8. LISTTPP e RUNTPP

Estes serviços permitem obter uma lista dos programas *TP* disponíveis no sistema de ficheiros do controlador e executar um qualquer programa.

LISTTP
RUNTP tpPrg

input arg	*descrição*	*tipo*
tpPrg	Nome de programa TP a executar	String
output	***Descrição***	***tipo***
LISTTP	Lista de programas	String
RUNTP	Status	String

Tabela 19 – Instruções listTp / runTp

Tabela 20 – Parâmetros listTp / runTp

O comando *LISTTP*, devolve a lista de programas disponível e não dispõe de qualquer argumento de entrada.

Quanto ao comando *RUNTP*, o único parâmetro de entrada é o nome do programa *TP* que se pretende executar. Devolverá o estado de execução do comando como *OK* se tiver sido possível executar o programa solicitado e como *NOK* se o programa não existir ou não estiver disponível. É possível configurar o servidor de modo a executar este comando de modo síncrono, em que a resposta só é devolvida após a completa execução do programa *TP*, ou de modo assíncrono, em que a resposta é devolvida logo depois do programa ter sido iniciado.

6.2.9. GETDIO e GETAIO

Estes serviços permitem obter o estado ou o valor analógico de entradas digitais ou analógicas respectivamente.

GETDIO iold
GETAIO iold

input arg	*descrição*	*tipo*
iold	Id da I/O	Inteiro
output	***descrição***	***tipo***
GETDIO	Estado digital	String (1/0)
GETAIO	Valor analógico	String (0 ... 10000)

Tabela 21 – Instruções getAio / getDio

Tabela 22 – Parâmetros getAio / getDio

O único parâmetro de entrada é o identificador da entrada de que se pretende obter o respectivo estado ou valor enquanto que o dado de saída é um estado digital se se tratar de uma entrada digital mas, relativamente a entradas analógicas, esta funcionalidade utiliza uma razão proporcional com a resolução de 1/10000 para especificar o respectivo valor. A real amplitude do valor dependerá da forma como a entrada em questão se encontre configurada mas pretendeu-se que esta questão não fosse tornada transparente ao utilizador remoto que não precisará de conhecer a configuração do hardware para poder obter um valor analógico num intervalo conhecido.

6.2.10. SETDIO e SETAIO

Estes serviços permitem atribuir um estado ou um valor analógico a saídas digitais ou analógicas respectivamente.

SETDIO ioId ioBool
SETAIO ioId ioVal

input arg	***descrição***	***tipo***
ioId	Id da I/O	Inteiro
ioBool	Estado digital	1 / 0
ioVal	Valor analógico	Inteiro (0 … 10000)

output	***descrição***	***tipo***
SETDIO	Status	string
SETAIO	Status	String (OK/NOK)

Tabela 23 – Instruções setAio / setDio

Tabela 24 – Parâmetros setAio / setDio

Os argumentos são o identificador da saída e o estado digital ou valor analógico a impor enquanto que o retorno é o estado de execução do comando. Conforme a configuração do servidor, retornará *OK* ou *NOK* consoante a operação tenha sido realizada com ou sem sucesso.

De notar que se se tratar de uma saída analógica, esta funcionalidade utiliza uma razão proporcional com a resolução de 1/10000 para especificar o respectivo valor. A real amplitude do valor dependerá da forma como a saída em questão se encontre configurada. Pretendeu-se que esta questão não fosse tornada transparente ao utilizador remoto que não precisará de conhecer a configuração do hardware para poder impor um valor analógico num intervalo conhecido.

6.2.11. SETTOOLFRM

Este serviço permite configurar o *TCP* do manipulador.

Esta é uma funcionalidade essencial na operação de manipuladores industriais e um parâmetro de importância crucial pois é sobre este ponto que as cinemáticas são calculadas e todos os pontos referenciados.

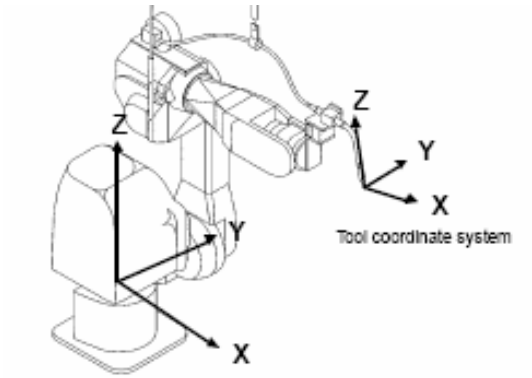


Figura 65 – Instrução setToolFrm [7]

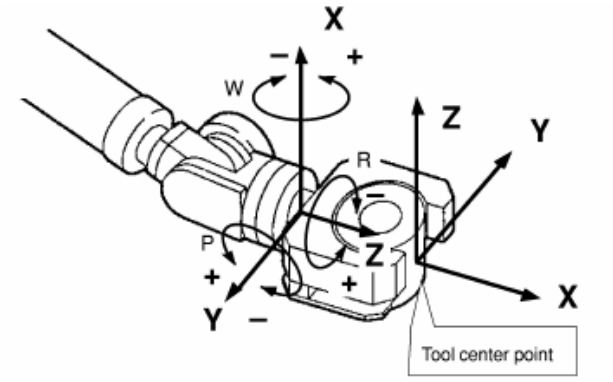


Figura 66 – TCP - Tool Center Point [7]

SETTOOLFRM toolId X Y Z w p r

input arg	*descrição*	*tipo*
toolId	Id da ferramenta	Inteiro
X Y Z	Offset cartesiano da ferramenta relativamente ao TCP_0	Real
w p r	Orientação da ferramenta relativamente ao sistema de coordenadas World	Real
output	***descrição***	***tipo***
SET^TOOLFRM	Status	String

Figura 67 – Instrução setToolFrm

Figura 68 – Parâmetros setAio / setDio

6.2.12. SETSPEED

Este serviço permite especificar a velocidade para movimentos simples, lineares ou de junta. O valor de velocidade especificado é valido até ser sobrescrito por este mesmo comando e será sempre atribuído um valor por defeito reduzido da inicialização da aplicação.

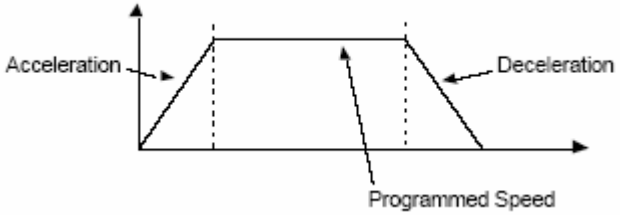


Figura 69 – Diagrama de acelerações [7]

Em movimentos de tipo linear, o valor deve ser considerado em mm/s com o máximo de 2000 mm/s, enquanto que em movimento de junta, o valor de velocidade é calculado como uma razão proporcional com a resolução de 1/2000 entre o valor atribuído e o valor máximo atingível. De notar que o movimento respeitará um perfil de aceleração / velocidade com rampas pré-definidas. [Figura 69].

SETSPEED speed

Figura 70 – Instrução setSpeed

input arg	*descrição*	*tipo*
speed	Velocidade	Real

output	*descrição*	*tipo*
SETSPEED	Status	String

Figura 71 – Parâmetros setSpeed

6.2.13. STOPSERV

Este serviço permite terminar de forma unilateral e definitiva a aplicação servidora, não existindo qualquer possibilidade de recuperar sem que seja utilizada a consola de programação.

STOPSERV

Figura 72 – Instrução stopServ

input arg	*descrição*	*tipo*

output	*descrição*	*tipo*
STOPSERV	Status	String

Figura 73 – Parâmetros stopServ

7. Conclusões e trabalho futuro

7.1. Conclusões

No âmbito deste trabalho foi desenvolvida e validada uma plataforma servidora para um controlador de um manipulador industrial. O valor acrescentado deste trabalho passa pela demonstração do potencial de aplicação do controlo remoto de um manipulador industrial.

Com o *robCOMM* foi conseguido um nível de abstracção que permite que o utilizador remoto não seja obrigado a conhecer as especificações do equipamento robotizado nem as suas linguagens de programação nativas.

A aproximação utilizada envolve uma estratégia baseada no modelo cliente - servidor, em que o controlador do manipulador actua como servidor, disponibilizando serviços a um dispositivo remoto. A implementação do *software* passou por uma implementação de *sockets* de *Berkeley* sobre *TCP/IP* com base no modelo *OSI*. Esta estratégia implica uma total independência de plataforma, ferramentas e linguagens de programação na óptica do programador remoto. A aplicação cliente do servidor *robCOMM* poderá ser desenvolvida como parte integrante duma aplicação de gestão de uma célula de fabrico sem que existam as restrições comuns às ferramentas comercialmente disponíveis.

Conseguiu-se desenvolver uma plataforma capaz de oferecer serviços de vários tipos, com especial relevância para os serviços relacionados com movimento. Com o *robCOMM* provou-se ser possível controlar e monitorizar remotamente um manipulador industrial, disponibilizando serviços de controlo de movimento, manipulação de dados, registos e *I/O*, cálculo de cinemáticas e execução remota de programas, tornando poderosa e simples a sua implantação numa célula real de fabrico.

Verificou-se que este trabalho pode ser muito útil na redefinição do que é uma célula de fabrico convencional, passando um *PC* a desempenhar um papel central e de interligação entre todos os elementos da célula de fabrico

7.2. Trabalho futuro

De seguida são discutidas algumas das evoluções consideradas relevantes e com potencial de evolução para o autor, verificadas no decorrer deste trabalho. Depois de validada a plataforma e o conceito, seria relativamente fácil desenvolver os conceitos desenvolvidos.

7.2.1. Extensão do servidor e da linguagem

A extensão da linguagem como forma de ampliar potencial do equipamento é, sem dúvida, a tarefa mais significativa.

Os serviços disponibilizados pelo servidor deverão ser estendidos até que se considere que a funcionalidade do equipamento está completamente explorada e a implementação. A funcionalidade dos serviços já disponíveis deve ser questionada e melhorada. São disso exemplo os serviços de leitura e escrita de registos visto que foram implementados apenas os mais relevantes e aqueles disponíveis em

todas as variantes do sistema operativo dos controladores FANUC, registos de posição e registos numéricos, mas poderá ser interessante estender esta funcionalidade a todos os registos específicos das variantes de sistemas operativos disponíveis. Seria também interessante tornar mais amplo e versátil o controlo de velocidade do movimento. À data de conclusão deste documento, é possível especificar a velocidade de comandos de movimento mas podia ser facilmente estendido de forma a incluir a velocidade como parâmetro na definição de um nodo, por forma a controlar a velocidade de deslocamento entre nodos de um dado caminho.

Seria também interessante permitir a construção de mais do que um caminho, armazenando a informação dos nodos localmente no controlador, e por outro permitir a edição de um qualquer nodo de um dado caminho, em alternativa a ter que reenviar todo o caminho sempre que se pretenda efectuar uma alteração em apenas um dos nós. Uma hipótese seria a de rearranjar a estruturação da aplicação para que novas instruções pudessem ser adicionadas sem que fosse necessário alterar e recompilar o código do servidor.

7.2.2. Plataforma multi-cliente

Seria interessante evoluir a plataforma servidora para um cenário completamente multi-*socket* / multi-cliente. Esta hipótese não foi explorada no decorrer deste trabalho mas entende-se que seria uma melhoria relevante.

Desta forma seria possível por um lado gerir pedidos remotos de mais do que um cliente e por outro lado permitiria que um cliente remoto pudesse dispor de um *socket* para a comunicação e troca de informação e um outro para gerir um canal de emergência.

O cenário multi-cliente pode ser contemplado na arquitectura cliente-servidor utilizando *sockets TCP/IP* utilizada neste trabalho, recorrendo ao conceito de *multi-threathing* em que uma dada tarefa bifurca sempre que existir a necessidade de executar mais do que uma tarefa em simultâneo.

7.2.2. Portabilidade da aplicação cliente

A implementação de um cliente *robCOMM* em dispositivos móveis além de conceptualmente simples, introduziria uma nova área de aplicabilidade do trabalho desenvolvido. Numa lógica *CIM*, seria interessante, por exemplo, monitorizar a operação de equipamentos robotizados através de *PDA's* por parte dos responsáveis de produção de uma dada instalação fabril.

7.2.3. Encapsulamento de instruções robCOMM

De forma a poder utilizar as funcionalidades de *intellisensing* [29] das plataformas de desenvolvimento actuais, faria sentido encapsular as instruções disponibilizadas pelo *robCOMM* num componente a ser utilizado ao nível do programador em linguagens orientadas por objectos. Seja um componente *activeX* [30] ou mesmo uma classe da mais recente *WFC* [31], este tipo de encapsulamento permitiria uma utilização mais simples e eficaz das potencialidades do *robCOMM*.

8. Referencias

[1] Pires, N., Sá da Costa, J., "Running an Industrial Robot from a typical Personal Computer", 1998, *IEEE International Conference on Electronics, Circuits and Systems, Volume 1, Issue , 1998 Page(s):267 - 270 vol.1*

[2] United Nations, "World Robotics 2004 – Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment", 2004, *United Nations*

[3] Hirchfeld, R.A., Aghazadeh, F., Chapleski, R.C., Survey of robot safety in industry", *The International Journal of Human Factors in Manufacturing, Vol. 3 No. 4, 1993, pp 369-79*.

[4] Young, K., Pickin, C. , "Accuracy assessment of the modern industrial robot", *Industrial Robot: An International Journal – Vol. 27 – N6, 2000, MCB University press*

[5] FANUC Robotics America, Inc., 2003, Robotics SYSTEM R-J3*i*B Controller KAREL Reference Manual, *2003*

[6] FANUC Robotics America, Inc., 2002, FANUC R-J3iB Ethernet Operator Manual, *2002*

[7] FANUC Robotics America, Inc., 2002, FANUC Robot series ARC TOOL Operator's Manual, *2002*

[8] *FANUC Robotics,* 2002*, FANUC M-6iB Series*

[9] FANUC Robotics America, Inc., 2001, FANUC R-J3iB Maintenance Manual for Europe, *2001*

[10] FANUC Robotics America, Inc., 2001, FANUC ARC Mate 100iB/M6iB Maintenance Manual, *2001*

[11] *FANUC Robotics, RJ3iC Training Manual*

[12] IBM, 1998, TCP/IP Tutorial and Technical Overview, *International Technical Support Organization, 1998*

[13] Nordin , Johnny, Persson, Marie, Industrial Ethernet - EtherNet/IP, *Department of Computer Science and Engineering, 2003*

[14] EtherNet/IPBM, 1998, TCP/IP Tutorial and Technical Overview, *International Technical Support Organization, 1998*

[15] Dagalakis, Nicholas G., Industrial Robotics Standards, *National Institute of Standards and Technology, Intelligent Systems Division, Maryland, U.S.A.*

[16] Cederberg, Per 2004, On Sensor-Controlled Robotized One-off Manufacturing*, PhD Thesis, Department of Mechanical Engineering - Lund Institute of Technology, 2004*

[17] Santos, V.S. 2003, Robótica Industrial – Apontamentos Teóricos, *Departamento de Engenharia Mecânica, Universidade de Aveiro, 2003*

[18] K. Fu, R. Gonzalez, C. Lee, Robotics: control, sensing, vision, and intelligence, *Mcgraw-hill, 1987*

[19] ISO Standard 8373:1994, Manipulating Industrial Robots

[20] Lozano-Perez, Tomas, Robot programming, *Procedings of the IEEE, Vol. 71, No. 7, July 1983*

[21] http://en.wikipedia.org/wiki/Programming_language, *Wikipedia*, lido em Julho de 2007

[22] http://en.wikipedia.org/wiki/OSI_model, *Wikipedia*, lido em Julho de 2007

[23] Zimmermsnn, Hubert, The OSI Model of Architecture for Open Systems Interconnection, *IEEE Transactions on communications, Vol. 28, No. 4, April 1980*

[24] http://www.fanucrobotics.com, FANUC Robotics

[25] http://www.abb.com, ABB Asea Brown Boveri Ltd,

[26] http://www.yaskawa.co.jp, Yaskawa Electric Company

[27] http://www.kuka.com, KUKA Roboter GmbH

[28] http://www.iana.org/assignments/port-numbers, Internet Assigned Numbers Authority (IANA), lido em Julho de 2007

[29] http://msdn2.microsoft.com/en-us/library/43f44291(VS.71).aspx, *Microsoft Corporation*, lido em Julho de 2007

[30] http://en.wikipedia.org/wiki/Component_Object_Model, *Wikipedia*, lido em Julho de 2007

[31] http://en.wikipedia.org/wiki/Windows_Communication_Foundation, *Wikipedia*, lido em Julho de 2007

[32] http://msdn2.microsoft.com/en-us/library/, *Microsoft Corporation,* lido em Julho de 2007

Anexos

Anexo A. Unidade FANUC M6iB RJ3iC

M-6*i*B™ Series

Basic Description

The M-6*i*B series is a six-axis, modular construction, electric servo-driven family of robots designed for a variety of industrial applications. Based on its simple and reliable construction, the M-6*i*B series provides sophisticated motion control and consistent performance with high productivity. The SYSTEM R-J3*i*B Controller and easy-to-use HandlingTool application software provide accurate and consistent path performance.

The M-6*i*B series, the latest generation material handling robot, has a compact design with superior motion range and speed. The compact yet flexible design simplifies installation, maximizes reach capability within confined areas and enables high-density installation of robots and peripherals.

M-6*i*B Series, the Solution for:

- Material handling
- Machine load/unload
- Packing
- Material removal
- Dispensing
- Assembly

Benefits

- Features highest motion speeds in its class for maximum performance and productivity.
- Best in its class reach versus stroke ratio.
- Standard, fail-safe brakes on all six axes increase safety, functionality and control.
- Compact design of the servo motors minimizes interference and simplifies installation of the robot system.
- Extremely fast wrist axes reduce air move times, thus improving throughput.

FANUC Robotics

Features

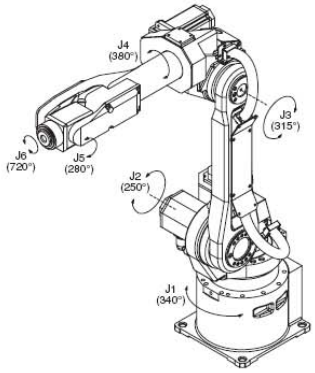
- *i*Pendant, a color, Internet-ready teach pendant for even easier programming and custom cell user interface design.
- Large work envelope with 1,373 mm (M-6*i*B) or 951 mm (M-6*i*B/6S) maximum reach.
- Compact wrist enables the robot to enter into small openings in the workspace.
- Repeatability of +/- 0.08 mm at full speed and full payload within entire work envelope.
- Dedicated pneumatic and electrical (with eight digital inputs and eight digital outputs) connections on J3 axis to simplify user's end-of-arm tooling design, integration and communication.
- Material handling style teach pendant with large LCD screen and ergonomic design offers intuitive control over the process.
- 6 kg payload on faceplate with an additional 12 kg payload on J3 axis.
- Multiple mounting positions include upright, inverted, wall or angle mount with no changes to the mechanical unit.
- Sealed bearings and brushless AC motor drives provide protection and improve reliability.
- RV reducer drivetrain with integral bearings provides rigidity and performance in a compact package.

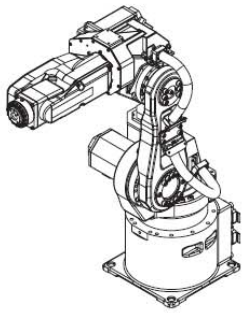
Options

- *i*Pendant is also available with touch screen support.
- J1 stroke modification kit.
- J1 axis 360-degree motion range.
- Various robot connection cable lengths for flexible cabinet placement and optional track rated cables.
- Auxiliary axis packages for integration of peripheral servo-controlled devices.
- Severe dust and liquid protection package for harsh environments.
- Monochrome pendant available.

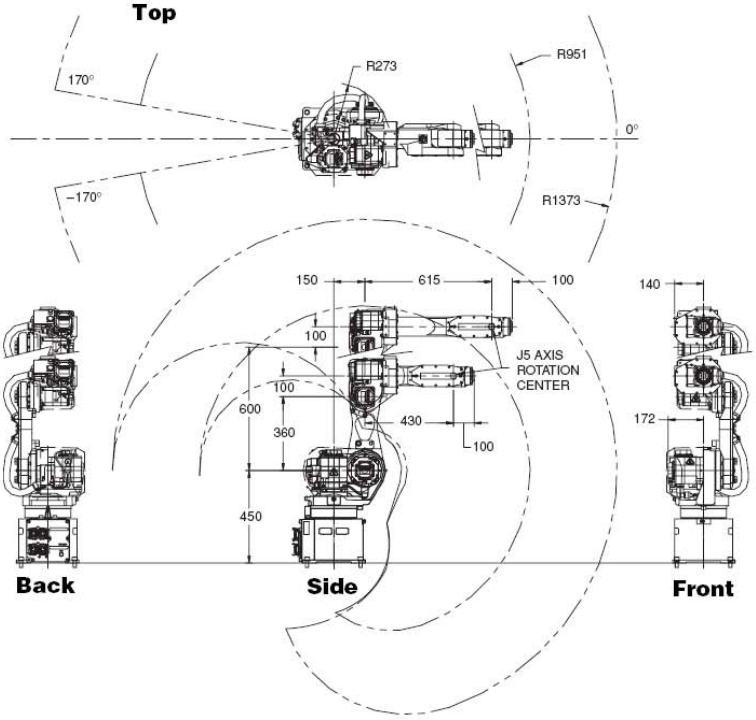
M-6*i*B Series Dimensions

Isometric





Top

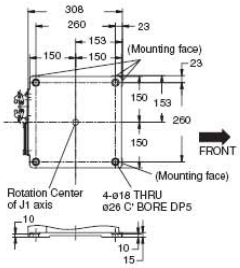


M-6*i*B Series Specifications

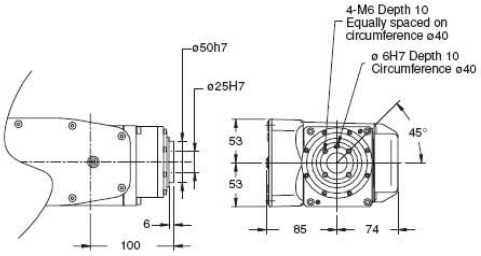
Items		M-6*i*B	M-6*i*B/6S
Axes		6	6
Payload (kg)		6	6
Reach (mm)		1373	951
Repeatability (mm)		±0.08	±0.08
Interference radius (mm)		273	273
Motion range (degrees)	J1	340	340
	J2	250	250
	J3	315	310
	J4	380	380
	J5	280	280
	J6	720	720
Motion speed (degrees/sec.)	J1	150	200
	J2	160	200
	J3	170	260
	J4	400	400
	J5	400	400
	J6	500	720
Wrist moment (kgfcm)	J4	160	160
	J5	100	100
	J6	60	60
Wrist inertia (kgfcms^2)	J4	6.4	6.4
	J5	2.2	2.2
	J6	0.61	0.61
Mechanical brakes		All axes	
Mechanical weight (kg)		138	135
Mounting method		Upright, inverted, wall and angle mount(1)	
Installation environment			
Temperature °C		0 to 45	
Humidity		Normally: 75% or less Short term (within a month): 95% or less No condensation	
Vibration (m/s^2)		4.9 or less	
Payload at axis 3 (kg)		12	

Notes:
(1) Motion range is de-rated for wall and angle mount.

Footprint



Wrist



Note: Dimensions shown in millimeters.

FANUC Robotics America, Inc.
3900 W. Hamlin Road
Rochester Hills, MI 48309-3253
(248) 377-7000
Fax (248) 276-4133

For sales or technical information, call:
1-800-47-ROBOT

marketing@fanucrobotics.com
www.fanucrobotics.com

Charlotte, NC (704) 596-5121
Chicago, IL (847) 898-6000
Cincinnati, OH (513) 754-2400
Los Angeles, CA (949) 595-2700
Toledo, OH (419) 866-0788
Toronto, Canada (905) 812-2300
Montréal, Canada (450) 492-9001
Aguascalientes, Mexico 52 (449) 910-8000
Sao Paulo, Brazil (55) (11) 3619-0599

Anexo B. Controlador FANUC RJ3iC

R-J3iC controller

DESCRIPTION

FANUC Robotics' System R-J3iC Controller uses advanced technology packaged in a proven, reliable and efficient design. Process capability and open architecture features provide intelligence to improve application and motion performance while simplifying system integration. The SYSTEM R-J3iC Controller incorporates FANUC Robotics' unique "plug-in options" concept which allows flexibility for application specific configurations while maintaining a commonality for all users of the system.

***The controller can manage up to 40 axes** that are using FANUC motors (divided in up to 8 groups; one group can control up to 9 axes), such as:*

- *4 robot axes +*
- *4 auxiliary axes groups (arc welding positioners, robotrails, servo hands, grippers...)*

The R-J3iC controller offers enhanced vibration control, this greatly reduces the robot's acceleration and deceleration times leading to reduced cycle times.
The controller provides an integrated 2d vision system to speed-up the installation of vision applications; this can be easily upgraded to provide a 3D Vision system.

... ROBOT DATA SHEET

FEATURES AND BENEFITS

Advanced communication R-J3iC

Simple transfer of robot programs to and from a server !

- **Built-in Ethernet (100 BaseTX)**
  - **2 ethernet ports and integrated software**
  - **TCP/IP connection**
  - **Web server with diagnostic web page.**
  - **FTP (client and/or server)**
  - **Telnet**
  - **Simple Network Time Protocol client (option)**
  - **Socket messaging (with KAREL option)**
  - **Connection to CIMPLICITY, GE FANUC's successful factory automation software**
- **3 serial interface connections (RS232C, RS422) thus allowing**
  - **Floppy Connect**
  - **Printer Connect**
  - **Data Transfer (option)**
  - **Sensor interface (option)**
  - **CRT keyboard (option)**
  - **User specific data transfer using KAREL option**
- **FANUC I/O-link (Master)**
  - **FANUC I/O unit model A (standard)**
  - **FANUC I/O unit model B (option)**
  - **FANUC process I/O (standard for Arc Welding, sealing and other process applications)**
  - **FANUC CNCs, GE FANUC 90-30 PLCs can be connected.**
- **Fieldbus (optional)**
  - **ProfiBUS (Master and slave, separated)**
  - **InterBUS (Master and slave, combined)**
  - **DeviceNET (Master and slave, separated)**
  - **FIP I/O (slave)**
  - **ControlNET (Master and slave, separated)**
  - **FL Net**
  - **Ethernet IP**

USB and PCMCIA interfaces

The R-J3iC controller has a USB interface in the front panel and a PCMCIA interface inside of the control cabinet.

- **This allows fast and economical back-up and restore of data and programs.**

Anexo C. Lista de funções robCOMM

string **MOVTOCPOS** x y z w p r Flip Cfg2 Cfg3 Turn1 Turn2 Turn3 WaitMode

Moves to a given cartesian position

string **MOVTOJPOS** J1 J2 J3 J4 J5 J6

Moves to a given joint configuration

string **GETCRCPOS**

Returns the current TCP cartesian position

string **GETCRJPOS**

Returns the current joint configuration

string **CHECKCPOS** x y z w p r

Verifies if a given cartesian position is in the work envelope and if it is reachable

string **CHECKJPOS** J1 J2 J3 J4 J5 J6

Verifies if a given joint configuration is in the work envelope and if it is reachable

string **GETDIRKIN** J1 J2 J3 J4 J5 J6

Returns the direct kinematics for a given joint configuration

string **GETREVKIN** x y z w p r

Returns the reverse kinematics for a given cartesian position

string **GETREG** regType regIndex

Returns a register data (R or PR regs)

string **SETREG** regType regIndex

Writes data in a register (R or PR regs)

string **SETPATH** x y z w p r (n nodes)

Builds a path given its nodes

string **MOVTHPTH** pathSpeed

Moves along a previously set path

string **LISTTPP**

Returns a TP programs list

string **RUNTPP** tpPrg

Runs a TP program given its name

string **GETDIO** ioIndex

Returns a digital I/O current status

string **GETAIO** ioIndex

Returns an analog I/O current value

string **SETDIO** ioIndex digIoValue

Sets a digital I/O status

string **SETAIO** ioIndex analogIoValue

Sets an analog I/O value

string **SETTOOLFRM** x y z w p r

Sets a Tool frame

string **SETUSERFRM** x y z w p r

Sets a User frame

string **SETLSPEED** linSpeed

Sets linear motion speed for next motion instruction

string **MOTIONSTOP**

Motion all stop

string **STOPSERV**

Terminates server

Anexo D. Cliente de *sockets* em modo síncrono (C# .NET) [32]

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
```

```
public class SocketClient {
    public static void StartClient() {
        // Data buffer for incoming data.
        byte[] bytes = new byte[1024];
        // Connect to a remote device.
        try {
            // Remote endpoint
            IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName())
            IPAddress ipAddress = ipHostInfo.AddressList[0];
            IPEndPoint remoteEP = new IPEndPoint(ipAddress,11000);

            // Create a TCP/IP socket.
            Socket sender = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp );

            // Connect the socket to the remote endpoint
           try {
                sender.Connect(remoteEP);
                Console.WriteLine("Socket connected to {0}",
                    sender.RemoteEndPoint.ToString());

                byte[] msg = Encoding.ASCII.GetBytes("Teste<EOF>");

                // Send data
                int bytesSent = sender.Send(msg);

                // Receive response
                int bytesRec = sender.Receive(bytes);
                Console.WriteLine("Echoed test = {0}",
                    Encoding.ASCII.GetString(bytes,0,bytesRec));

                // Release the socket
                sender.Shutdown(SocketShutdown.Both);
                sender.Close();

            } catch (ArgumentNullException ane) {
            } catch (SocketException se) {
            } catch (Exception e) {
            }
        } catch (Exception e) {
        }
    }
```

```
    public static int Main(String[] args) {
        StartClient();
        return 0;
    }
}
```