Universidade de Aveiro Departamento de Engenharia Mecânica
2017

Vítor Augusto Nogueira da Silva

Integração de Manipulador FANUC na Plataforma Robuter para Manipulação Móvel

Universidade de Aveiro Departamento de Engenharia Mecânica
2017

Vítor Augusto Nogueira da Silva

Integração de Manipulador FANUC na Plataforma Robuter para Manipulação Móvel

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestrado em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro e de Rui Cancela, R&D em robótica industrial na instituição Motofil Robotics, S.A.

O júri / The jury

Presidente / President

Prof. Doutor Jorge Augusto Fernandes Ferreira
Professor Auxiliar da Universidade de Aveiro

Vogais / Committee

Prof. Doutor Germano Manuel Correia dos Santos Veiga
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto (arguente)

Prof. Doutor Vítor Manuel Ferreira dos Santos
Professor Associado da Universidade de Aveiro (orientador)

Agradecimentos / Acknowledgements

Aos meus pais, irmão e avó por me oferecerem sempre as melhores condições para alcançar os meus objetivos e por, mesmo nos momentos difíceis, nunca deixaram de acreditar em mim.
Ao professor Vítor Santos pelo regular acompanhamento, orientação e pela motivação ao longo de todo o semestre.
Gostaria ainda de agradecer à empresa Motofil pela cedência do robô industrial.
Por último, mas não menos importante, agradeço a todos os meus amigos que, de uma forma ou de outra, contribuíram para a minha formação pessoal e académica.

Palavras-chave

ROS-Industrial; ROS-FANUC; FANUC LR Mate 200iD; Plataforma Robuter II; Manipulação Móvel; Sistema Robótico Integrado; Indústria 4.0

Resumo

A manipulação móvel está a tornar-se numa frente com cada vez mais aplicações em ambiente industrial e já se pensa na sua aplicação futura em ambiente doméstico. De facto, é evidente a infinidade de aplicações que pode ter um robô manipulador montado em cima de uma plataforma móvel. Com este avanço tecnológico surgem problemas e desafios, como é o caso da precisão do *gripper*, localização do sistema, navegação/planeamento de trajetória, controlo e perceção, que estariam relativamente bem controlados num manipulador fixo.

A presente dissertação insere-se no projeto Robonuc em que é pretendido o desenvolvimento de um sistema robótico integrado com um custo consideravelmente inferior aos já existentes no mercado.

A instalação mecânica e elétrica do manipulador FANUC LR Mate 200iD na plataforma móvel Robuter II, ambos os recursos disponíveis no Laboratório de Automação e Robótica (LAR), foi a primeira tarefa a ser realizada. Seguiu-se o desenvolvimento de uma arquitetura de *software* modular, em ambiente *Robot Operating System* (ROS), que possibilitasse a comunicação entre todos os dispositivos do sistema, a *interface* com o utilizador e o controlo eficaz do manipulador. Ao nível do *software*, foi também resolvido o problema do controlo de I/O's do manipulador - uma das limitações do ROS-FANUC - e ainda contornada a limitação cinemática da biblioteca MoveIt!.

Para introduzir segurança no sistema, foram instalados sensores laser de varrimento para deteção da aproximação de pessoas e/ou objetos. Uma integração dos dados dos sensores na arquitetura de *software*, com vários níveis de alarme, permitiu a colaboração de humanos com o sistema robótico enquanto se realizam operações de manipulação.

A fase de testes revelou-se essencial para a correção de alguns aspetos, como o controlo remoto das aplicações ou a parametrização da unidade de segurança. A realização de aplicações ilustrativas do sistema global demostrou que as metodologias adotadas no decorrer dos trabalhos foram as adequadas.

Keywords ROS-Industrial; ROS-FANUC; FANUC LR Mate 200iD; Robuter Platform; Mobile Manipulation; Integrated Robotic System; Industry 4.0

Abstract

Mobile manipulation is becoming a front with more and more applications in the industrial environment and its future application in the domestic environment is already in progress. As a matter of fact, the plethora of applications that a robotic manipulator can have while mounted on the top of a mobile platform are evident.

With this technological breakthrough, many problems and challenges arise, such as gripper precision, system localization, trajectory, navigation/planning, perception and control which would be relatively well controlled in a fixed manipulator.

The present dissertation is a part of the Robonuc project in which the development of an integrated robotic system with a considerably lower cost relative to the market is pretended.

Mechanical and electrical implementation of the FANUC LR Mate 200iD manipulator on the Robuter II mobile platform, both of which available in Laboratório de Automação e Robótica (LAR) was the first task to be accomplished. Afterwards, a modular software architecture was developed on the Robotic Operating System (ROS), which allows communication between all system devices, interface with the end user and efficient manipulator control. At a software level, the problem of the manipulator's I/O control - which is one of the ROS-FANUC limitations - was solved. The kinematic limitation of MoveIt! library was also solved.

To introduce system safety, laser range finder sensors were installed in order to detect the proximity of people and objects. The integration of sensor data with different alarm levels in the software architecture made it possible to collaborate with the robotic system during manipulation tasks.

The testing phase was essential for the correction of some aspects, such as the remote control of applications or the setup of safety parameters. Illustrative applications were made to demonstrate that the adopted methodologies during this work were adequate.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Acrónimos

2D Bidimensional.

3D Tridimensional.

ADU *Application Data Unit.*

API *Application Programming Interface.*

CAD *Computer Aided Design.*

DEM Departamento de Engenharia Mecânica.

DFA *Design for Assembly.*

DFM *Design for Manufacturing.*

FTP *File Transfer Protocol.*

HTTP *Hypertext Transfer Protocol.*

I/O *Input/Output.*

IP *Internet Protocol.*

LAR Laboratório de Automação e Robótica.

LRF *Laser Range Finder.*

OSI *Open System Interconnection.*

PWM *Pulse Width Modulation.*

ROS *Robot Operating System.*

ROS-I *ROS Industrial.*

SSH *Secure Shell.*

STL *Stereolithography.*

TCP *Transmition Control Protocol.*

TF *Transform Frame.*

URDF *Unified Robot Description Format.*

Capítulo 1

Introdução

1.1 Manipulação móvel

A manipulação móvel [1] é uma área de investigação e desenvolvimento da robótica que se preocupa com o desenvolvimento de sistemas robóticos integrados, capazes de atender a uma ampla gama de aplicações em ambientes reais. Os sistemas de manipulação móvel devem conseguir realizar um determinado conjunto de tarefas, adquirir novas competências e aplicá-las em novas situações. Devem portanto ser capazes de se adaptar continuamente ao meio onde circulam e às suas restrições mantendo ou, se possível, melhorando a sua performance.

A capacidade de se deslocar, a generalidade necessária na execução de tarefas e o uso de vários sensores e atuadores tornam o problema da manipulação móvel muito abrangente, o que se torna numa dificuldade ao seu desenvolvimento tendo que haver a cooperação de várias, senão todas as áreas da robótica tradicional, como é o caso da perceção, planeamento de trajetória, manipulação, controlo, inteligência artificial, etc.

O termo manipulação móvel surgiu pela primeira vez nos anos 90, quando os laboratórios de pesquisa começaram a montar robôs manipuladores em plataformas móveis. Naquela época, o termo pretendia captar apenas isso: uma plataforma experimental que combina capacidades de locomoção e manipulação.

Rapidamente se tornou evidente que combinar mobilidade com manipulação era um grande salto na robótica. A mobilidade permite que os manipuladores circulem por um espaço não controlado, enfrentando complexidades que têm que ser resolvidas imediatamente. A conjugação da manipulação com a locomoção levou a que a pesquisa que outrora tinha sido bem sucedida para ambientes controlados, se tornasse frágil ou até mesmo inadequada. O trabalho já feito estava fortemente dependente do conhecimento prévio do ambiente para ser bem sucedido; requeria um ambiente experimental cuidadosamente especificado, o que contrasta com a imprevisibilidade e a não estacionabilidade dos ambientes quotidianos.

Assim, no contexto da manipulação móvel, ambientes controlados - como é o caso de um laboratório com configurações experimentais específicas - são frequentemente referidos como ambientes estruturados. Em contraste, ambientes não estruturados são ambientes que não foram modificados especificamente para facilitar a execução de uma tarefa por parte de um robô. Naturalmente que estes dois tipos de ambientes são opostos, no entanto, é possível notar que mesmo os ambientes não estruturados contêm uma quantidade significativa de estruturação que pode ser explorada pelo sistema robótico [2].

Em ambientes não estruturados, qualquer tarefa específica - como recuperar um livro da biblioteca - pode exigir que o robô resolva desafios adicionais, como abrir portas, operar elevadores, mover uma cadeira para se aproximar da prateleira ou pedir a localização do mesmo na biblioteca. Hoje, nenhum sistema robótico possui tais capacidades. Não obstante esse facto, começam a surgir alguns modelos no mercado que estão em constante desenvolvimento. A figura 1.1 ilustra alguns exemplos de sistemas robóticos integrados atualmente no mercado.

Figura 1.1: Exemplo de sistemas robóticos integrados da marca Kuka [3]

Investigadores na área da manipulação móvel verificaram que o progresso alcançado nas áreas tradicionais de pesquisa da robótica não pode simplesmente ser combinado para produzir o nível de competência que eles procuram. Dessa forma, a combinação de manipulação e mobilidade leva à fundação da manipulação móvel como uma área de pesquisa. A pesquisa nesta área busca desenvolver sistemas robóticos capazes de executar tarefas de forma autónoma, em ambientes não estruturados e dinâmicos, onde a cooperação com humanos pode ser um requisito. Dada a complexidade do problema este processo será gradual. Desta forma, a pesquisa associada à manipulação móvel aumenta progressivamente a autonomia, robustez e o número de aplicações de sistemas robóticos, ao mesmo tempo que reduz gradualmente a dependência de informações prévias sobre o ambiente e a tarefa.

1.2 Enquadramento e motivação

A manipulação móvel é uma área de atividade em crescimento exponencial no mundo industrial. O surgimento da indústria 4.0 impulsionou ainda mais o seu desenvolvimento tendo surgido, nos últimos tempos, um vasto conjunto de sistemas robóticos integrados. No futuro, também já se consegue prever a expansão da manipulação móvel para ambientes domésticos.

Este crescimento advém principalmente da infinidade de aplicações que pode ter um

robô manipulador montado sobre uma plataforma móvel. A liberdade de movimento do manipulador por um espaço incondicionado, sem estar restrito a uma única posição ou a um carril - como se faz tradicionalmente - são condições que realçam as suas valências.

O poder, sofisticação e robustez da manipulação abriu asas para que se pensasse em movê-la por um espaço de trabalho mais alargado, pelos seus próprios meios, desempenhando variadas tarefas em diversos postos de trabalho. O transporte de objetos entre locais distintos e outras funções onde seja necessária simultaneamente manipulação e locomoção, são outras das aplicações que se podem dar a um sistema robótico integrado.

A existência de um manipulador FANUC e de uma plataforma móvel no LAR possibilitou o início de um novo projeto. Este projeto propõe o desenvolvimento de um sistema robótico integrado que será usado para o estudo de problemas e desafios - como é o caso da precisão, localização, navegação/planeamento de trajetória, perceção, etc. Para isso, numa primeira fase, é necessário fazer uma instalação mecânica e elétrica de um manipulador numa plataforma móvel, desenvolver uma arquitetura de *software* e de comunicações entre os componentes e efetuar testes no sentido de perceber alguns dos problemas da manipulação móvel.

1.3 Objetivos

Com a presente dissertação pretende-se atingir os principais objetivos:

Instalação mecânica e elétrica do manipulador

O primeiro objetivo engloba a instalação mecânica e elétrica do manipulador FANUC LR Mate 200iD na plataforma Robuter II, ambos disponíveis no LAR. Para tal, é necessário fazer algumas alterações na plataforma, onde se destaca a conceção de um novo patamar para alojamento do controlador do robô e ainda de toda a cablagem associada. Questões como a localização do manipulador, a altura e montabilidade do sistema e a resistência mecânica da estrutura deverão ser tidas em conta nesta fase. No que diz respeito à instalação elétrica, deverá ser assegurada a alimentação do manipulador através de um inversor de corrente, que será alimentado pelas baterias da plataforma, ou de um cabo que terá que acompanhar sempre a plataforma.

Instalação de infraestruturas de *software*

Esta é uma fase crítica do trabalho, pois o *software* que vai estar em execução no equipamento tem que ter a robustez, a fiabilidade e a facilidade de acesso suficiente para que o sistema robótico seja viável. Terá que ser assegurada uma comunicação em tempo real entre a plataforma, o manipulador, e uma unidade de controlo, que fará também a interface com o utilizador. À partida, há duas opções que serão usadas, nomeadamente o ROS e o protocolo ModBus/TCP. Prevê-se a utilização do ROS para controlo e monitorização do manipulador, assim como para assegurar a sua comunicação com a unidade de controlo e plataforma. Paralelamente, o uso do protocolo ModBus será utilizado para controlo de I/O's do robô. Nesta fase de desenvolvimento de *software* as condições de segurança têm que ser levadas em conta pois trata-se de um aspeto fundamental na conceção de qualquer máquina.

Desenvolvimento de uma aplicações ilustrativas do sistema global

Depois do sistema robótico desenvolvido, segue-se a fase de testes, onde podem ser detetados problemas que implicarão alterações do ponto de vista mecânico ou do *software*. Deverá ser testada a comunicação entre as partes integrantes do sistema, a viabilidade mecânica da estrutura, interface com o utilizador, etc. Para tal, deverão desenvolver-se aplicações demonstrativas do sistema global. Dada a limitação no número de sensores a utilizar, a variedade de aplicações será reduzida. No entanto, deverá desenvolver-se uma aplicação que explore ao máximo as funcionalidades implementadas no sistema.

1.4 Trabalhos relacionados

Encontram-se na literatura um número considerável de trabalhos na área da manipulação móvel. Sendo uma área em crescimento, torna-se o alvo de estudo de inúmeros trabalhos de pesquisa e, por vezes, chega-se mesmo a construir o protótipo de um sistema robótico integrado para testes e estudos mais práticos. Encontram-se ainda vários trabalhos sobre a remodelação da plataforma Robuter. Como já é uma plataforma antiga, o seu *hardware* e *software* ficaram desatualizados com a evolução tecnológica registada nos últimos anos. Entre todos os trabalhos revistos, destacam-se dois que se descrevem de seguida:

1.4.1 Integração de ROS-Industrial num robô FANUC para flexibilixar atividades de cooperação

Na dissertação de mestrado de Tiago Simões [4], foi integrado o ROS ao manipulador robótico FANUC de forma a oferecer uma nova API ao utilizador para o seu controlo. A nova API revelou-se uma ferramenta muito útil no processo de simplificação da programação do robô, possibilitando a utilização de uma linguagem mais próxima da especificação da lógica que traduz o comportamento do sistema, abstraindo o utilizador de detalhes.

Um outro progresso passa pela visualização do planeamento de trajetória numa interface gráfica antes da execução do movimento físico do robô. Com esta funcionalidade, pode antecipar-se o movimento do braço robótico, num ambiente virtual, a fim de aferir se o movimento está em conformidade com o pretendido e, em caso negativo, podem evitar-se trajetórias indesejadas.

A deteção de colisões contra o próprio robô ou contra o meio envolvente é outra das evoluções conseguidas com este trabalho. Através da introdução dos modelos *Computer Aided Design* (CAD) dos elos do manipulador e/ou objetos presentes no meio envolvente, é possível a construção de uma cena virtual, onde para cada movimento é verificada se existe a sobreposição de modelos que revela a colisão entre componentes.

Tendo sido o primeiro projeto no LAR que integra a API ROS-Industrial num manipulador industrial, algumas limitações e/ou fragilidades podem ser detetadas. O controlo de I/O's do manipulador - indispensável para o acionamento do *gripper* do robô ou para a sua troca - é a principal limitação deste trabalho. Verifica-se que nem o *ROS Industrial* (ROS-I) nem o ROS FANUC, ambos fundamentais no controlo de movimento do manipulador, estão preparados para o controlo de I/O's do robô. Como na sua aplicação de demostração era necessário o acionamento do *gripper* (acionado através

de ar comprimido), isso era feito de forma manual. O desenvolvimento de uma rede de campo, baseada no protocolo ModBus, foi a proposta sugerida pelo autor para o controlo de variáveis lógicas digitais.

Uma outra limitação deste trabalho envolve o movimento do manipulador. Não sendo possível a escolha da redundância com que o manipulador chega à posição pretendida, por vezes essa posição não vai ao encontro da expetativa do utilizador. O facto de ser usada uma cinemática inversa aproximada - onde apenas é utilizada a matriz jacobiana inversa para o cálculo dos valores de juntas correspondentes aos valores cartesianos - também tem as suas consequências no planeamento de trajetória entre dois pontos. Por vezes, a solução de trajetória que leva a ponta do robô de um ponto inicial para outro final é ineficiente ao passo que envolve movimentos complexos, ao invés de um simples e curto movimento.

1.4.2 *Mobile manipulation: a case study*

Um outro trabalho, alvo de uma análise mais profunda, foi o dos autores A. Hentout, B. Bouzouia, I. Aklt e R. Toumi cujo título é *Mobile Manipulation: A Case Study* [5]. Trata-se de um trabalho onde se desenvolveu um projeto, do ponto de vista mecânico, muito similar ao presente projeto dada a utilização da plataforma Robuter e de um manipulador com 6 graus de liberdade. A figura 1.2 ilustra o trabalho desenvolvido pelos referidos autores no que respeita ao *hardware*:

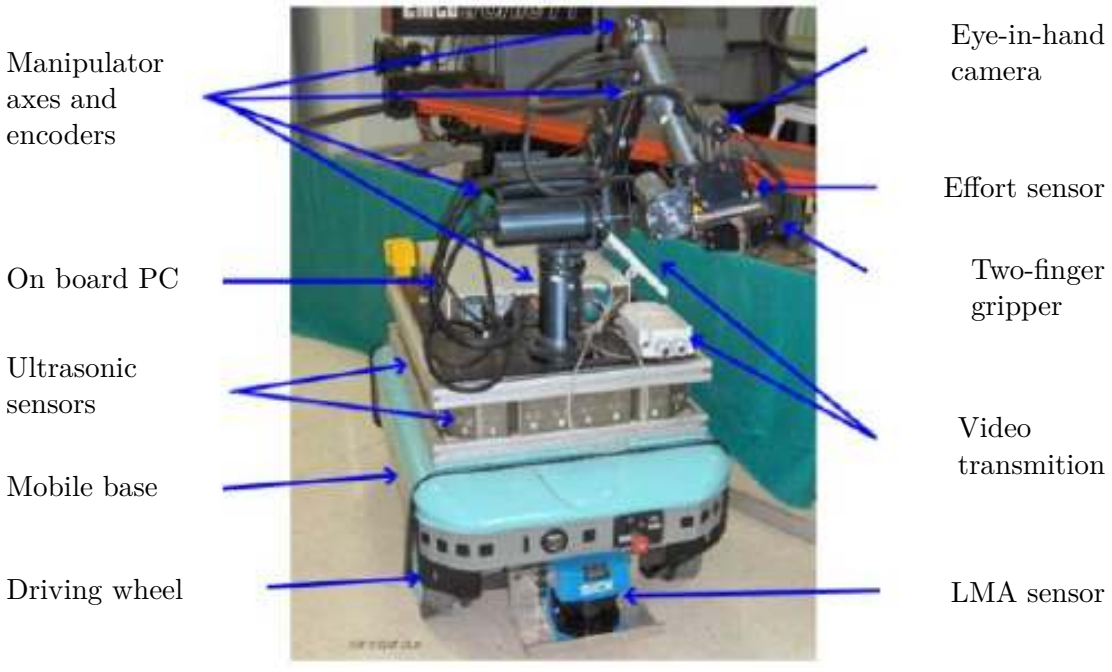


Figura 1.2: Sistema robótico integrado RobuTER/ULM

Neste trabalho, além da conceção de um sistema robótico integrado, foram estudados assuntos que são relevantes no corrente projeto, nomeadamente um estudo cinemático do conjunto plataforma mais manipulador com o objetivo do conhecimento da posição da ponta do robô em relação a um referencial global. As variáveis de entrada são as

coordenadas da plataforma em relação ao referencial global (x_B, y_B, θ_B), obtidas por odometria, e as coordenadas do manipulador em relação à plataforma $(\psi_E, \theta_E, \varphi_E)$. À parte a introdução de mais uma variável (z_B) para definição da posição da plataforma, o estudo cinemático utilizado neste projeto será muito semelhante.

Quanto à comunicação entre os diversos componentes do sistema (manipulador, sensores, unidade de controlo, ...), os autores utilizaram o protocolo de comunicação Ethernet TCP/IP que troca informação através de variáveis partilhadas protegidas por "semáforos". No caso do sistema Robonuc, que se irá projetar, pretende utilizar-se o ROS como alternativa a esta forma de comunicação.

No que diz respeito à unidade de controlo, os autores usam dois computadores: um que opera a bordo da plataforma, responsável pela ramificação do controlo aos diferentes dispositivos, e um outro, remoto, que opera fora do sistema em comunicação com o primeiro e que está a fazer a interface com o utilizador. A ideia dos dois computadores revela-se interessante pois, desta forma, permite que o operador tenha controlo sob a plataforma, em tempo real, conseguindo cancelar, modificar e monitorizar o que está a acontecer em todo o sistema robótico.

Desde já se consegue aferir que um dos principais problemas do sistema experimental é a sua localização no espaço. Observa-se, neste trabalho, que a localização recorrendo à odometria - baseada nos *encoders* incrementais das rodas - não é suficientemente precisa, pois existem erros acumulados difíceis de controlar. Deverá ser usado um método de localização no espaço que consiga dar a posição da plataforma no espaço com uma precisão significativa. A melhor forma encontrada pelos autores foi a instalação de uma câmara na ponta do robô que, com recurso à perceção, permitiu a obtenção de valores aceitáveis de precisão para o *gripper* do robô.

1.5 Estrutura da dissertação

A presente dissertação é formada por oito capítulos, organizados da seguinte forma:

1. **Introdução** - É feito um breve enquadramento teórico, enumerados os objetivos da dissertação e uma análise de trabalhos relacionados;

2. **Infraestrutura existente de *hardware* e *software*** - Descrição do principal *hardware* e *software* a utilizar para a concepção e desenvolvimento do protótipo;

3. **Sistema robótico integrado** - Apresentação da proposta de intervenção mecânica e simulações associadas, assim como configurações em modelos CAD da API;

4. **Arquitetura de *software*** - Descrição detalhada da arquitetura de *software* desenvolvida para o funcionamento do sistema;

5. **Controlo de I/O's** - Exposição das metodologias abordadas para o controlo de I/O's do manipulador;

6. **Unidade de Proteção e Segurança** - Implementação de sensores para segurança e sua integração na arquitetura de *software* principal;

7. **Aplicações de demostração e estudo** - Demostração e análise dos primeiros testes realizados;

8. **Conclusão** - Apresentação das conclusões do trabalho e propostas de trabalho futuro.

Capítulo 2

Infraestrutura Existente de *Hardware* e *Software*

Numa primeira fase, antes de se proceder ao desenvolvimento de qualquer estrutura ou *software*, é fundamental o conhecimento e familiarização com o *hardware* que se vai utilizar; desta forma adquire-se algumas ferramentas e conhecimento que será uma mais-valia no decorrer do projeto. O *hardware*, que será um do objetos de estudo desta dissertação, está disponível no LAR do Departamento de Engenharia Mecânica. Neste capítulo é também descrita a plataforma de desenvolvimento de *software* para robôs (ROS) assim como de algumas das suas extensões como é o caso do ROS Industrial, biblioteca MoveIt! ou o ROS FANUC.

2.1 FANUC LR Mate 200iD

O componente mais importante que será utilizado no projeto de dissertação é o manipulador FANUC LR Mate 200iD [6]. Trata-se de um robô industrial versátil, ideal para operações de montagem, empacotamento, *pick and place*, testes e também para atividades educacionais e escolares. A sua versatilidade está ainda justificada pela sua flexibilidade quanto à posição de montagem que adiciona um vasto conjunto de aplicações ao robô.

No que respeita às características técnicas, presentes na tabela 2.1, o manipulador apresenta 6 juntas distribuídas de forma a conferir ao seu *end-effector* 6 graus de liberdade. Os 6 eixos utilizam servomotores de elevado desempenho que conferem ao braço industrial um conjunto de propriedades muito interessantes, nomeadamente uma rigidez elevada, que se reflete numa repetibilidade de ± 0.02mm e ainda um movimento suave e sem vibração mesmo a velocidades elevadas. De realçar, também, o seu alcance de 717 mm e a sua capacidade de carga de no máximo 7kg, pelo que, por este motivo, se assemelha com a morfologia de um braço humano.

À região dentro da qual o *end-effector* consegue operar sem limitações nos seus graus de liberdade, dá-se o nome de espaço de trabalho. Essa região é limitada pelas dimensões físicas do manipulador e também pelos limites das juntas. Na figura 2.2 está ilustrado o espaço de trabalho do robô.

Figura 2.1: Manipulador FANUC LR Mate 200iD

Model		LR Mate 200*i*D
Controlled axes		6 axes (J1, J2, J3, J4, J5, J6)
Reach		717mm
Installation (Note 1)		Floor, Upside-down, Angle mount
Motion range (Maximum speed)	J1 axis	340°/ 360° (option) (450°/s) 5.93 rad/6.28 rad (option) (7.85 rad/s)
	J2 axis	245° (380°/s) 4.28 rad (6.63rad/s)
	J3 axis	420° (520°/s) 7.33 rad (9.08rad/s)
	J4 axis	380° (550°/s) 6.63 rad (9.60 rad/s)
	J5 axis	250° (545°/s) 4.36 rad (9.51 rad/s)
	J6 axis	720° (1000°/s) 12.57 rad (17.45 rad/s)
Max. load capacity at wrist		7kg
Allowable load moment at wrist	J4 axis	16.6 N·m
	J5 axis	16.6 N·m
	J6 axis	9.4 N·m
Allowable load inertia at wrist	J4 axis	0.47 kg·m^2
	J5 axis	0.47 kg·m^2
	J6 axis	0.15 kg·m^2
Repeatability		± 0.02 mm
Mass (Note 2)		25 kg
Installation environment		Ambient temperature : 0~45℃ Ambient humidity : Normally 75%RH or less (No dew nor frost allowed), Short term 95%RH or less (within one month) Vibration : 0.5G or less

Tabela 2.1: Especificações do FANUC LR Mate 200iD [7]

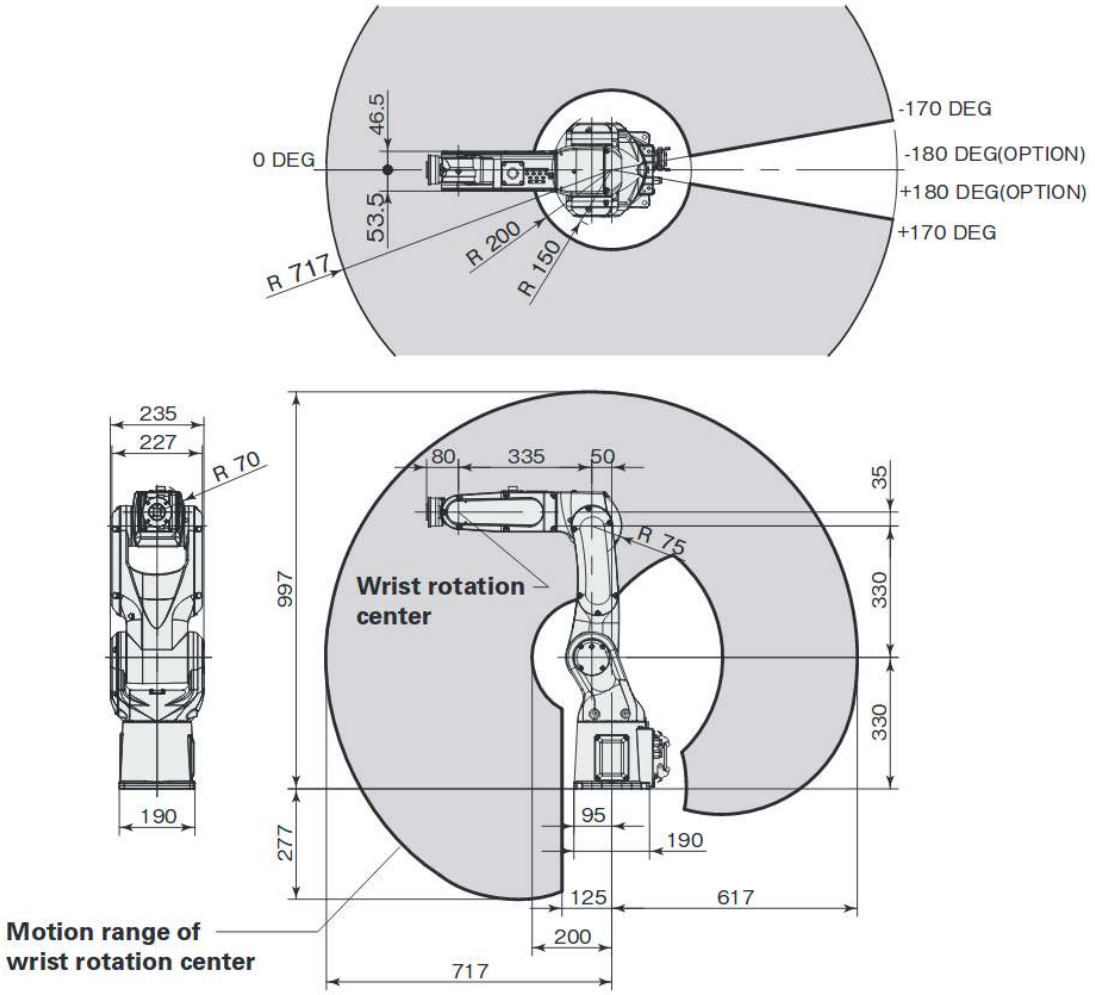


Figura 2.2: Dimensões do FANUC LR Mate 200iD e seu espaço de trabalho [7]

2.2 Controlador R-30iB Open Air

Todo o manipulador tem um controlador associado, um elemento fundamental pois está responsável pelo comando de todo o *hardware* associado ao braço robótico, pela comunicação com outros dispositivos e ainda pela interface com o utilizador. Caracterizado pelo seu elevado nível de desempenho e velocidade de processamento relativamente aos outros modelos da marca, o controlador utilizado é o R-30iB Open Air Cabinet [8]. É bastante usado para controlo de robôs da gama Mate dado o seu tamanho reduzido (370x350x250mm), silêncio em funcionamento e possibilidade de empilhamento.

Figura 2.3: Controlador R-30iB Open Air Cabinet

Várias características relativas ao referido controlador foram tidas em conta no decorrer do projeto, a destacar:

1. Interface com o utilizador

   De forma a possibilitar ao utilizador a interação com o controlador, desempenhando tarefas como a monitorização de I/O's, verificação de erros e alarmes, programação, edição e execução de programas, entre outras, o controlador dispõe de uma consola ou *teach pendant* ilustrado na figura 2.4, um componente do robô essencial na ligação Homem - Máquina.

2. Protocolos de comunicação disponíveis com o controlador

   (a) *Hypertext Transfer Protocol* (HTTP) - Utilizado como meio de ligação e troca de mensagens entre o PC e o controlador. É através deste protocolo que se enviam as instruções que são interpretadas pelo controlador para desempenhar funções específicas. Para que este processo de concretize é necessário que um servidor esteja a correr no controlador e à escuta das mensagens para posteriormente as processar. Na secção 2.4.3 será explicitado mais pormenorizadamente todo o funcionamento deste processo.

   (b) *File Transfer Protocol* (FTP) - Apenas utilizado para a transferência de ficheiros entre o PC e o controlador; esses ficheiros são previamente elaborados e compilados no *software* RoboGuide (Simulador FANUC). Na secção 5.1.2 voltará a ser mencionado este protocolo e também o referido *software*.

Figura 2.4: Teach Pendant FANUC

Ambos os protocolos usam como meio físico de ligação a porta Ethernet. O controlador dispõe ainda de uma porta USB que pode ser usada para transferência de dados, não tendo sido utilizada no decorrer dos trabalhos.

3. Interface I/O's

   O controlador do robô possui na parte de trás uma ficha com algumas opções de monitorização e controlo do robô de onde se destaca a carta de entradas e saídas digitais. Dado que os objetivos da dissertação passam pelo controlo de I/O's do robô (secção 5.6), o conhecimento da carta de entradas revelou-se fundamental.

2.3 Plataforma Robuter II

A plataforma ilustrada na figura 2.5 será o meio de locomoção do sistema, pelo que é também um componente central. A referida plataforma, com a sua estrutura original, onde estavam incluídos todos os seus 24 sensores de ultrassons e toda a eletrónica constituinte, tinha uma massa aproximada de 150 quilogramas, maioritariamente devido às 4 baterias de chumbo (4×12V) que asseguram a sua autonomia, no entanto, após as alterações que têm vindo a ser feitas no sentido de modernizar tanto o *hardware* como o *software*, o seu peso ficou cerca de 5% mais reduzido [9].

A sua capacidade de carga é de 120 quilogramas (com a redução de peso este valor aumenta), a sua velocidade pode variar até um máximo de 1.25 metros por segundo. A locomoção da plataforma é conseguida através de dois motores DC, alimentados a 48V e que têm uma potência de 300 Watt cada. Quanto à transmissão, sendo que cada rodas dispõe de um controlo independente, podemos classificá-la como diferencial.

2.4 Infraestrutura de *software*

Não obstante a existência de outros sistemas distribuídos para controlo de robôs - como o Mobile Robot Programming Toolkit [10], Microsoft Robotics Developer Studio [11] ou CARMEN [12] - o ROS (Robot Operating System) será o escolhido para todo o controlo e monitorização do sistema robótico integrado a desenvolver no decorrer do trabalho de dissertação.

Figura 2.5: Plataforma Robuter

A razão da sua utilização reside no facto de ser uma alternativa já explorada em diversos trabalhos realizados no LAR e ainda ao facto do manipulador FANUC - a ser usado como dispositivo de manipulação - também já estar preparado para comunicar com a plataforma ROS. Desta forma, e do que do ponto de vista da concepção de *software*, é considerado um avanço significativo.

2.4.1 ROS

Criada em 2007, e fortemente impulsionada pela *Willow Garage*, a plataforma de desenvolvimento ROS integra uma coleção de *frameworks* de *software*, essencialmente bibliotecas e ferramentas, dedicadas especialmente ao desenvolvimento de aplicações para robôs [13].

Dada a sua arquitetura modular, é possível a reutilização de módulos e/ou bibliotecas de projetos anteriores no desenvolvimento de novos projetos. Ao invés das aplicações centralizadas para robôs - em que se alguma *thread* do código principal deixar de funcionar toda a aplicação pode parar - o ROS adota uma filosofia de descentralização, fornecendo métodos robustos para, se um processo deixar de funcionar, os restantes continuarem em execução de forma a que o sistema continue em funcionamento.

A linguagem de programação para desenvolvimento de uma estrutura ROS também foi levada em conta por parte dos autores. Como tal, o ROS suporta quatro tipos de linguagem, nomeadamente C++, Phyton, Octave e LISP, em que o programador pode optar pela que lhe seja mais conveniente [14].

Esta e outras vantagens, como a disponibilização de drivers e pacotes de interface para sensores e atuadores de diversas marcas, a comunidade ativa de desenvolvimento de *software* e/ou esclarecimento de dúvidas dos utilizadores ou mesmo o facto de ser *open source*, levam a que o ROS seja geralmente a preferência do utilizador quando pretendem desenvolver *software* para robôs [15].

Metapackage

Similarmente a um sistema operativo, o ROS está organizado de uma forma muito específica. No topo da estrutura, encontram-se as *metapackages* que não são mais que um conjunto de *packages* com um determinado objetivo. Exemplos de *metapackage* são o

ROS navigation, utilizado para navegação com base em odometria, ou o ROS Industrial, usado maioritariamente para o controlo de manipuladores industriais [13].

Package

A *package* é a unidade mais básica de um *software* ROS e nela estão incluídos nodos, ficheiros de configuração, etc. Todas as *packages* estão organizadas numa estrutura típica de ficheiros, separados em pastas, consoante a sua função.

Nodo

Genericamente, nodos podem ser definidos como processos ou executáveis encarregues de computação. Numa arquitetura ROS geralmente são utilizados vários nodos com funções simples em vez nodos que suportam todas as funcionalidades do sistema. Um nodo pode desempenhar um vasto leque de funções, tais como adquirir dados de sensores (câmaras, lasers, receptores GPS), fazer processamento de dados, comunicar com dispositivos remoto, etc. Cada nodo pode comunicar com outros através de tópicos, serviços ou parâmetros.

Mensagens

As mensagens são a forma mais usada de comunicação entre os vários nodos de uma arquitetura ROS. É usada uma linguagem de definição de mensagens simplificada para descrever o tipo de valores que cada nodo publica. Assim, consegue-se a geração de uma mensagem que é reconhecida por todos os nodos, independentemente da linguagem em que estejam programados. Já se encontram definidos vários tipos de mensagens *standard* (std_bool, std_int8, std_string,...), no entanto, o utilizador poderá definir qualquer outro, desde que obedecendo aos requisitos da criação de mensagens ROS.

Tópico

Cada mensagem ROS é transportada através de um meio a que se deu o nome de tópico. Quando um nodo envia uma mensagem através de um tópico, diz-se que o nodo é publicador desse tópico. Imediatamente a seguir e através desse mesmo tópico, a mensagem enviada será recebida noutro nodo. Assim, diz-se que o nodo receptor é um subscritor do tópico fechando-se, desta forma, o ciclo de comunicação. Para que este tipo de comunicação unidirecional se possa concretizar, o tópico tem que ser associado ao mesmo tipo de mensagem por parte de todos os nodos que participam na comunicação, sendo que pode haver vários nodos a publicar uma mensagem para um tópico ou vários a subscrever. Poderá ainda ocorrer a situação de haver tópicos que não são publicados ou subscritos por nenhum nodo.

Serviços

Em algumas aplicações o modelo de publicação/subscrição de tópicos não chega para dar resposta às necessidades, nomeadamente se for exigida uma comunicação bi-direcional entre dois nodos. Surge então o conceito de uma comunicação servidor/cliente [16], onde é possível uma interação de pedido/resposta por parte de diferentes nodos, uma ferramenta muito útil em sistemas distribuídos [14]. No caso dos serviços, é necessário definir dois

tipos de mensagens: um para as mensagens que circulam num sentido (*request*), outro para as mensagens que circulam no sentido contrário (*response*). Ao contrário dos tópicos, os serviços só podem ser publicados por um nodo [17].

***Transform Frame* (TF)**

Os sistemas robóticos precisam com frequência de definir relações espaciais entre os seus componentes como, por exemplo, a posição relativa entre dois elos de um manipulador, a posição absoluta de uma plataforma móvel no espaço, etc [18]. Como tal, o ROS disponibiliza uma ferramenta (TF) que facilita a gestão de referenciais espaciais. Trata-se de uma transformada dinâmica que consegue definir completamente a posição de um objeto no espaço (translação e rotação) ou a posição relativa entre dois objetos. Essa transformação geométrica, dependendo da sua aplicação, poderá ser dada pelo utilizador ou pode ser gerada a partir de sensores [19].

Parâmetros

Enquanto se programa um robô, geralmente há a necessidade de definir alguns parâmetros, por exemplo o seu *Internet Protocol* (IP), velocidade máxima, limites físicos, etc. Por vezes, é ainda necessário que determinados valores possam ser acedidos por vários nodos ao mesmo tempo. Assim, o ROS providencia um servidor de parâmetros [20] partilhado que pode ser acedido por todos os nodos da arquitetura. Tanto o utilizador como qualquer nodo pode ler, escrever ou eliminar valores de parâmetros do servidor.

2.4.2 ROS Industrial

Com os avanços no desenvolvimento do ROS, em 2012 criou-se um novo projeto, numa colaboração entre a a Yaskawa Motoman Robotics, Willow Garage e Southwest Research Institute (SwRI), que procurava ser o meio de impulsão da chegada do ROS ao mundo industrial [21]. Ao projeto deu-se o nome de ROS Industrial [22]. É uma *metapackage* que surgiu como uma solução de controlo de manipuladores industriais usando as ferramentas ROS. Esta extensão do ROS contém portanto bibliotecas, *drivers*, modelos virtuais (URDF) e outras ferramentas destinadas a *hardware* industrial.

O desenvolvimento desta aplicação assenta em quatro fundamentos:

1. **Exploração das funcionalidades do ROS** - O ROS-Industrial foi concebido usando como base o *framework* ROS. Como tal, alargou-se a utilização das ferramentas ROS a robôs industriais, permitindo operações como a resolução de cinemáticas, manipulação de objetos usando perceção 2D e 3D, etc. São disponibilizadas ainda ferramentas como o RViz, Gazebo ou rqt_gui para visualização, simulação ou *debugging* de operações com robôs industriais.

2. **Inovação de aplicações** - A interface ROS disponibiliza ferramentas avançadas de percepção que, facilitando operações de manipulação de objetos complexos por parte dos robôs, expande o número de aplicações em que os robôs se podem inserir.

3. **Simplificação da programação de robôs** - Usando o ROS-I é possível uma programação *offline* de robôs onde os pontos de destino são o único *input*. O *software* irá encarregar-se de procurar a melhor trajetória levando em conta a heurística da não colisão do robô.

4. **Redução custos** - Simuladores robóticos são na sua maioria dispendiosos. Sendo o ROS-I um *software open source*, podendo ser usado para fins comerciais sem restrições, a poupança associada à sua utilização é significativa.

Pra se efetivar o controlo do robô FANUC LR Mate 200iD, o ROS-I depende de várias *metapackages* como o ROS MoveIt, ROS FANUC e das drivers a instalar no controlador. No entanto, é ao nível do ROS industrial que se encontram definidas algumas bibliotecas fundamentais na interação do ROS com os manipuladores industriais. A *metapackage* de maior destaque é a *Industrial Core*; é nela que se encontram definidas grande parte das mensagens específicas para aplicações industriais(Industrial msgs), o protocolo de comunicação entre o ROS e os controladores (Simple Message), bibliotecas de clientes para se conectarem com servidores alojados em controladores (Industrial robot client), bibliotecas para deteção de erros ou falhas de execução (Industrial utils), etc.

2.4.3 fanuc_driver

Da autoria de G.A. vd. Hoorn (TU Delft Robotics Institute), a *package fanuc_driver* [23] contém as *drivers* a instalar em qualquer controlador FANUC para possibilitar a sua comunicação com o ROS-I [24].

Esta *package* contém vários ficheiros destinados a serem compilados, recorrendo ao simulador FANUC Roboguide, de forma a gerar ficheiros executáveis que posteriormente serão enviados para um controlador FANUC. Tratam-se portanto das *drivers* do controlador em que se conseguem identificar quatro ficheiros principais, nomeadamente:

1. `ros_state.kl`

   Programado em linguagem karel, este ficheiro resultará no executável `ROS_STATE.SM`. Trata-se de um servidor ROS que estará à escuta numa porta pré definida, esperando que um cliente se conecte. Qualquer um que se conecte irá, segundo as normas do protocolo *simple message* e a uma taxa configurável, receber a posição atual das juntas do robô na forma de array contendo os valores correspondentes à posição das juntas em relação à sua *home position*.

2. ros_`relay.kl`

   Também programado em linguagem karel, dará origem ao executável `ROS_RELAY.SM`. É também um servidor, este responsável por receber valores de juntas, também obedecendo ao protocolo *simple message*, correspondentes a uma posição específica do manipulador. O servidor ficará ainda responsável por disponibilizar esses valores de juntas em registos de memória assim como outros parâmetros de configuração.

3. `ros_move.ls`

   Ao contrário dos ficheiros anteriores, este ficheiro está programado em linguagem TP. Quando compilado resultará no ficheiro `ROS_MOVESM.TP` cuja função é efetivar o movimento do manipulador de um ponto para outro obedecendo a parâmetros (velocidade, aproximação, ...) pré-definidos aquando a configuração dos dois servidores descritos. A comunicação dos servidores com este programa é assegurada através de registos de memória partilhados.

4. `ros.ls`

   Depois de compilado, dá origem ao ficheiro `ROS.TP`, o único ficheiro que será executado pelo utilizador. A sua função passa pela execução paralela dos três ficheiros acima.

Existem também outros ficheiros (maioritariamente bibliotecas) cuja sua função é a de suporte aos ficheiros principais.

A *package* ROS é ainda formada por dois nodos de teste de funcionamento de ambos os servidores ROS (*motion_streaming_interface* e *robot_state*) em execução no robô. Como os servidores podem ser testados diretamente com a interface gráfica de controlo do manipulador, estes nodos não foram explorados no decorrer da dissertação.

2.4.4 MoveIt!

Sendo uma parte integrante do ROS Industrial, o MoveIt! é um conjunto de *packages* e ferramentas destinadas à manipulação dinâmica de robôs [25]. A biblioteca MoveIt! pode ser utilizada para perceção 2D/3D, planeamento de trajetórias dinâmicas, cálculo de cinemáticas, verificação de colisões, controlo de *hardware*, etc.

Já disponível para interface com mais de 65 robôs, esta aplicação pode ser utilizada de duas formas específicas. A primeira utilizando a sua interface gráfica desenvolvida em ambiente RViz (figura 2.6) e a segunda recorrendo à programação. Em ambos os casos, e dependendo das ações que o utilizador pretende executar, haverá uma interação, levada a cabo através de tópicos e parâmetros, com a arquitetura de controlo ROS previamente em execução.

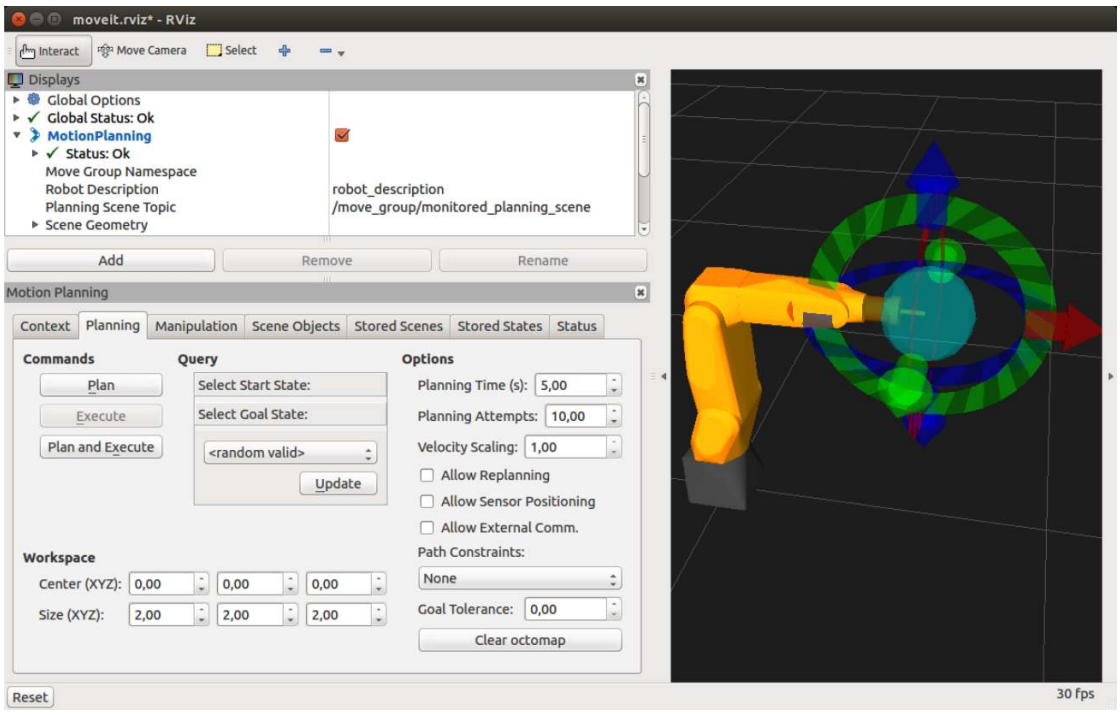


Figura 2.6: Interface RViz da biblioteca MoveIt!

Para poder oferecer as suas funcionalidades, o MoveIt! depende tanto das ferramentas do ROS como das do ROS-I, nomeadamente de tópicos e parâmetros para comunicação, da interface RViz para visualização ou mesmo das bibliotecas para comunicação com o controlador [26]. Quando se pretende utilizar a biblioteca MoveIt! para o controlo de um robô específico, é ainda necessário definir vários parâmetros, como por exemplo o tipo de juntas, seus limites físicos, velocidades máximas, etc. Para dar resposta a esta necessidade e objetivando o controlo de manipuladores FANUC, surge então o ROS FANUC.

2.4.5 ROS FANUC

Ao conjunto de *packages* para interação de manipuladores FANUC com a plataforma ROS dá-se o nome de ROS FANUC [27]. Apenas as três *packages* focadas no manipulador LR Mate 200iD foram utilizadas e desta forma ficam completos todos os requisitos ao nível do *software* para controlo do manipulador referido. Nas *packages* abordadas estão incluídos todos os ficheiros de configuração com informações relativas ao *hardware*, algoritmos de cálculo de cinemáticas, modelos STL dos elementos do robô para definição do modelo URDF, etc. A *launch file* que invoca as ferramentas do ROS Industrial, a biblioteca MoveIt! e o ROS FANUC permite lançar toda a base arquitetural para o controlo do manipulador (figura 2.7).

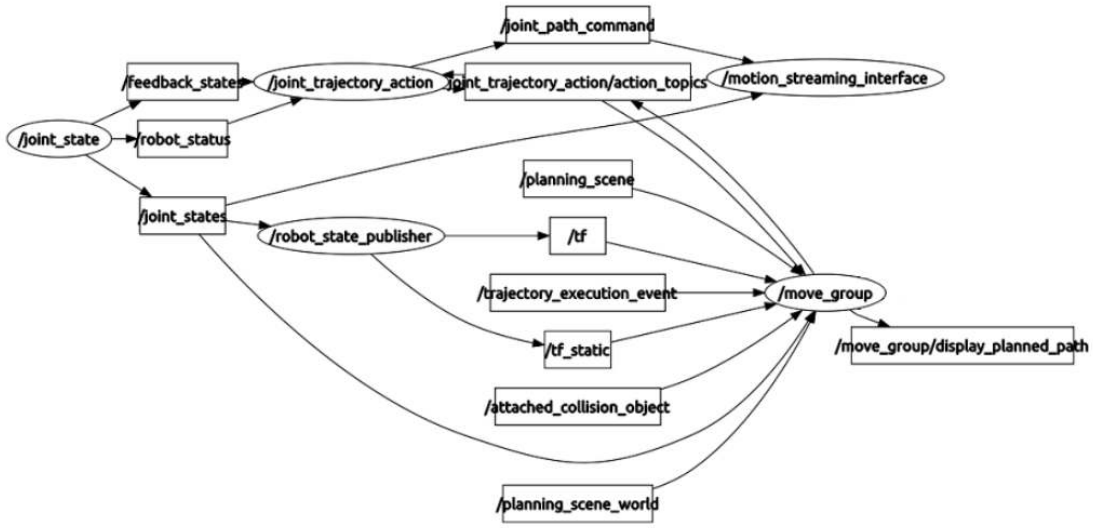


Figura 2.7: Arquitetura ROS de interface do utilizador com o robô industrial

Na figura pode identificar-se vários nodos, nomeadamente o `joint_state` - responsável por receber do controlador do manipulador o valor das suas juntas (dados pelos *encoders*) -, o `robot_state_publisher` - que calcula as transformações geométricas associadas aos valores de juntas -, o `joint_trajectory_action` - que planeia as trajetórias do manipulador, `motion_streaming_interface` - que transmite ao robô a posição para a qual é pretendido que o manipulador se mova -, e o nodo `move_group` - responsável por interligar toda a informação e permitir ao utilizador o controlo do manipulador o da cena [28].

Os tópicos, além de servirem de meio de comunicação entre os diversos nodos, também

podem dar informações aos nodos. Posições definidas por coordenadas cartesianas ou os objetos inseridos na cena de planeamento, introduzidos pelo utilizador, são exemplos dessas informações.

Assim, a forma mais simples de controlo do manipulador FANUC passa pela inicialização dos servidores no controlador do robô, e pela execução do comando `roslaunch fanuc_lrmate200id_moveit_config moveit_planning_execution.launch sim:=false robot_ip:=192.168.0.231` numa unidade de controlo onde estejam instalados o ROS, ROS-I e a biblioteca MoveIt!. Além da arquitetura da figura 2.7, onde estão incluídos os nodos e tópicos responsáveis por monitorizar o estado dos robô, comunicar com o controlador, definir o ambiente de operação, etc , o comando irá executar uma janela (figura 2.6) onde, se a conexão for estabelecida com sucesso, será possível a interface com o manipulador real.

Capítulo 3

Sistema Robótico Integrado

Como se pode observar na figura 2.5, a plataforma Robuter tem uma morfologia compacta e não está preparada para receber todo o *hardware* associado ao manipulador FANUC. Este capítulo descreve as intervenções mecânicas realizadas de forma a prepará-la para a acomodação não só do manipulador, mas também de todo o material necessário à sua autonomia. São também apresentadas as simulações de validação do projeto mecânico e, por último, é detalhado o processo de atualização do modelo URDF associado ao sistema robótico.

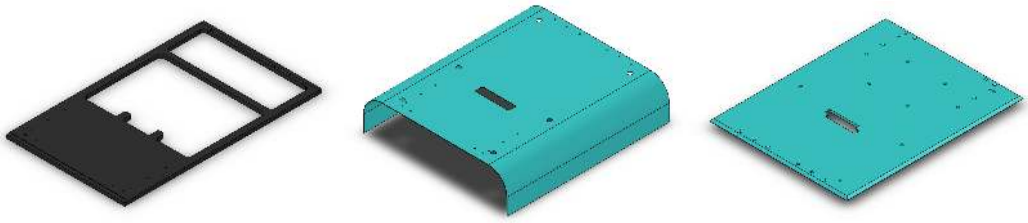
3.1 Projeto mecânico

O principal objetivo do projeto mecânico é a criação de mais um plano de trabalho na plataforma podendo assim dividir-se a sua estrutura em 3 patamares principais. O inferior estará dedicado ao controlo da navegação do sistema e à acomodação das baterias, o intermédio à arrumação do controlador R-30iB, toda a cablagem do robô e um eventual inversor de corrente DC-AC e/ou compressor de reduzidas dimensões e por último, o patamar superior deverá comportar o manipulador FANUC e ainda algum espaço livre para transporte de objetos (uma das finalidades da manipulação móvel).

Sendo este um projeto de reabilitação de uma estrutura já existente, é fundamental o seu estudo antes de se proceder a qualquer desenvolvimento. Para tal, foi desenvolvido um modelo CAD de todas as peças da plataforma envolvidas na intervenção mecânica. Na figura 3.1 encontram-se ilustradas essas peças, sendo que as primeiras continuarão a ser montadas no seu local original e apenas a última ficará elevada a um nível que permita a integração. No mesmo sentido, foram também analisados os pormenores mecânicos da estrutura e o seu método de montagem atual.

O primeiro passo do projeto mecânico foi a decisão da localização do manipulador na superfície superior e a do controlador no nível intermédio. Decidiu-se que ambos os componentes deveriam ficar por cima das rodas motoras da plataforma, a fim de minimizar o momento de inércia da plataforma. O manipulador fica ainda numa posição que otimiza a utilização do seu espaço de trabalho em operações de interação com o meio envolvente, ao passo que na sua retaguarda fica reservada uma grande área para o transporte de objetos.

Seguiu-se a definição da altura da estrutura, em que ficou decidido que se iria elevar a plataforma em 300 milímetros. Dado que o controlador tem uma altura de 250 milímetros,



(a) Placa estrutural inferior (b) Chapa para fins estéticos (c) Placa estrutural superior

Figura 3.1: Elementos de relevância para o projeto mecânico de um novo nível de trabalho

fica-se com 50 milímetros de margem para aparafusar o robô (localizado acima) e obtém-se uma altura total do sistema de 850 milímetros - a mesma altura de uma bancada de trabalho comum no LAR.

O passo seguinte envolvia a escolha do tipo de estrutura a utilizar, a sua localização e o seu dimensionamento. O material disponível na oficina do Departamento de Engenharia Mecânica (DEM) foi um fator que influenciou significativamente a escolha do tipo de estrutura a utilizar. Existindo chapa da liga de alumínio 5083 H111, de 10 milímetros de espessura, escolheu-se esse elemento como elemento estruturante.

No dimensionamento das placas de suporte tentou encontrar-se uma combinação entre resistência mecânica, visibilidade para o interior e estética. A adoção de uma treliça foi o recurso que melhor conseguiu conciliar todos estes fatores. O *Design for Manufacturing* (DFM) foi ainda levado em conta na medida em que se evitaram esquinas vivas ou o uso de determinadas características que implicassem complexas gamas de maquinagem.

O principal fator que influenciou a localização dos elementos estruturantes foi a montagem do sistema, sendo que para que o sistema seja montável, a localização da estrutura de apoio não pode estar sobre a zona a vermelho da figura 3.2. Desta forma, tendo em conta o *Design for Assembly* (DFA) e tentando sempre maximizar o espaço útil no nível intermédio (representado a amarelo na figura 3.2), garantindo uma fácil acessibilidade desse espaço, surgiu uma proposta de solução que pode ser observada na figura 3.9.

As figuras 3.4a e 3.4b representam os dois elementos usados na estrutura. A utilização de diferentes elementos está relacionada não só com o aproveitamento do espaço, como também com as limitações estruturais da placa inferior. Idealmente, deveriam ser utilizados elementos idênticos ao primeiro, no entanto, a estrutura que os apoiaria não permite a sua implementação, tendo que se recorrer ao segundo elemento.

3.2 Simulação

Após o projeto, e antes da sua fabricação, importa verificar se a estrutura resiste mecanicamente à carga a que vai ser sujeita. Dado que a simulação global do sistema é um processo muito complexo - devido a todas as interações entre partes e principalmente devido as ligações mecânicas através de parafusos -, decidiu-se realizar um conjunto de simulações utilizando elementos parciais da estrutura global.

O método de simulação utilizado passou pela avaliação da força máxima que cada elemento consegue suportar até que seja atingida a tensão de cedência do material. Para cada elemento, foram empregues forças pontuais nas 3 direções ilustradas nas figuras 3.5a

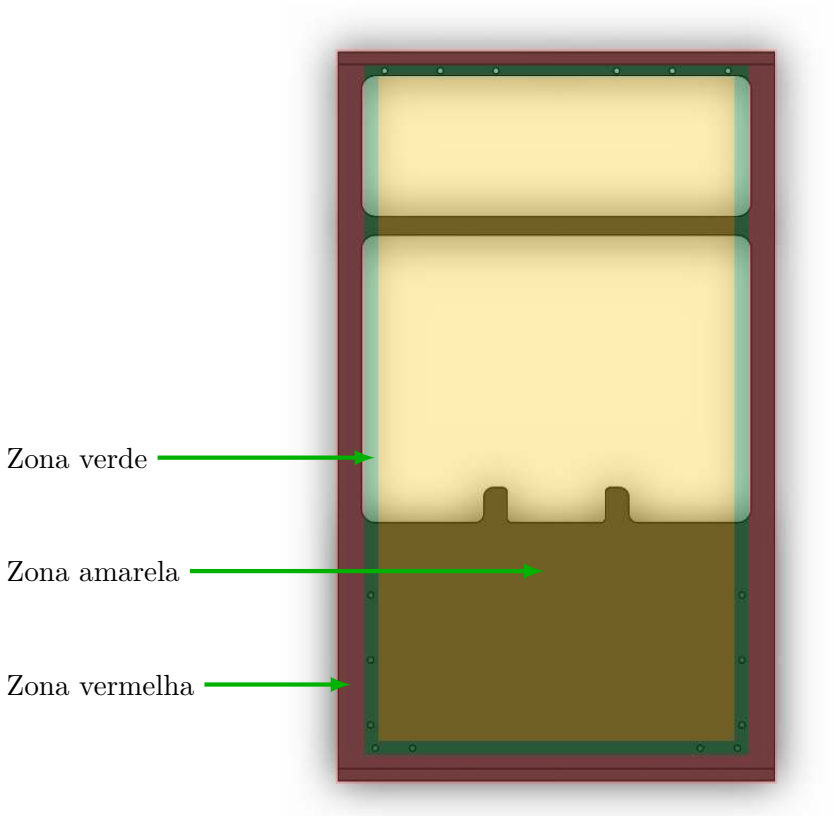


Figura 3.2: Representação de zonas da placa estrutural preta
Zona verde: Área de suporte disponível e rentável
Zona amarela: Área disponível mas não rentável
Zona vermelha: Área indisponível para suporte

e 3.5b. Admitindo uma tensão de cedência de 295 MPa [29], os resultados das simulações encontram-se expressos na tabela 3.1.

Como esperado, os resultados acabaram por corresponder às expetativas. Do ponto de vista mecânico, o segundo elemento estruturante é mais frágil do que o primeiro, sendo que a sua principal limitação é a tolerância à carga na direção C. Por outro lado, o primeiro elemento apresenta uma resistência mecânica maior, pelo que se prevê que compense a estrutura mais frágil quando montado o sistema global.

Focando a análise no sistema completo, a estrutura global suporta os seguintes valores de carga:

1. Compressão

$$\begin{aligned} F &= 2 \times 4815 + 2 \times 11904 \\ &= 33438\text{N} \end{aligned} \tag{3.1}$$

2. Direção do comprimento da plataforma

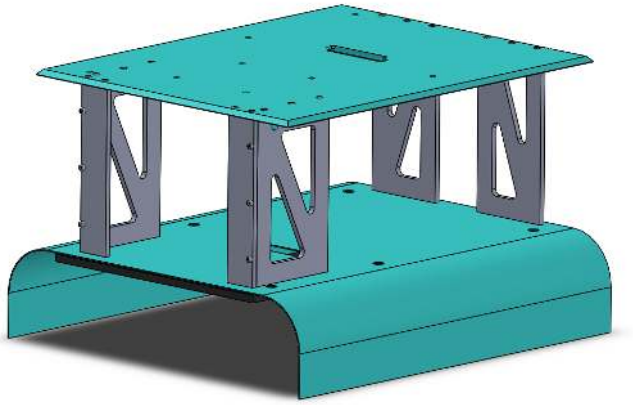
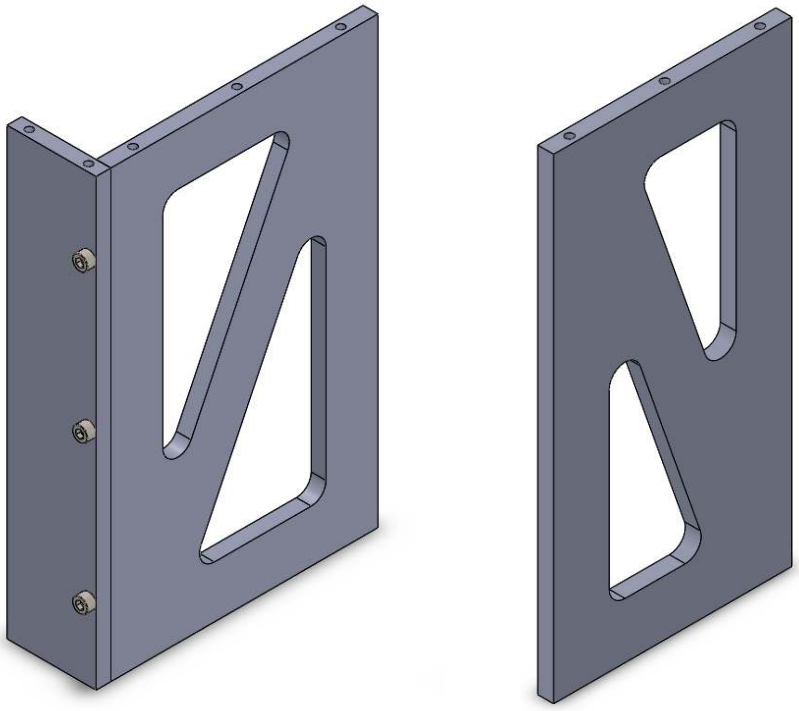


Figura 3.3: Representação da estrutura final

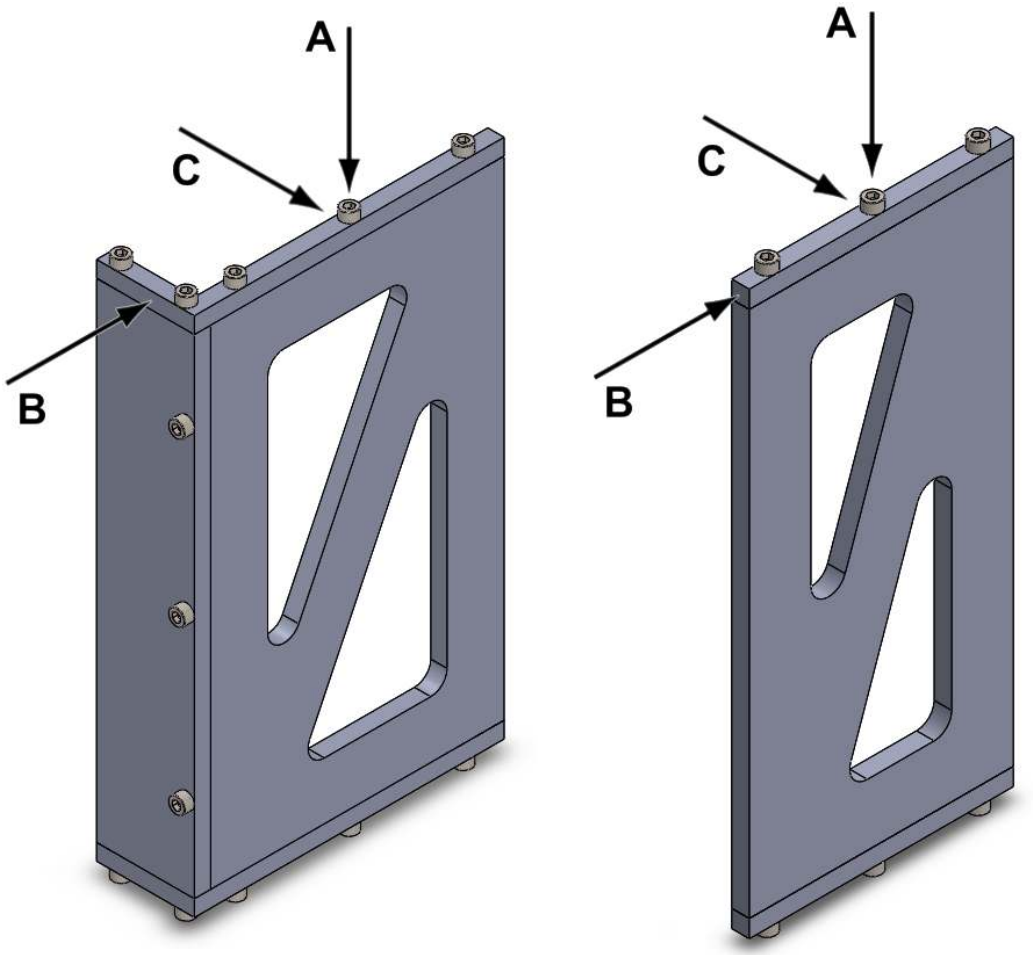


(a) Elemento 1 (b) Elemento 2

Figura 3.4: Elemento estruturantes projetados

$$\begin{aligned} F &= 2 \times 1326 + 2 \times 83 \\ &= 2818\text{N} \end{aligned} \tag{3.2}$$

3. Direção da largura da plataforma



(a) Infraestrutura de simulação 1 (b) Infraestrutura de simulação 2

Figura 3.5: Elemento estruturantes projetados

$$\begin{aligned} F &= 2 \times 787 + 2 \times 402 \\ &= 2378\text{N} \end{aligned} \tag{3.3}$$

Com esta análise, facilmente se percebe que a estrutura de elevação de nível suporta uma carga muito superior a que estará sujeita, que será apenas o peso do robot (25 kg), o seu payload (7kg), mais os eventuais objetos que possa transportar e toda a dinâmica associada. A força de corte máxima em ambas as direções (B e C), está também consideravelmente acima do valor máximo esperado de carga a que o sistema vai estar sujeito, confirmando a previsão tomada.

Assim, foram fabricados todos os componentes projetados e furados todos os outros contemplados na figura 3.1, de forma a permitir a ligação mecânica das placas e a montagem de todo o sistema robótico integrado.

	El. Estruturante 1	**El. Estruturante 2**
Compressão(Dir. A)	11904 N	4815 N
Corte (Dir. B)	1326 N	787 N
Corte (Dir. C)	402 N	83 N

Tabela 3.1: Resultado das simulações da carga máxima que os elementos suportam nas três direções principais

3.3 Atualização do modelo URDF

Associado ao ROS industrial, o URDF [30] contém informações e formatos utilizados para a representação do modelo virtual de um robô e o do movimento relativo entre os elos que o compõe. Para esta definição, é requirido o modelo 3D de todos os elos de um robô, que serão devidamente referenciados de acordo com a *home position* do sistema global. Posteriormente, a partir de um elo fixo, são definidos todos os movimentos relativos (juntas rotacionais ou prismáticas) entre os sistemas de referência de cada elo até se chegar ao último elo correspondente ao *gripper* do robô.

Concluído o modelo URDF do robô, a sua utilização passa pela verificação de colisões entre os próprios elos do robô e/ou com o meio envolvente e também pela visualização no estado do conjunto em tempo real. O algoritmo de cinemática inversa, *IKFast*, planeia também as trajetórias e faz os cálculos do espaço de juntas a partir do ficheiro macro, onde são definidos todos os parâmetros necessários à conceção do modelo virtual [21].

Ainda durante esta fase que envolvia modelações 3D, foram atualizados alguns dos modelos CAD utilizados pelo URDF, sendo eles o *gripper* e a base do robô.

No primeiro elemento (figura 3.6) apenas se fizeram pequenas alterações no sentido de ajustar o modelo às medidas reais da ponta do manipulador que, no decorrer de alguns trabalhos passados, sofreu algumas modificações. A correspondência exata do *Tool Center Point* do modelo virtual com o robô real também teve que ser assegurada com a alteração do ficheiro *lrmate200id.urdf*, nomeadamente o campo homólogo ao último elo (*gripper*).

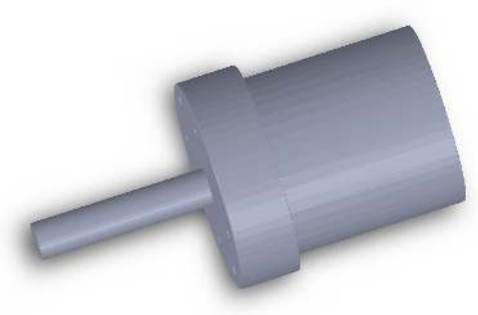


Figura 3.6: Modelo CAD do gripper do modelo virtual do robô

O segundo elemento (figura 3.7) foi mais trabalhoso, na medida em que se teve que desenvolver um sólido CAD, onde está presente tanto a base do robô como também a placa estrutural superior. Esta foi introduzida devido ao facto de ser um elemento constituinte do sistema robótico, que estará sujeito ao contacto com o manipulador em

operações de manipulação. Desta forma, achou-se por bem a sua inclusão, dado que os modelos são usados para deteção de colisão entre os próprios elos que fazem parte do modelo URDF do sistema.

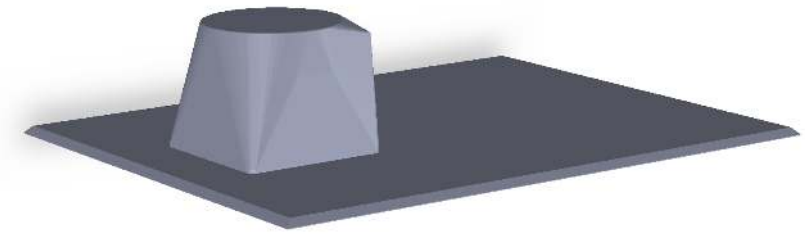


Figura 3.7: Modelo CAD da base do modelo virtual do robô

Durante o processo de desenvolvimento virtual 3D das estruturas, foi levado em conta o peso computacional associado à verificação de colisões. A resolução dos modelos STL - formados por uma malha de triângulos gerada a partir dos modelos CAD - foi reduzida ao máximo com vista a evitar longos tempos de verificação de colisão. Já nos modelos usados apenas para fins visuais, na interface RViz, não foi preciso este cuidado, tendo-se usado modelos com uma resolução agravável à vista do utilizador.

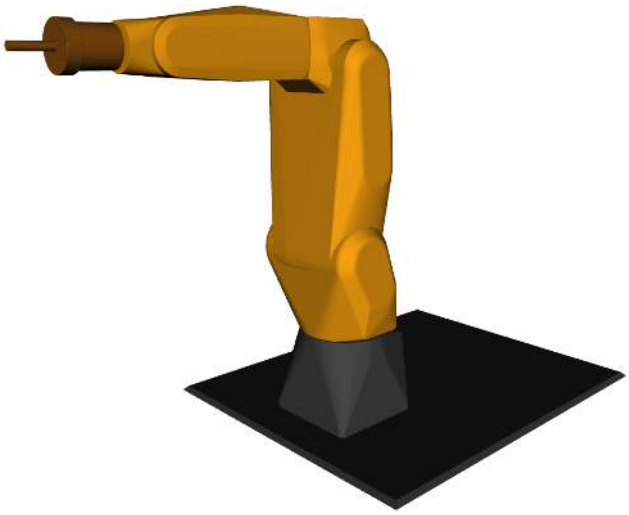


Figura 3.8: Modelo virtual do robô na interface RViz

O resultado da atualização do modelo URDF do robô está ilustrado na figura 3.8. Pode observar-se os dois modelos CAD descritos e ainda ter a perceção que a Tool Center Point (respresentada numa esfera a azul) usada em cálculos de cinemáticas coincide com o modelo virtual, o que comprova a correta configuração dos parâmetros macro e a correta integração dos sólidos 3D.

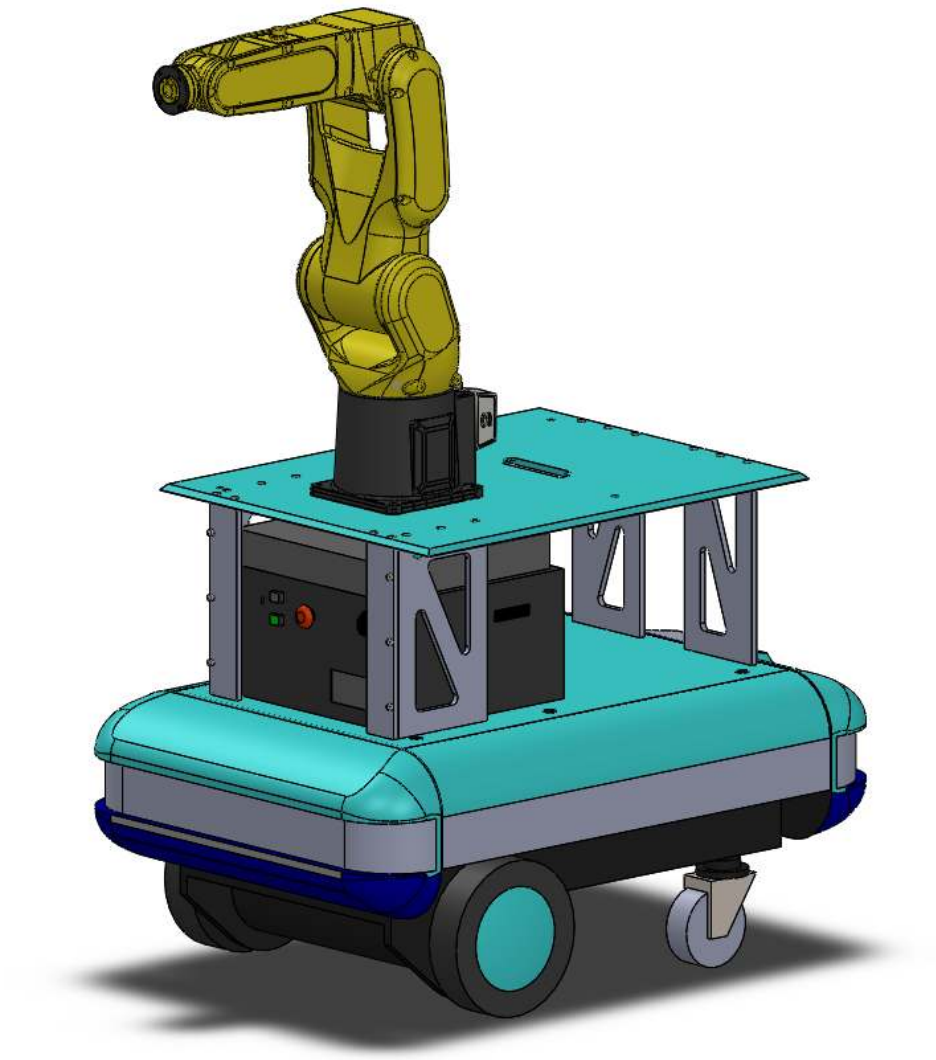


Figura 3.9: Modelo do sistema robótico integrado [31]

Capítulo 4

Arquitetura de *Software*

Neste capítulo descrevem-se os avanços ao nível do *software*. Inclui todo o desenvolvimento da arquitetura de controlo e monitorização do sistema robótico integrado, configurações ao nível do ROS FANUC para otimização do movimento do *gripper* e ainda a criação de um toolkit de funções para uma utilização mais eficaz da biblioteca MoveIt!.

4.1 *Software* desenvolvido

Como ponto de partida para a conceção de uma arquitetura de *software*, haviam alguns requisitos a ser considerados. Pretendia-se o desenvolvimento de uma arquitetura robusta e modular, que fosse ao encontro da filosofia ROS. Era também requirido que estivessem integrados três nodos com as seguintes funções: um de interface com o utilizador, um outro de simulação de controlo da plataforma e o último de controlo do manipulador através da biblioteca MoveIt!.

A infraestrutura de *software* proposta inicialmente encontra-se representada na figura 4.1. Numa fase mais avançada dos trabalhos foram adicionadas novas funcionalidades ao *software* e, como tal, a complexidade da estrutura aumentou no sentido de dar resposta a essas novas aplicações.

Na figura podem identificar-se dois tipos essenciais de elementos: ovais e retangulares. Os ovais representam os nodos criados para a arquitetura de controlo do sistema, com exceção do elemento verde que representa a arquitetura de base ao ROS Industrial, responsável pela comunicação com o controlador do manipulador. Já os elementos retangulares representam todos os tópicos e serviços que estabelecem a comunicação entre os nodos da arquitetura desenvolvida. Nas secções seguintes encontra-se a descrição detalhada do funcionamento dos nodos incluídos no diagrama.

4.1.1 `vs_keyboard`

O primeiro nodo da arquitetura ROS desenvolvida chama-se `vs_keyboard` devido à sua principal função, que é adquirir *inputs* do teclado, inseridos pelo utilizador, e envolve-los na estrutura de *software* restante. Para tal, este nodo publica um tópico `/chatter` que vai ser subscrito pelo nodo `vs_platform_sim` descrito na secção 4.1.2.

As mensagens informam o nodo subscritor acerca da função que se pretende que o sistema robótico desempenhe, nomeadamente o movimento ou monitorização do manipulador, interface com I/O's (secção 5.6) ou o movimento da plataforma. Definem ainda

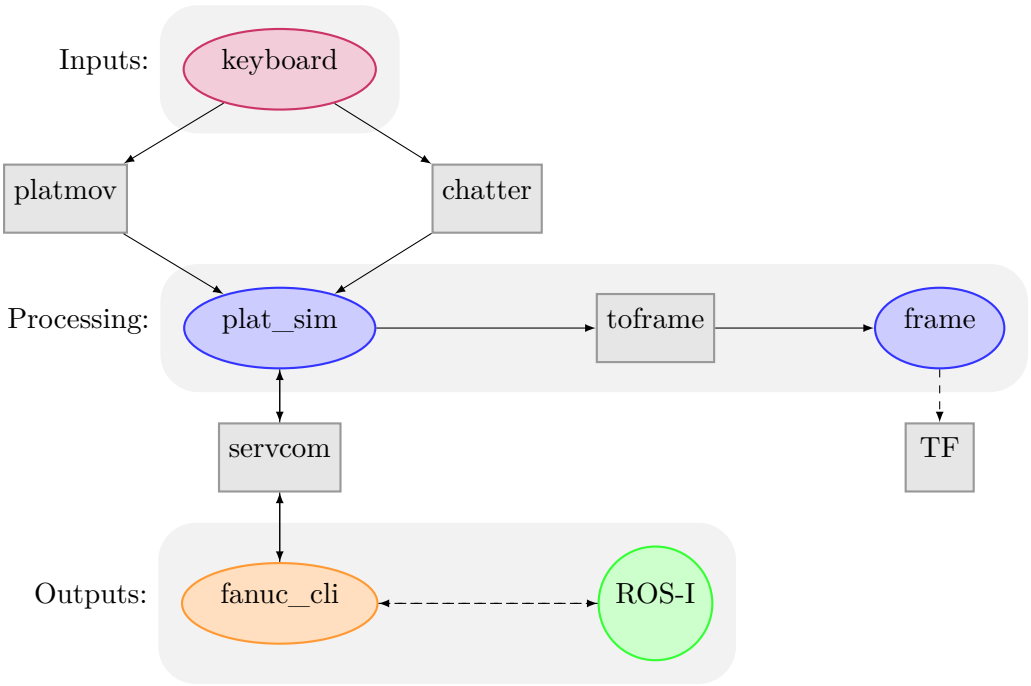


Figura 4.1: Infraestrutura de software ROS (nodos a oval e tópicos/serviços a retangular)
Elementos ovais: Nodos
Elementos retangulares: Tópicos/Serviços
Elemento circular: Arquitetura ROS Industrial

algumas variáveis, caso sejam necessárias. No último caso, de forma a que utilizador possa deslocar a plataforma pelo referencial global, foi criado um novo tópico (`/platmov`) que publica valores correspondentes a cada comando do teclado.

4.1.2 `vs_platform_sim`

O nodo `vs_platform_sim` foi criado com o propósito de simular o nodo de controlo da plataforma. Desempenha funções associadas a dois grupos, nomeadamente ao movimento e monitorização da posição da plataforma e ao pedido da execução de tarefas por parte do manipulador.

No primeiro grupo incluem-se variadas tarefas, tais como a subscrição do tópico que traduz qual a ordem pressionada no teclado, a conversão desses valores em valores de coordenadas no espaço e ainda o envio desses valores, através do tópico `/toframe` para o nodo `vs_frame`, descrito em 4.1.4.

No processo de conversão usou-se um método incremental, em que as setas horizontais definem a direção para o qual se pretende mover o sistema e as setas verticais definem o sentido do movimento. Esta forma, apesar de não se assemelhar na íntegra à realidade, simula o movimento da plataforma pelo espaço; todavia, no futuro, este processamento não será necessário e a localização da plataforma será dada por sensores e eventuais cálculos associados (ex: odometria).

O segundo grupo de tarefas está diretamente relacionado com o serviço `/servcom` que é a forma de comunicação com o nodo `vs_fanuc_client`. É através deste tipo de comunicação bidirecional que o nodo vs_platform_sim pede a execução de tarefas por parte do manipulador (pedido) e que recebe que a tarefa já foi concluída (resposta).

Quando determinado comando chega a este nodo através do tópico `/chatter`, imediatamente é processado e, dependendo da função que lhe está associado, pode ser encaminhado para o nodo `vs_fanuc_client` (descrito em 4.1.3) juntamente com as variáveis imprescindíveis à realização dessas funções. A título de exemplo, se for pretendido o movimento em juntas por parte do manipulador, será indispensável o envio do comando agregado a essa função e dos valores das juntas para o qual se pretende que o robô se mova.

Será este nodo que fará a ligação entre a arquitetura desenvolvida no decorrer deste projeto e o trabalho desenvolvido no controlo da plataforma Robuter.

4.1.3 `vs_fanuc_client`

O nodo `vs_fanuc_client` é responsável por interagir diretamente com o controlador do manipulador através da biblioteca MoveIt! e do ROS FANUC. Aquando a inicialização do nodo, é estabelecida a ligação ao ROS-I que está previamente em execução no ambiente ROS do computador; após esse momento, a biblioteca MoveIt! pode ser utilizada com o propósito da interação com o ROS FANUC em execução no controlador do manipulador.

Após a receção de um comando e das variáveis necessárias à sua execução (ambos chegados envolvendo uma comunicação *server-client* através do serviço `/servcom`) uma das funções do Toolkit FANUC (secção 4.3) será utilizada, envolvendo, na mesma linha, a comunicação *server-client* com o controlador.

Quando a tarefa que o robô deve desempenhar estiver terminada, o nodo `vs_fanuc_client` irá enviar no sentido inverso, através do mesmo serviço, uma resposta que confirma o fim

de funções, dando luz verde à plataforma para continuar com a navegação do sistema, na certeza que o manipulador já terminou a execução das tarefas que lhe tinham sido incumbidas.

4.1.4 `vs_frame`

Por último, o nodo `vs_frame` é responsável pela gestão dos sistemas de referência. Numa primeira fase apenas publica a transformada do referencial global que é fixo (`world`) para o referencial da base do robô (`base_link`). As coordenadas (x,y,z) e a rotação (w) são fornecidas pelo nodo `vs_platform_sim` através do tópico `/toframe`. Subsequentemente, são convertidas numa matriz de transformação (TF) que é publicada e intersetada na interface RViz (figura 4.2) sendo possível, desta forma, observar todo o sistema em movimento relativamente ao mundo.

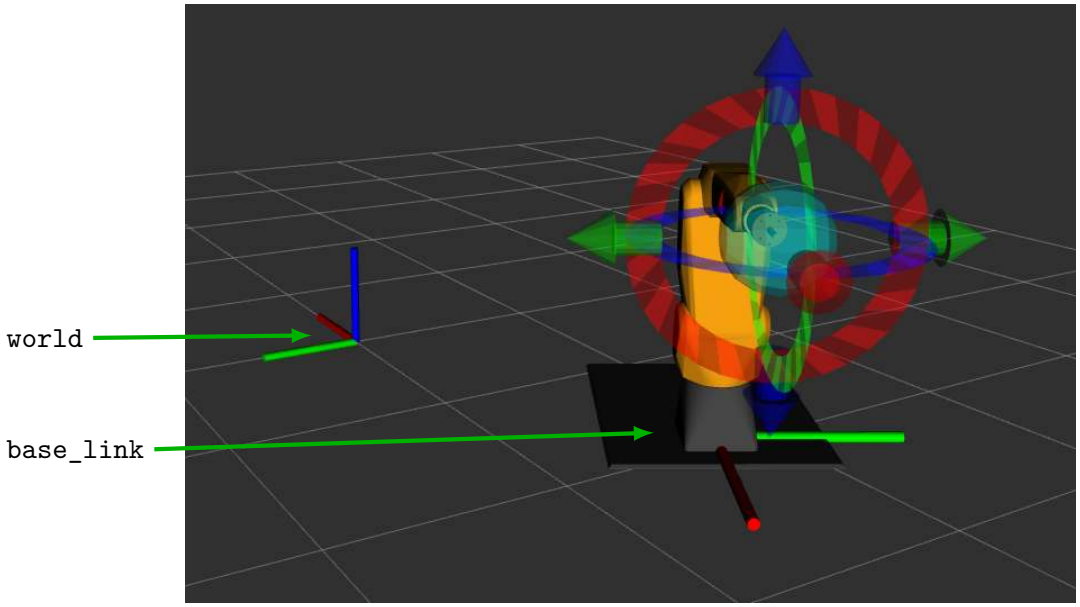


Figura 4.2: Representação do referencial do robô deslocado em relação ao referencial global

4.2 Otimização de movimento

O manipulador FANUC utilizado nesta dissertação apresenta 6 graus de liberdade, ou seja, salvaguardando alguma exceções, para cada ponto no seu espaço de trabalho, o número de combinações de valores de juntas que satisfazem essa posição é limitado. Por outras palavras, a resolução da cinemática inversa para uma dada posição e orientação tem um número finito de soluções. Não há dúvidas que a biblioteca MoveIt é uma mais valia no controlo de robôs; de facto, são inúmeras as opções que oferece, permitindo a programação *offline* de um manipulador e possibilitando ainda a visualização 3D do planeamento de trajetória, simulações, detecção de colisões, etc.

No entanto, apesar de todas as suas vantagens, também possui limitações. Sendo uma biblioteca genérica - que funciona para qualquer robô - a sua utilização está limitada em

alguns aspetos. Ao invés de usar a cinemática inversa, pela análise do código do algoritmo de cálculo da cinemática inversa (IKFast), percebeu-se que a biblioteca MoveIt usa apenas a matriz jacobiana invertida gerada a partir do ficheiro lrmate200id.urdf. Desta forma, é usada a cinemática inversa incremental para obtenção das variáveis de juntas a partir das variáveis cartesianas. Esta aproximação implica que não seja possível escolher a redundância com a qual o robô chega à posição pretendida, o que dificulta o planeamento de trajetórias. Além disso, quando a trajetória de um ponto inicial para outro final envolvia um deslocamento longo do manipulador, o algoritmo por vezes era inviável, não conseguindo chegar a uma solução de trajetória que enviasse o robô para a posição pretendida. Nestes casos, era necessário mover o robô para uma posição intermédia e só depois para a posição que se pretendia realmente.

A solução encontrada para de alguma forma contornar o problema foi a criação de posições *default* em localizações estratégicas e bastante usuais nas operações de manipulação que se irão realizar com o sistema robótico. Assim sendo, para longas trajetórias, promove-se um movimento preliminar no espaço de juntas para uma das posições especificadas, nomeadamente a mais próxima da posição final pretendida. Desta forma, garante-se que o manipulador chega a esta posição na redundância que se pré definiu, sendo que, a partir desse ponto, é possível o movimento para a posição final desejada, no espaço cartesiano, sem que a biblioteca tenha de fazer planeamentos complexos que, muitas vezes, resultavam em movimentos desnecessários e ineficientes. Note-se que a detecção de colisão é realizada tanto em movimentos no espaço de juntas como em movimentos no espaço cartesiano, não se estando com esta solução a abdicar de qualquer prevenção ou segurança.

Outra das limitações é que, no movimento de um ponto para outro, como se está a usar uma cinemática inversa incremental, a biblioteca cria pontos intermédios de passagem para conseguir aproximar a posição de chegada real à calculada. O problema está no facto de o movimento ao longo desses pontos ser disputado pelo próprio controlador, no programa ROS_MOVESM.TP, com base nos pontos recebidos através protocolo *simple message* [21]. Ora, como seria de esperar, o movimento entre o ponto inicial e final deixa de ser fluido, sendo um movimento intermitente, com variações de velocidade evidentes nos pontos intermédios.

Este último problema não tem uma solução única, no entanto o movimento foi otimizado com a alteração de alguns parâmetros do ROS FANUC, nomeadamente a aproximação de passagem do *end-effector* a cada um dos pontos intermédios. Reduziu-se com efeito o correspondente parâmetro (move_cnt) de 50% para 10%, resultando um movimento bastante mais fluido e natural a custo de insignificantes desvios da trajetória que não têm quaisquer consequência.

4.3 Toolkit FANUC

Apesar da existência da biblioteca MoveIt! para controlo do manipulador, devido à falta de documentação, a sua utilização não é trivial e, por vezes, torna-se difícil para o utilizador perceber como se utilizam algumas funcionalidades. De forma a resolver este problema e a facilitar trabalhos futuros neste projeto, desenvolveu-se um conjunto de funções para interface com o manipulador FANUC. Integradas na classe *FanucCommands*, as funções desenvolvidas são as seguintes:

1. `id_position`

   Permite que o manipulador se mova, em juntas, para as posições pré-definidas e já referidas na secção 4.2;

2. `joints_position`

   Move em juntas para os valores definidos pelo utilizador que são parâmetros de entrada da função;

3. `xyz_position`

   Tal como a função anterior, trata-se de uma função para deslocamento do manipulador, neste caso, para um movimento em coordenadas absolutas que são de igual forma parâmetros de entrada;

4. `get_man_pos`

   É responsável por ler a posição absoluta atual da ponta do robô e retornar, numa estrutura de dados (RobotCoord), essa localização em relação ao referencial da base;

5. `get_sys_pos`

   Esta função tem exatamente a mesma funcionalidade que a anterior, no entanto, retorna a posição absoluta da ponta em relação a um referencial global deslocado em relação ao referencial de origem do robô. A transformada geométrica que descreve essa transformação é dada pelo nodo `vs_platform_sim`, que tem acesso às coordenadas da plataforma em relação a um referencial global.

As coordenadas que definem a plataforma são unicamente uma translação (t_x, t_y, t_z) e uma rotação (w), desta forma, para se saber a localização do *end-effector* (E) em relação a um referencial global (G), transformação essa definida por ${}^{G}\mathrm{T}_{E}$, usou-se a seguinte expressão:

$$ {}^{G}\mathrm{T}_{E} \;=\; {}^{G}\mathrm{T}_{R} \times {}^{R}\mathrm{T}_{E} \tag{4.1} $$

Através das coordenadas da plataforma podemos definir diretamente ${}^{G}\mathrm{T}_{R}$ que representa a transformação do referencial global para o robô (R).

$$ {}^{G}\mathrm{T}_{R} = \begin{bmatrix} \cos(w) & -\sin(w) & 0 & t_x \\ \sin(w) & \cos(w) & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2} $$

Para se obter a matriz de transformação global é preciso calcular a matriz de transformação do robô para o *end-effector*. Dada por ${}^{R}\mathrm{T}_{E}$, essa matriz tem que passar por um processo de conversão antes da sua definição.

Após a obtenção das coordenadas da ponta do robô em relação à base, usando o ROS FANUC, são retornados os valores (t_x, t_y, t_z) correspondentes à translação e os valores (q_w, q_x, q_y, q_z) correspondentes à rotação. Como se pode observar, a rotação está definida através do conjugado de um quaternião, uma extensão $\mathbb{H}$ do conjunto dos números complexos $\mathbb{C}$ [32]. A conversão deste sistema de localização para o sistema clássico que usa uma matriz de transformação 4x4 [33] é dada pela seguinte matriz:

$$ {}^{G}\mathrm{T}_{E} = \begin{bmatrix} 1-2q_y^2-2q_z^2 & 2q_xq_y-2q_zq_w & 2q_xq_z+2q_yq_w & t_x \\ 2q_xq_y+2q_zq_w & 1-2q_x^2-2q_z^2 & 2q_yq_z-2q_xq_w & t_y \\ 2q_xq_z-2q_yq_w & 2q_yq_z+2q_xq_w & 1-2q_x^2-2q_y^2 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3} $$

Desta forma, resta apenas a multiplicação das duas matrizes para obtenção da matriz representativa da transformação global conforme indicado em 4.1.

De forma inversa, pode ser usada uma transformação geométrica expressa numa matriz para definir uma posição do *end-effector* em relação ao referencial do robô. Dado que a biblioteca MoveIt! só aceita a definição da rotação por quaterniões, deverá previamente efetuar-se a seguinte conversão:

Admitindo uma transformação genérica dada por $\begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ rapidamente se chegam aos valores de (x,y,z) e (q_w, q_x, q_y, q_z) através das seguintes expressões [34]

$$ \begin{cases} x = t_x \\ y = t_y \\ z = t_z \\ \\ q_w = \dfrac{\sqrt{(1+r_{00}+r_{11}+r_{22})}}{2} \\ q_x = \dfrac{r_{21}-r_{12}}{4q_w} \\ q_y = \dfrac{r_{02}-r_{20}}{4q_w} \\ q_z = \dfrac{r_{10}-r_{01}}{4q_w} \end{cases} \tag{4.4} $$

Capítulo 5

Controlo de I/O's

Este capítulo descreve o problema de acesso aos I/O's do manipulador FANUC. Numa primeira fase, são descritas as limitações e as vantagens associadas a esse controlo. Posteriormente, são enumeradas as metodologias abordadas para levar a cabo esse controlo dando especial foco à solução final que, por de momento ser a única viável, foi a implementada.

A solução passa pela introdução de uma nova unidade de controlo no sistema, pelo desenvolvimento de um circuito de controlo de relés e pelo desenvolvimento de *software*, nomeadamente na arquitetura ROS, no controlador do robô e na nova unidade de controlo. Todos estes passos são devidamente descritos ao longo deste capítulo.

5.1 Problema de acesso aos I/O's do manipulador

Relembrando a secção 1.4.1, a principal limitação na dissertação exposta era o controlo de saídas digitais do robô FANUC, uma ferramenta fundamental para a realização de operações de manipulação que envolvam a utilização do *gripper* de sucção a vácuo. Mesmo implementando outro tipo de *gripper* na ponta do robot - como por exemplo o elétrico - o controlo das saídas do robot é da mesma forma essencial.

O controlo de I/O's é então uma ferramenta essencial na medida em que permite aceder, controlar e comunicar com unidades periféricas, como por exemplo sensores e atuadores. No caso da sua implementação no manipulador, permitiria ainda a troca de *end-effector* entre operações, um funcionalismo que poderia ampliar o leque de operações possíveis de executar com o manipulador.

As *metapackages* de interação com o manipulador FANUC (ROS-I e MoveIt!) não estão ainda preparadas para o controlo de I/O's do robô, dessa forma foi necessária a procura de uma solução que resolvesse o problema. Até se chegar a uma solução final viável, foram abordadas duas potenciais soluções mas que, devido a fatores incontroláveis, tiveram que se abandonar. Descrevem-se de seguida estas metodologias.

5.1.1 ModBus

Uma das formas de comunicação/controlo de I/O's do manipulador é a utilização do protocolo ModBus TCP/IP. Trata-se de um protocolo que define uma estrutura de mensagem que circula na camada da aplicação do modelo *Open System Interconnection* (OSI) [35].

A sub-mensagem definida pelo protocolo é associada ao nível da *Application Data Unit* (ADU); contém informação relativa ao número da transação, tamanho da mensagem, identificação do dispositivo de destino, etc. É posteriormente integrada nos níveis inferiores da mensagem formando assim um pacote ModBus TCP/IP [36] que será transmitido, neste caso, através do meio físico Ethernet.

Na figura 5.1 pode ver-se a construção completa de um pacote Modbus TCP/IP. Encontram-se representados os vários níveis da mensagem, que são sucessivamente encapsulados nos níveis inferiores, mais abrangentes. Estes acrescentam informação essencial para a comunicação entre dispositivos, nomeadamente o seu IP, porta de comunicação, etc.

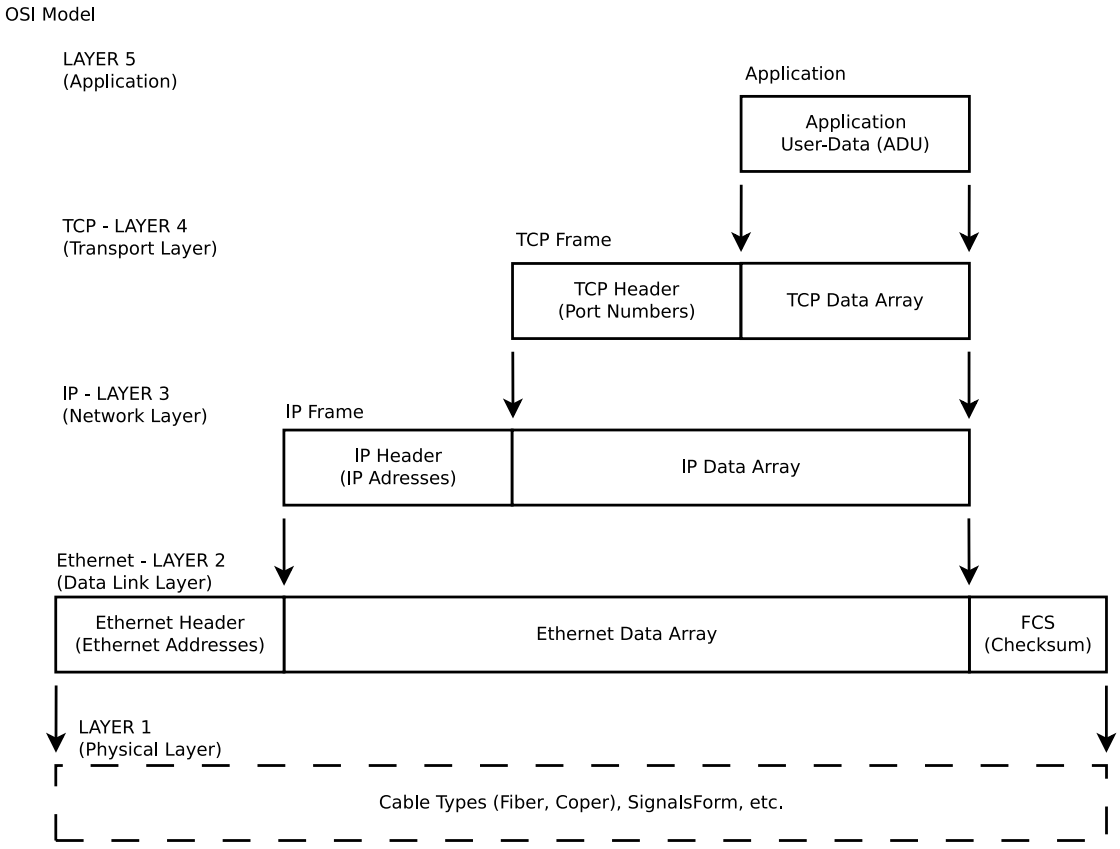


Figura 5.1: Construção de um pacote Modbus TCP/IP

Na tabela 5.1 pode observar-se uma das possíveis sub-mensagens correspondentes ao nível da aplicação, neste caso, com a função de ativar o primeiro registo digital de um dispositivo que esteja à escuta deste tipo de mensagens.

Número transação		**Ident. do Protocolo**		**Tamanho Mensagem**		**Ident. Disp.**	**Função**	**Primeiro Registo**		**Valor**	
0x00	0x01	0x00	0x00	0x00	0x06	0x01	0x05	0x00	0x00	0xFF	0x00

Tabela 5.1: Primeiro nível da mensagem Modbus

O protocolo adota o modelo arquitetural servidor-cliente [36] em que a comunicação começa com o envio de um pedido, por parte da unidade cliente, para outra servidora à escuta na porta 502 (*default* do protocolo Modbus). O dispositivo servidor encarrega-se de desempenhar a função que lhe foi pedida, nomeadamente a alteração do valor de posições de memória ou o retorno do seu valor para o cliente e, por último, a comunicação acaba com o envio de uma mensagem de resposta do dispositivo servidor para o cliente.

Com o objetivo da utilização do protocolo Modbus para interface com o manipulador LR Mate 200iD, foi desenvolvida uma aplicação cliente que utiliza este protocolo para comunicação o robô (servidor). A aplicação dispunha de uma interface gráfica (figura 5.2) que permitia ao utilizador um controlo e monitorização eficiente de várias variáveis, nomeadamente variáveis associadas a entradas e saídas digitais.

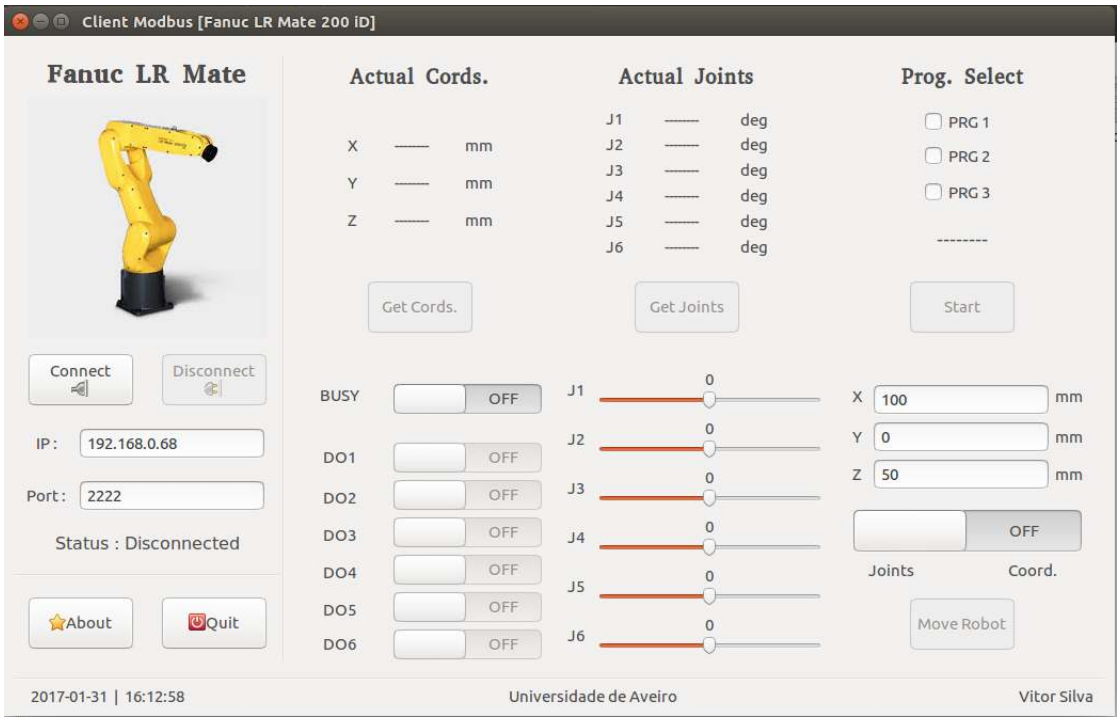


Figura 5.2: Interface gráfica para interface com o manipulador FANUC LR Mate 200iD através do protocolo ModBus [37]

Devido a uma impossibilidade na obtenção do *software* necessário para instalar o protocolo no controlador do manipulador, esta opção tornou-se inviável. Apesar do trabalho ter ficado disponível para uma possível utilização futura (aquando a instalação do protocolo se concretizar) esta ferramenta teve que ser posta de lado e a procura de uma solução alternativa tornou-se indispensável.

5.1.2 Server FanucRosCGIO

Com a impossibilidade da instalação do protocolo ModBus, foi necessário procurar uma outra forma de resolver o problema. Depois de uma pesquisa intensiva, observou-se que existe um servidor que se pode instalar no robô e que funciona paralelamente ao servidor necessário para o funcionamento do ROS. Esta à espera que um cliente se conecte

com ele no sentido de atuar e monitorizar o estado variáveis lógicas. Está opção foi também confirmada e sugerida pelo autor do *software* ROS FANUC G.A. vd. Hoorn. Num dos e-mails trocados com o autor, percebeu-se que o problema já era recorrente e, por esses motivos, este decidiu criar um servidor para determinados controladores FANUC que se encarrega de satisfazer essa necessidade dos utilizadores.

Para se compreender o funcionamento desta ferramenta é necessário entender dois pontos chave:

1. Servidor

   Como se referiu anteriormente, o servidor fica em execução no controlador do manipulador sem interferir com a utilização do ROS FANUC, pelo que pode funcionar juntamente com ele, sem interferências. Ao receber uma mensagem do cliente, é feito um processamento e, dependendo da mensagem, é alterado o estado de qualquer saída digital pré configurada para o efeito, ou retornada para o cliente uma nova mensagem com o estado de qualquer I/O.

2. Cliente

   Do lado do PC, com o objetivo de comunicar com o servidor, tem-se o cliente. O autor do servidor desenvolveu também um cliente, em linguagem Python, já preparado para comunicar com a aplicação que o escuta para efetivar a interface com I/O's. A aplicação inclui uma biblioteca que torna mais fácil para o utilizador a codificação e descodificação de mensagens enviadas e recebidas respetivamente, com funções específicas, sejam elas para leitura ou escrita de valores no servidor.

De forma a implementar este método, recorreu-se ao simulador FANUC (Roboguide) para compilar o ficheiro `ros_cgio.kl`, escrito em linguagem Karel e descarregado de um repositório *online* [38]. Depois de compilado, o ficheiro `ros_cgio.pc` resultante foi transferido para o controlador através do *file transfer protocol* já referido na secção 2.2. O processo da transferência do ficheiro encontra-se representado na figura 5.4.

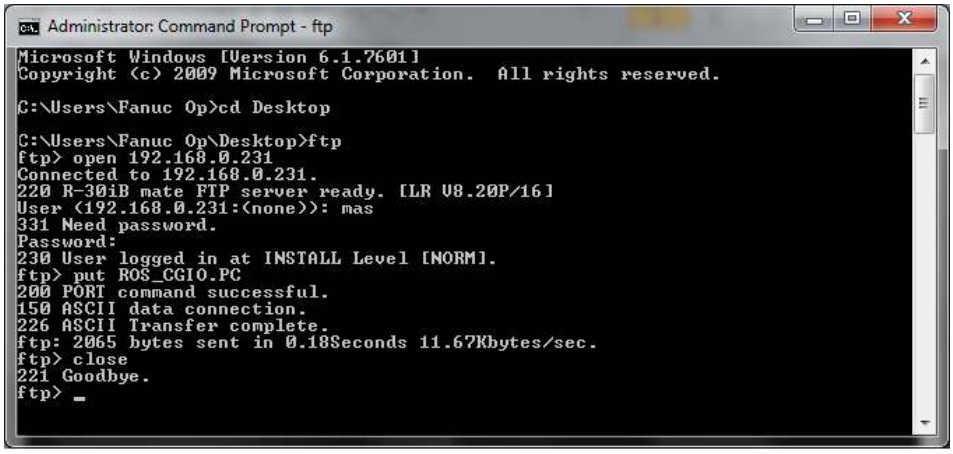


Figura 5.3: Transferência de ficheiro por *ftp* do PC para o controlador R-30iB

No entanto, depois do programa compilado e transferido, no momento da sua execução foram encontrados alguns problemas. Estes ficaram explicados quando se verificou que,

para o servidor funcionar, era necessária a instalação de uma opção da FANUC que, tal como o ModBus, não estava instalada no controlador. A opção em causa é a *Web Server Enhancements* (R626) e, dadas as circunstâncias, esta metodologia não pôde ser considerada e, portanto, foi necessário providenciar uma alternativa viável.

5.2 Metodologias adotada

Postos os problemas que inviabilizaram as duas opções anteriores, adotou-se uma metodologia alternativa que resolve perfeitamente o controlo de I/O's. A solução passa pela utilização de um microcontrolador, em comunicação com o computador, que através de um circuito comanda relés responsáveis por atuar entradas digitais do manipulador que, por sua vez, irão disputar saídas digitais.

5.2.1 Unidade de controlo

O Arduino Leonardo ETH [39] (figura 5.4) foi o controlador utilizado no desenvolvimento da solução. Trata-se de uma placa que integra o microcontrolador ATmega32U4 e o controlador Ethernet embebido W5500 TCP/IP. Dispõe de 20 I/O's, sendo que 7 podem ser utilizados como saídas PWM e 12 como entradas analógicas, um cristal com uma frequência de oscilação de 16MHz e diversos conectores, nomeadamente o *jack* de alimentação, a conexão RJ45 e o conector USB usado para transferir o programa para a memória flash do microprocessador.

A sua escolha é justificada por vários fatores que contribuíram ativamente para a facilidade e eficácia de implementação da solução. Destaca-se a porta Ethernet (RJ45) - por facilitar e uniformizar o método de comunicação entre todos os dispositivos do sistema -, pelo facto de ser uma placa pronta a utilizar - na medida em que já vem montada com o cristal, reguladores de tensão, conversores USB-RS232, etc -, e ainda pelo facto de existir uma enorme variedade de bibliotecas já desenvolvidas que facilitam a sua programação.

Figura 5.4: Arduino Leonardo ETH usado como unidade de controlo de I/O's

5.2.2 Código INO

A programação do microcontrolador passou pela elaboração de um ficheiro .ino, esquematizado a figura 6.9, que desempenha as seguintes funções:

1. Inicializar a conexão Ethernet

   Antes de se efetivar qualquer tipo de comunicação envolvendo o controlador Arduino, é necessário ativar a conexão ethernet atribuindo um IP para o dispositivo e definindo uma porta de comunicação; só a partir deste momento é que o microcontrolador fica apto para trocar mensagens ao abrigo do protocolo TCP/IP.

2. Inicializar um servidor

   Admitindo que se pretende uma comunicação entre dois processos em execução em diferentes dispositivos (PC e microcontrolador), e dado que as alternativas exploradas na secção 5.1 utilizavam uma arquitetura *server-client*, achou-se por bem usar uma arquitetura semelhante para fazer o controlo de I/O's do sistema. Com a conexão estabelecida, o passo seguinte foi a inicialização de um aplicação servidora, que fica à escuta de clientes na porta já definida.

3. Interagir com clientes

   Havendo um cliente que se conecte ao servidor, e enquanto a conexão for válida, o programa entra num ciclo responsável por receber, descodificar e processar mensagens, desde que a estrutura e valores estejam dentro dos limites definidos pelo utilizador. Os dados recebidos pelo servidor podem ter uma das seguintes funções:

   (a) Ligar uma saída
   (b) Desligar uma saída
   (c) Monitorização de uma saída

   Nos pontos (a) e (b) o programa encarrega-se da alteração do estado da saída com o número especificado na mensagem. Já no caso do ponto (c) o programa verifica o estado de uma saída (também especificada na mensagem) e responde ao servidor o seu valor lógico

5.3 Nodo `vs_IO_client`

O nodo responsável pela comunicação com unidade de controlo de I/O's tem o nome `vs_IO_client` e, como seria de esperar, trata-se de um *client* TCP/IP que se conecta ao Arduino através do mesmo IP e porta definidos no servidor, podendo haver uma comunicação bidirecional entre os dois processos em execução em diferentes dispositivos (comunicação por *sockets*).

O nodo subscreve um tópico (`client_messages`) que é publicado pelo nodo `vs_fanuc_client`. Quando o tópico é publicado, imediatamente é gerada uma *callback* no nodo subscritor que se encarrega de codificar uma mensagem que será enviada para a aplicação servidora em execução no dispositivo remoto. Como seria de esperar, a mensagem é codificada levando em conta a função pretendida (expressas em 5.2.2) e número do I/O a que se pretende aceder.

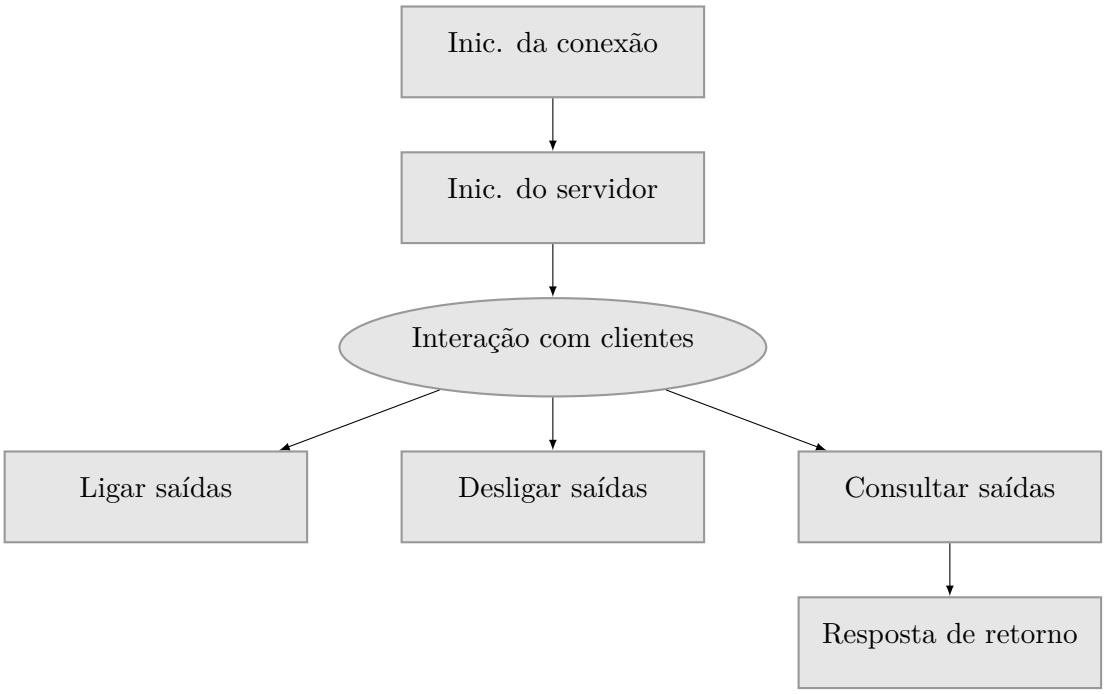


Figura 5.5: Representação esquemática do programa da unidade de controlo de I/O's

É a função `monitoring_ios` que está encarregue de publicar o referido tópico e, sendo que tem a incumbência do controlo de *hardware* no manipulador, achou-se por bem a sua integração no *Toolkit FANUC* já introduzido na secção 4.3.

5.4 Circuito de ligação ao controlador R-30iB

Nesta fase já se consegue, através do ROS, fazer um controlo das saídas do controlador Arduino. O próximo passo é converter as saídas microcontrolador em sinais que o controlador do robô consiga interpretar. Para tal, usou-se um circuito para relés (representado na figura 5.6), constituído por um LED de sinalização, uma resistência de 220 Ohm, um díodo UF4001, um transístor NPN BC337 e um relé SRD-05VDC-SL-C. Em função do estado de certa saída do controlador, o circuito é responsável por atracar e desatracar o relé que faz a ligação dos 24 V disponíveis no controlador a uma entrada digital do robô, acionada a 24 V.

A tabela 5.2 representa o *pinout* da ficha utilizada para *interface* com I/O's do manipulador.

O circuito completo que integra o Arduino, a placa desenvolvida para controlo de relés e a ligação às entradas do robô encontra-se representado no apêncice B. Apesar de não se terem ligados todos os 8 relés que a placa e o código estão preparados para suportar, o esquema ficou disponível e a sua integração é de fácil execução.

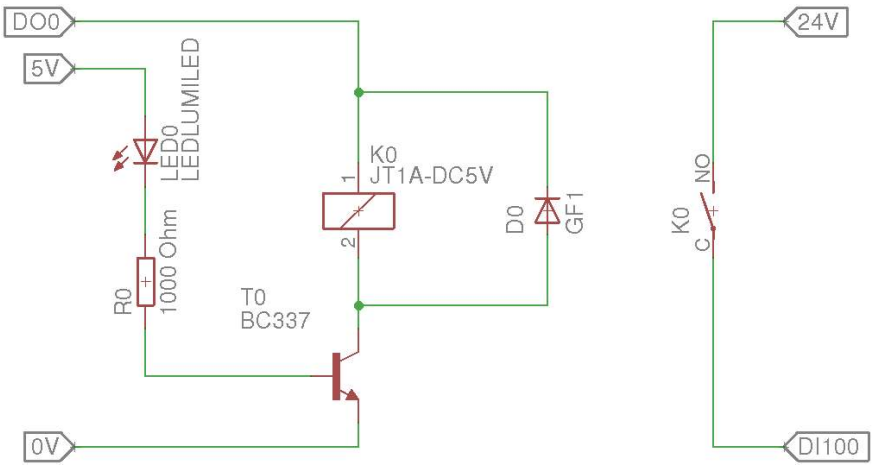


Figura 5.6: Circuito de cada relé

	B	A
1	DO101	DI101
2	DO102	DI102
3	DO103	DI103
4	DO104	DI104
5	DO105	DI105
6	DO106	DI106
7	DO107	DI107
8	DO108	DI108
9		SDICOM1
10		DI109
11		DI110
12		DI111
13		DI112
14		DI113
15		DI114
16		DI115
17		DI116
18		DI117
19	0V	DI118
20	0V	DI119
21	24F	DI120
22	24F	SDICOM2
23	24F	0V
24		0V

DI: *Digital Input*
DO: *Digital Output*
0V: 0 Volt
24F: 24 Volt
SDICOM1: *Selects a common for DI101 ∼ DI108*
SDICOM2: *Selects a common for DI109 ∼ DI120*

Tabela 5.2: *Pinout* da ficha CRMA62 do controlador FANUC R-30iB Open Air

5.5 Configurações no controlador

O último passo para o controlo das saídas do robot é mapear as saídas que se pretendem controlar às entradas que são controláveis através da arquitetura ROS desenvolvida. Para tal, desenvolveu-se um programa em linguagem TP que guarda em registos (variáveis do robot) os valores definidos pelo estado das entradas e logo de seguida, iguala esse valor ao estado de uma saída que se pretenda controlar.

Entre as referidas saídas está, por exemplo, a válvula que controla o ar comprimido no *end-effector* do robot. Acontece que a válvula é bi-estável e, para se alterar o seu estado, é necessário ligar uma saída e desligar outra. Assim sendo, programou-se o ficheiro TP de forma a um único *Input* controlar dois *Outputs* com estados contrários, evitando assim o uso de entradas desnecessárias.

Resta apenas a execução contínua do programa paralelamente à execução dos programas ROS necessários para o funcionamento do servidor. A solução passa pela utilização da opção *Background Logic* disponível nos controladores FANUC. Esta opção permite a execução contínua de programas, em *backgound*, sem interferir com eventuais programas que sejam executados ou estejam em execução no ambiente principal. A figura 5.7 mostra o conjunto das configurações necessárias ao robô para igualar saídas a entradas, ambas digitais, incluindo o programa TP utilizado.

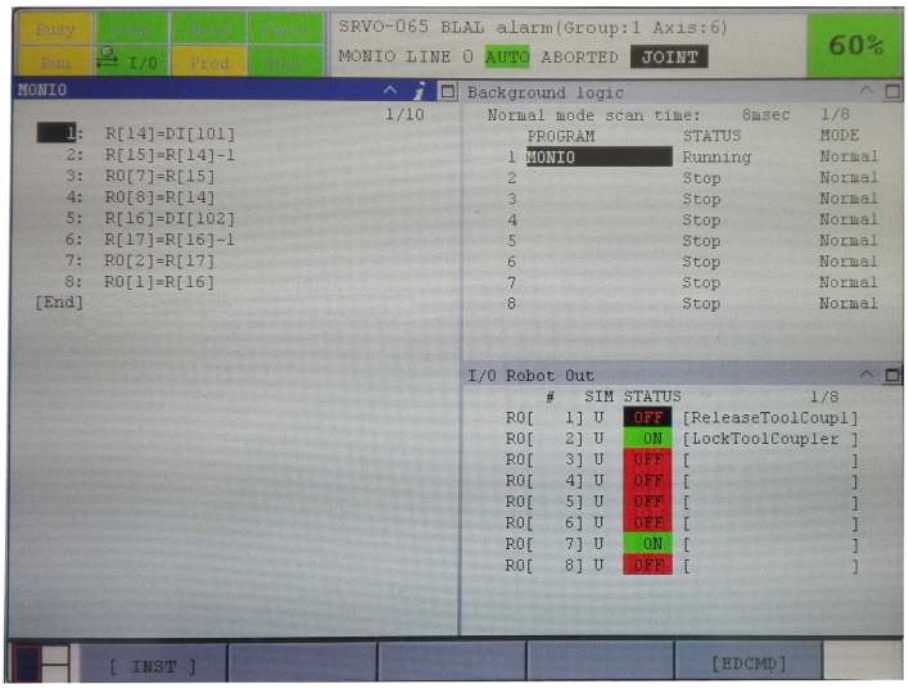


Figura 5.7: Programa TP para ligar saídas a entradas

Analisando a figura 5.8 e seguindo o exemplo da ação da ferramenta, pode verificar-se a seguinte situação: Sendo R[9] = DI[101], caso do valor da entrada digital DI101 seja "1", o registo número 9 (R[9]) toma o valor "1". Como R[10] = R[9]-1, o valor do registo R[10] será "0". Por sua vez, como RO[7]=R[10] e RO[8]=R[9], as saídas digitais do robô números 7 e 8 tomarão os valores "0" e "1", respetivamente. Tomando as saídas digitais os valores especificados, o *gripper* do robô será desacoplado do braço robótico. A mesma

dedução permite convalidar que caso a entrada digital DI[101] tome o valor "0", o estado das saídas digitais associadas serão opostos.

```
 1:   ;
 2:  !tool action ;
 3:   ;
 4:  R[14]=DI[101] ;
 5:  R[15]=R[14]-1 ;
 6:  RO[7]=R[15] ;
 7:  RO[8]=R[14] ;
 8:   ;
 9:  !change tool ;
10:   ;
11:  R[16]=DI[102] ;
12:  R[17]=R[16]-1 ;
13:  RO[2]=R[17] ;
15:  RO[1]=R[18] ;
16:   ;
```

Figura 5.8: Programa TP de mapeamento de saídas em função de entradas

5.6 Caixa de proteção

De forma a acomodar o circuito desenvolvido, a placa Arduino e todas as ligações associadas, foi desenvolvido um modelo CAD de uma caixa desenhada exclusivamente para guardar o material eléctrico necessário para a solução. Usando a técnica de prototipagem rápida de impressão 3D, foi criado um modelo físico da caixa que, depois de montado o conjunto, resultou no ilustrado na figura 5.10.

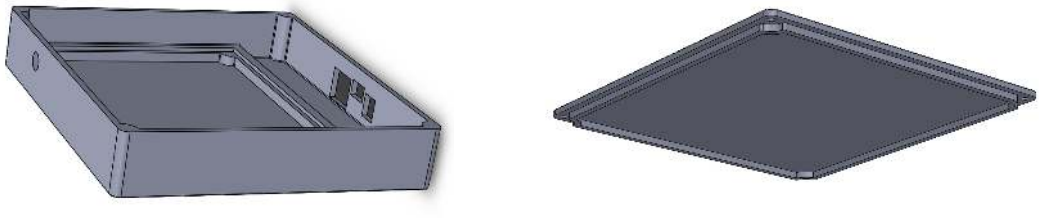


Figura 5.9: Modelos CAD dos componentes da caixa de acomodação

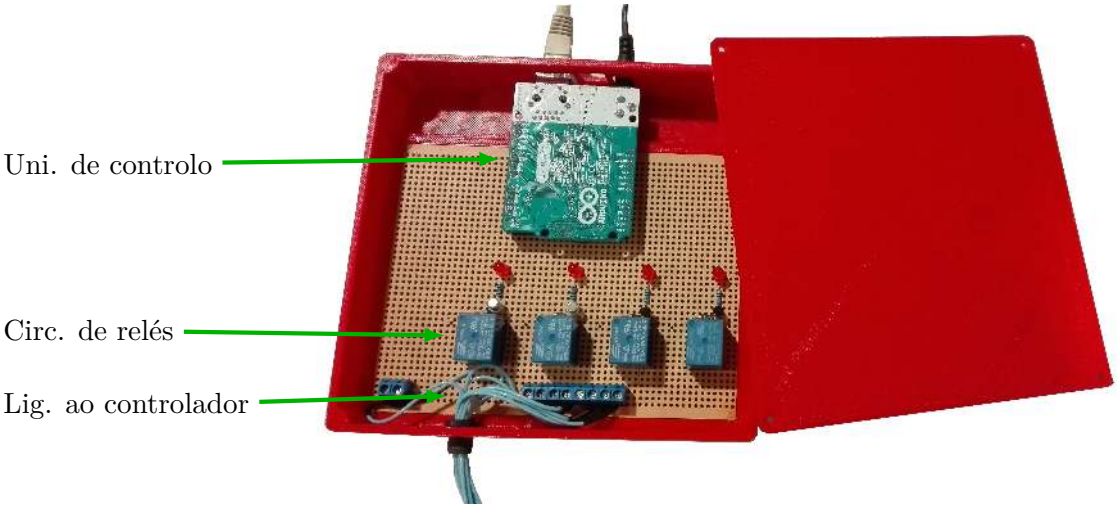


Figura 5.10: Caixa de acomodação da unidade de controlo de I/O's

Capítulo 6

Unidade de Proteção e Segurança

Neste capítulo é feita uma breve referência à importância da segurança e um levantamento dos dispositivos de segurança que se encontram, de origem, disponíveis no sistema robótico.

De seguida são descritos os sensores instalados com o objetivo principal do reforço da segurança do protótipo. Para tal, está envolvida a instalação e configuração dos sensores, a integração dos seus dados na arquitetura principal de *software*, o tratamento dos dados e o seu uso como segurança de operações de manipulação.

6.1 Importância e estado da segurança

No projeto de qualquer sistema robótico, a segurança é um aspeto fundamental que deve ser levado em conta no sentido de prevenir eventuais acidentes com o equipamento ou com humanos. Deverão, em todos os casos, ser tomadas medidas de forma a reduzir e/ou eliminar ricos, proteger os utilizadores contra os que não podem ser eliminados e informá-los dos residuais [40].

Como tal, a plataforma móvel já prevê dois dispositivos para o efeito, nomeadamente os *bumpers* e o botão de emergência. Apesar da sua existência, de momento apenas o botão de emergência se encontra instalado na plataforma. Este pode ser pressionado antes de haver contacto entre a plataforma e o obstáculo; no entanto, ao ser um dispositivo manual, pode não ser possível o seu acionamento, dada a velocidade ou mesmo a acessibilidade do mesmo. Os *bumpers*, por outro lado, são um método automático de segurança passiva pelo que só atuam caso haja contacto da plataforma com um objeto ou pessoa, sem ser precisa a intervenção do utilizador. No que diz respeito ao manipulador, apenas dispõe de um botão de segurança que, devido às mesmas limitações que o botão de segurança da plataforma, não é um dispositivo de segurança muito eficaz para o sistema móvel.

Como consequência deste facto, decidiu-se implementar sensores que desempenharão um papel de dispositivos de segurança ativa durante o movimento ou operação do sistema robótico integrado. Os sensores foram aplicados e integrados de forma a eliminar o máximo de riscos possível. A sua utilização possibilita também a exploração de outros horizontes além da segurança, como é o caso do mapeamento, planeamento de trajetória de navegação, localização de precisão, entre outros.

6.2 Sensores para a unidade de segurança

Dos sensores possíveis, os mais usuais são à base de ultrassom ou infravermelho. Como estes últimos estavam disponíveis e apresentam uma boa resposta e fiabilidade, foram os escolhidos. Os sensores instalados foram os seguintes:

1. Hokuyo URG-04LX-UG01

   O primeiro laser a ser instalado está representado na figura 6.1; trata-se de um laser infravermelho, alimentado a 5VDC através da sua interface USB, com um alcance que vai dos 0.02 aos 4 metros. Varre no máximo 240° em sua volta com uma resolução de 0.36°, o que impõe que tenha um leitura de 683 pontos por varrimento. O LRF permite uma precisão de ±30mm e o conjunto das suas características definem que a sua classe de segurança é a número 1 (IEC60825-1). Importa ainda referir que o seu motor roda a uma velocidade de 600 rpm o que significa que cada *scan* demora 100ms a estar concluído e portanto, consiga 10 varrimentos por segundo.

Figura 6.1: Hokuyo URG-04LX-UG01

2. Hokuyo UTM-30LX

   O segundo laser a ser instalado ficará responsável por cobrir, na maior parte do tempo, a área para onde a plataforma está a navegar. Por outras palavras, ficará virado para o lado da frente da plataforma e, por esse motivo, decidiu-se que deveria ser o sensor representado na imagem 6.2 por se tratar do laser com maior qualidade.

   Com um campo de visão de 270° e uma resolução angular de 0.25°, fornecem uma leitura de 1080 pontos por *scan*. O seu alcance pode ir até aos 30 metros, sem comprometer a sua precisão de ±30mm. Devido à elevada velocidade de rotação do motor (2400 rpm), consegue uma leitura de 40 *scans* por segundo, 4 vezes mais que o primeiro laser. Tal como o LRF anterior, pertence à classe 1 de segurança; no entanto, necessita de ser alimentado externamente a 12 VDC, traduzindo-se na sua principal desvantagem em relação ao primeiro.

A disposição dos lasers no sistema robótico pode ser observada na figura 6.3. Como se pode conferir na figura 6.4, a sua localização estratégica permite um varrimento de 360° em volta da plataforma, permitindo assim a detecção de objetos, no plano de varrimento,

Figura 6.2: Hokuyo UTM-30LX

em todas as direções por onde eles possam surgir. O espaço de trabalho do manipulador também foi levado em conta e, estando os sensores naquela posição, pensa-se que a sua interferência com os movimentos do robot é mínima. No entanto, com o intuito de evitar possíveis acidentes, foi adicionado um modelo aproximado do Hokuyo UTM-30LX à cena de planeamento de trajetória. Assim, o sensor é contemplado no modelo URDF do sistema. Este é utilizado antes da realização de qualquer movimento com o objetivo de prever e evitar colisões.

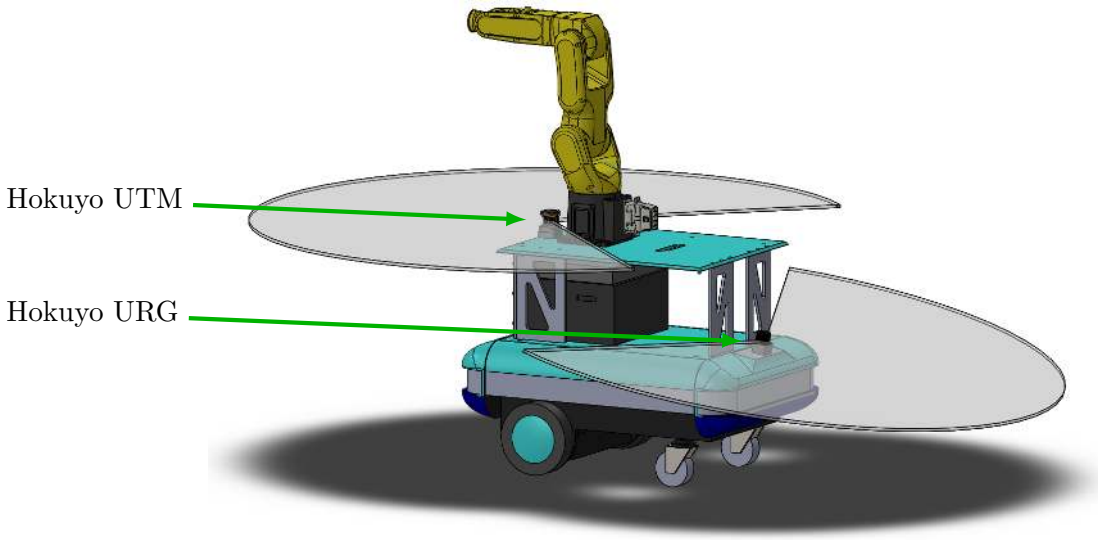


Figura 6.3: Disposição dos sensores no sistema robótico

6.3 Instalação dos sensores

Para recolher dados a partir dos lasers e integrá-los no ambiente ROS, são necessárias algumas configurações e montagem de um *setup* algo complexo. O primeiro passo foi a procura de um nodo responsável por adquirir os dados dos lasers Hokuyo e publica-los

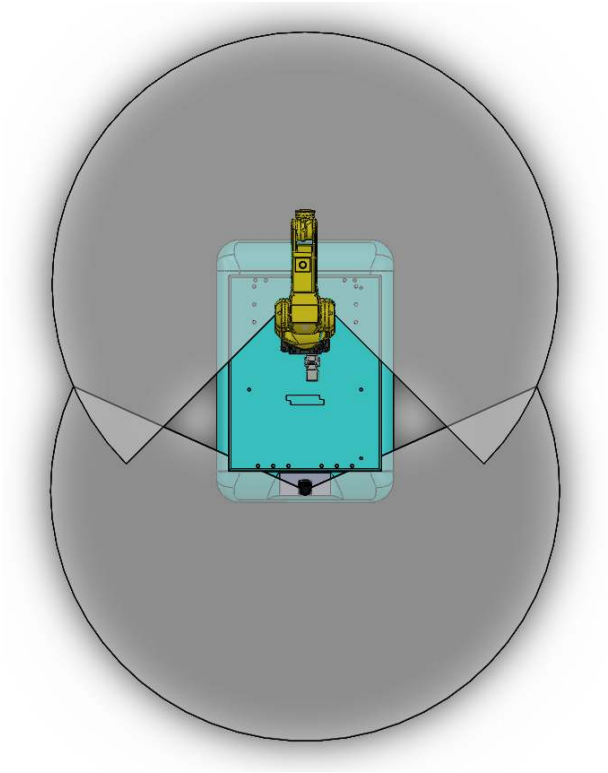


Figura 6.4: Zona de varrimento dos sensores

num tópico acessível a outros nodos. Sem muitas dificuldades, foi encontrada uma *package* ROS já desenvolvida que, entre várias ferramentas, dispõe de um nodo (`hokuyo_node`) que satisfaz as necessidades expressas [41].

Como o objetivo da aplicação envolve a utilização de dois lasers, criou-se uma *launch file* que se encarrega de lançar dois nodos do tipo `hokuyo_node` cada um deles responsável pela recolha de dados de um laser. Nesse ficheiro definem-se ainda alguns parâmetros que diferem entre os dois dispositivos, tais como o ângulo máximo e mínimo do campo de visão dos laser e a porta associada a cada um dos componentes.

O tratamento dos dados em separado é conseguido através de um remapeamento do tópico `/scan` que é publicado por ambos os nodos para os tópicos `/scan0` e `/scan1` responsáveis pelos lasers URG e UTM, respetivamente. Além disso, para a visualização dos dados no ambiente RViz, é inevitável a alteração do nome do parâmetro `frame_id`, que está associado ao nome do referencial a partir da qual os dados dos lasers são visualmente projetados, para os nomes `laser0` e `laser1`. Com o propósito de simplificar o *setup* imprescindível ao correto funcionamento, as referidas configurações foram acrescentadas na *launch file* evitando a execução de vários comandos na *shell*.

Para uma conveniente visualização de dados na *interface* RViz, resta ainda a gestão de transformações entre o referencial da base do FANUC e o referencial de cada laser. Já existindo um nodo (`vs_frame`) dedicado a essa gestão, acrescentou-se essa responsabilidade com a publicação de duas TF's que expressam a transformação geométrica do referencial `base_link` para os referenciais `laser0` e `laser1`. A figura 6.5 representa todos os sistemas de coordenadas geridos pelo nodo `vs_frame`.

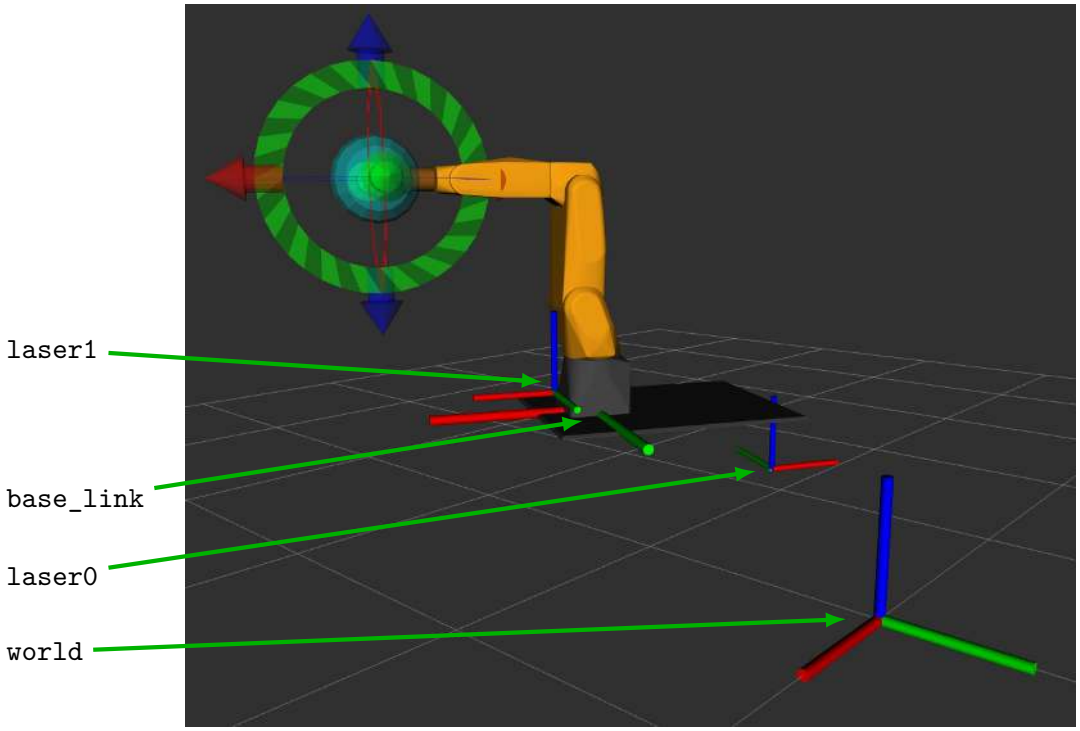


Figura 6.5: Sistemas de coordenadas geridos pelo nodo `vs_frame`

Desta forma, torna-se finalmente possível a visualização dos dados dos lasers no ambiente RViz juntamente com o manipulador já representado anteriormente, basta apenas configurar a interface de forma a acrescentar dois *LaserScan's* aos elementos de visualização associados aos tópicos `/scan0` e `/scan1`.

6.4 Integração dos LRF's na arquitetura de *software*

O ponto de partida para a interligação dos dados dos lasers à estrutura principal de *software* são os dois tópicos referidos na secção 6.3. Para dar continuidade a esta fusão, criou-se mais um nodo ROS a que se deu o nome `vs_laser_receptor`. O nodo mencionado está constantemente a receber vetores de dados com distâncias, em coordenadas polares, provenientes dos dois sensores.

Para cada recepção de dados, através dos tópicos `/scan0` ou `/scan1`, o nodo analisa o referido vetor no sentido de identificar a presença de obstáculos a uma distância inferior às de segurança, configuradas pelo utilizador. As distâncias de segurança são parametrizáveis no ficheiro *default.yaml*. Nele são definidos, individualmente para cada LRF, dois valores distanciais que se podem associar a dois perímetros de segurança em volta dos sensores (figura 6.7).

Durante a análise dos dados, dependendo da proximidade dos pontos mais próximos do sensor, é possível detetar situações de alarme e definir a sua severidade. Em todo o caso, verifica-se uma das seguintes situações:

1. Severidade nula (Alarme OFF) - Os pontos mais próximos do sensor encontram-se fora do maior perímetro de segurança (figura 6.8a).

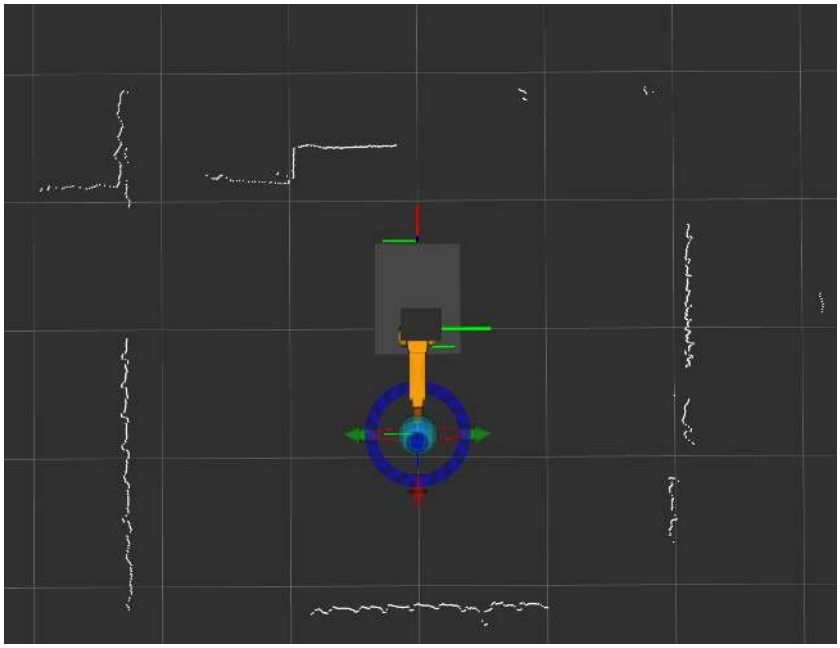


Figura 6.6: Representação dos dados dos lasers no ambiente RViz

2. Severidade moderada (Alarme ON) - Os pontos mais próximos do sensor encontram-se fora do menor perímetro de segurança mas dentro do maior (figura 6.8b).

3. Severidade elevada (Alarme ON) - Os pontos mais próximos do sensor encontram-se dentro do menor perímetro de segurança (figura 6.8c).

A presença de ruído foi levada em conta nesta fase de forma a evitar falsos alarmes. Como tal, se houver um *scan* isolado em condições de alarme ou um pequeno conjunto de pontos isolados dentro do perímetro de segurança, o alarme não é acionado.

Após esta análise de dados, é possível definir uma mensagem com informação relativa ao estado de alarme de cada laser (se está em alarme ou não) e o nível de severidade desse alarme. Assim sendo, foi criada uma mensagem semelhante que é publicada pelo nodo `vs_laser_receptor` caso se altere algum dos parâmetros de alarme. O tópico `/alarm` é o meio de circulação da mensagem e será subscrito pelo nodo central de navegação (`vs_platform_sim`). A situação de alarme, ao ser atualizada em tempo real no nodo de simulação de controlo da plataforma, permite que o seu controlo atue de acordo com e situação de alarme permitindo, por exemplo, a interrupção da navegação do sistema robótico caso algum objeto seja detetado no plano de ação dos lasers e dentro do perímetro de segurança.

6.5 Extensão da segurança a operações de manipulação

A segurança do sistema robótico não se resume à segurança durante a navegação. As operações de manipulação são tarefas que envolvem algum perigo e que, por isso, carecem da utilização de dispositivos de segurança. Relembrando a secção 4.1, o nodo `vs_platform_sim` é, entre outras tarefas, responsável por enviar comandos ao nodo

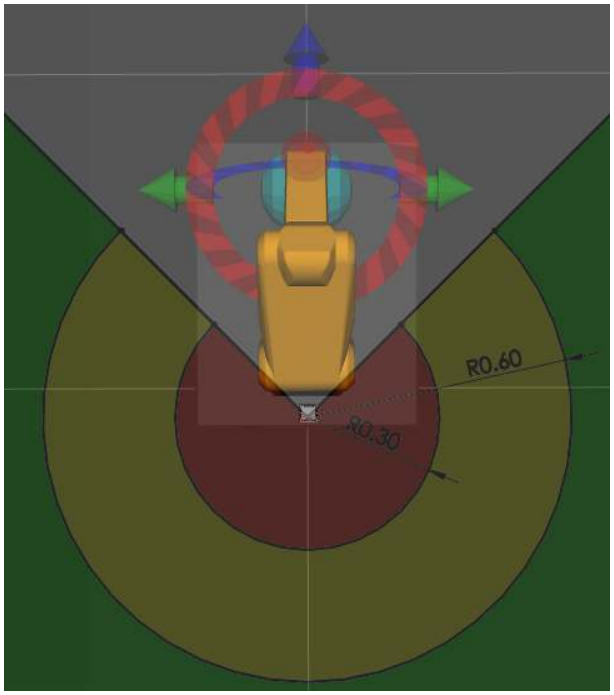
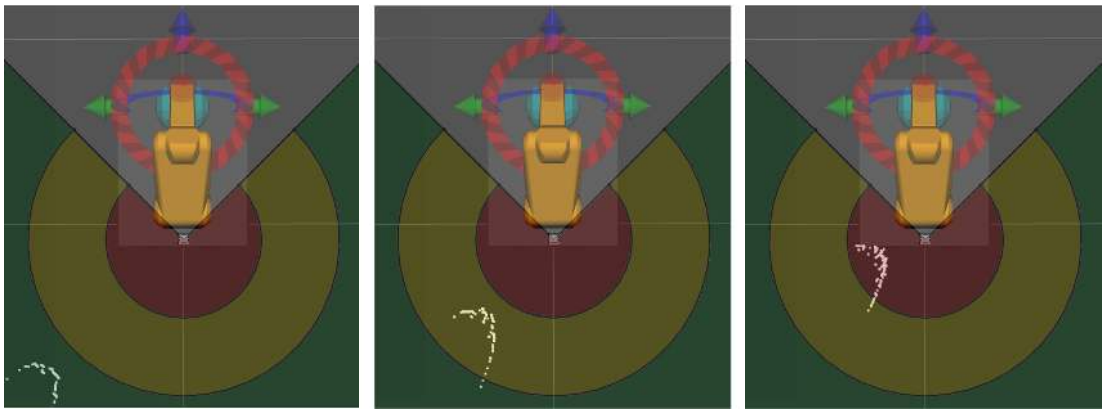


Figura 6.7: Correspondência das distâncias parametrizáveis aos perímetros de segurança



(a) Severidade nula (b) Severidade moderada (c) Severidade elevada

Figura 6.8: Correspondência dos dados dos sensores aos diferentes níveis de alarme

`vs_fanuc_client` que estão associados à execução de uma determinada tarefa ou conjunto de tarefas. No último caso, o controlo do manipulador está unicamente associado ao nodo cliente do manipulador, estando o nodo de controlo da plataforma unicamente à espera de receber um serviço que confirme que o manipulador acabou a execução de todas as tarefas correspondentes ao comando que enviou. Este facto requer que o estado de segurança seja também possível de aceder a partir do nodo de controlo do FANUC mas, contrariamente aos restantes fluxos de informação entre nodos, o momento de alteração de estado não é relevante.

Assim, foram utilizados parâmetros ROS para informar o nodo `vs_fanuc_client`

do estado de segurança imposto pelos lasers. Atualizados em tempo real pelo nodo `vs_platform_sim`, os parâmetros expressam o nível de gravidade do alarme de cada laser. Antes do manipulador executar qualquer movimento é verificado o estado máximo de alarme e dependendo desse valor, pode acontecer uma das seguintes situações:

1. Severidade nula

   O manipulador desempenha as tarefas que lhe são pedidas à velocidade máxima, configurável.

2. Severidade moderada

   O braço robótico exerce as suas funções a uma velocidade reduzida. Como a alteração de velocidade tem que ser imposta no programa TP de efetuação de movimentos do ROS FANUC, foi usado o controlo de I/O's para atuar uma entrada do controlador (DI109). Desta forma torna-se possível a verificação deste estado específico de segurança e em função disso, mover o robot com diferentes velocidades.

3. Severidade elevada

   Neste caso a ordem de movimento não é enviada para o controlador do robô, ficando o programa num ciclo contínuo de verificação dos parâmetros de segurança, à espera do desembaraço que disputou aquele estado de alarme. Aquando esse acontecimento, o manipulador continua a execução do seu trabalho a partir do ponto onde estava.

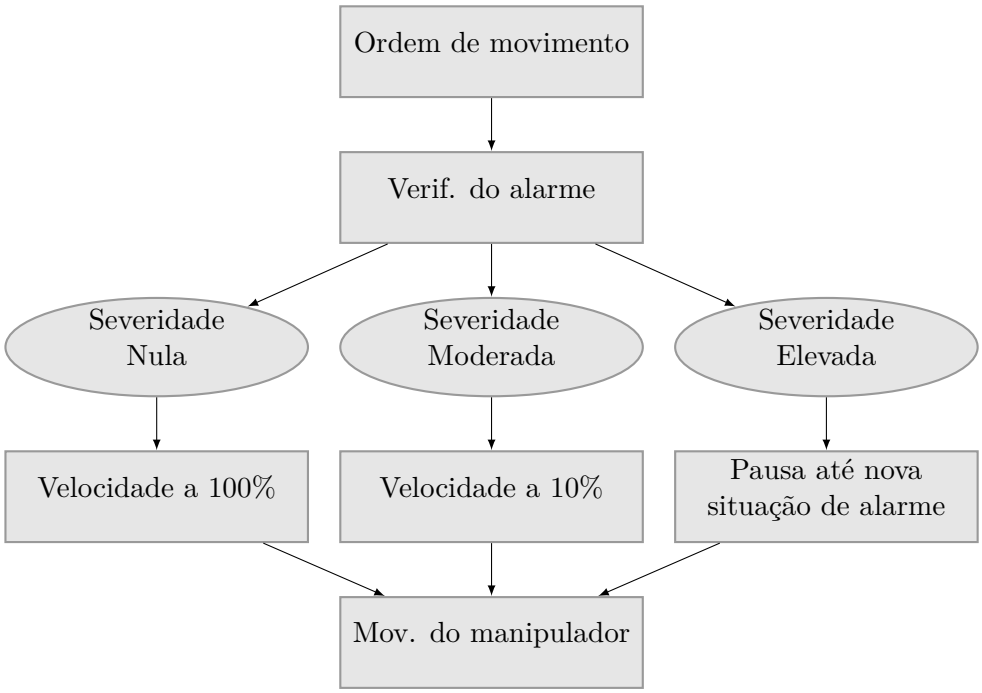


Figura 6.9: Ações em função da severidade de alarme

Toda a arquitetura envolvente à segurança do sistema robótico integrado encontra-se ilustrada na figura 7.7. Desta forma, fica também completa a arquitetura global de *software* responsável pelo controlo e monitorização do protótipo (figura 6.11).

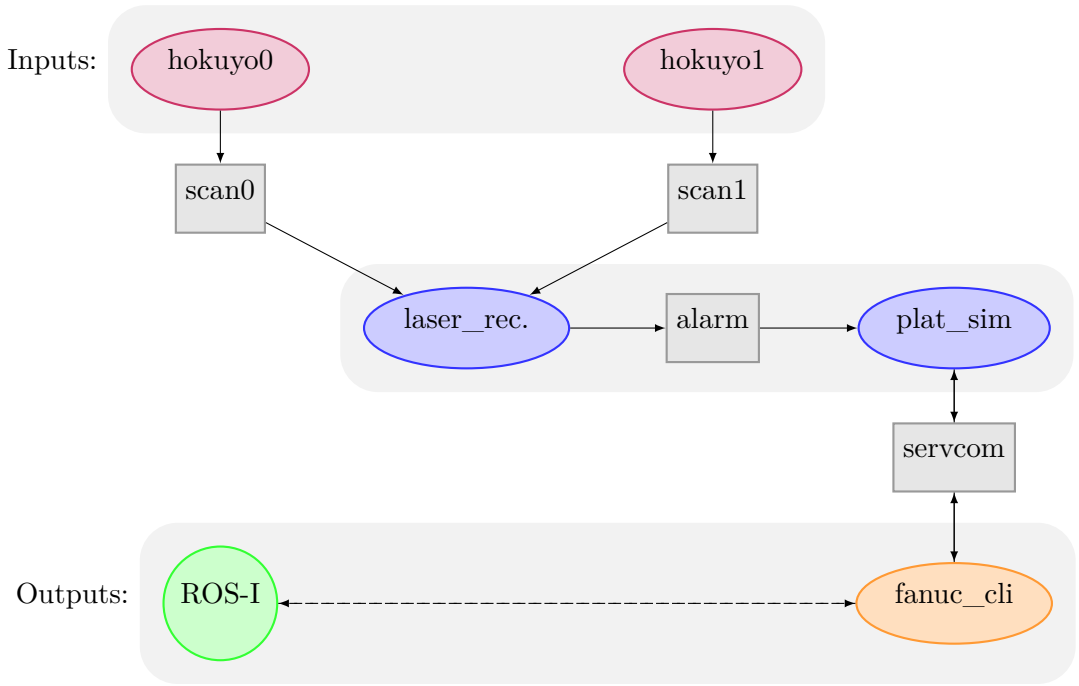


Figura 6.10: Infraestrutura de *software* responsável pela segurança do sistema
Elementos ovais: Nodos
Elementos retangulares: Tópicos/Serviços
Elemento circular: Arquitetura ROS Industrial

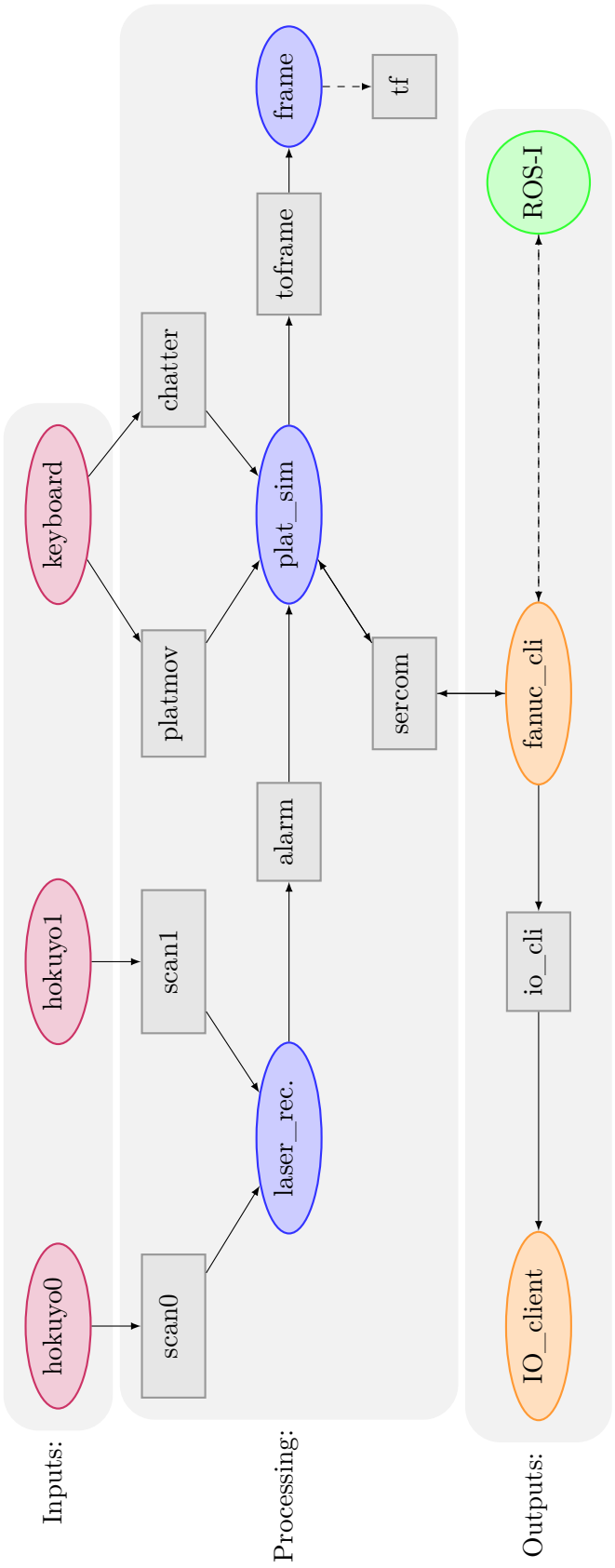


Figura 6.11: Integração da arquitetura de segurança na arquitetura global de *software*
Elementos ovais: Nodos
Elementos retangulares: Tópicos/Serviços
Elemento circular: Arquitetura ROS Industrial

Capítulo 7

Aplicações de Demonstração e Estudo

Neste capítulo serão apresentadas as primeiras aplicações de demonstração e estudo com o sistema robótico integrado, no sentido de testar as suas funcionalidades e de demostrar o seu potencial de aplicações.

7.1 Demonstração 1 (Demo 1)

A demo 1 é uma aplicação de teste do funcionamento do sistema robótico integrado. O foco principal da aplicação é o teste de todas as funcionalidades isoladas do *software*, nomeadamente a interface com o utilizador através do teclado, o controlo de posição da plataforma através de comandos específicos, todas as funções relativas ao *Toolkit FANUC* e o controlo de I/O's do robô. Importa referir que, na fase da sua realização, a segurança associada à instalação dos LRF's ainda não tinha sido introduzida; desta forma, a segurança não fará parte da panóplia de testes durante a demo 1. De seguida é detalhada toda a demo 1.

1. A aplicação começa com o movimento do manipulador, no espaço de juntas, para a posição de navegação (figura 7.1), uma das posições *default* já criadas para esse efeito;

2. Seguidamente, é realizada a navegação virtual e manual do sistema (figura 7.2) até uma posição aleatória do referencial global (`world`);

3. A inserção de um comando através do teclado informa que a plataforma chegou à posição final e que as operações de manipulação podem agora ser realizadas. A mudança de posição do manipulador para a posição *zero-hardware* modificada, observável na figura 7.3, permite verificar essa situação;

4. É efetuado um movimento, no espaço de juntas, para a posição *default* de aproximação à superfície de trabalho do sistema (figura 7.4a);

5. Um novo movimento, neste caso no espaço cartesiano, posiciona o *gripper* do manipulador na posição de aproximação à recolha do objeto (figura 7.4b);

Figura 7.1: Posição do braço robótico durante a navegação do sistema

6. Finalmente, é alcançada a posição de recolha ilustrada na figura 7.4c; é enviado um comando para unidade de controlo de I/O's que alterará o seu estado de forma ao controlador receber um sinal elétrico que atuará o *gripper*;

7. Um movimento no espaço cartesiano é responsável por retirar o objeto do seu *slot* (figura 7.4d);

8. Por último, um movimento no espaço de juntas transportará o objeto para a posição frontal ao sistema robótico ilustrada na figura 7.5. É enviado um novo sinal para a unidade de controlo de I/O's que se encarregará de desligar o *gripper*, depositando o objeto na última posição;

9. São repetidos os pontos 4 a 8 para mais dois objetos;

10. O movimento do manipulador de retorno para a posição de navegação, representada na figura 7.6, indica que a operação de manipulação foi concluída.

Todo o controlo associado à realização da aplicação está a cargo do computador central do sistema robótico integrado. A interface desse computador com o utilizador é conseguida através de um PC remoto em comunicação com o primeiro através do protocolo de rede criptográfico *Secure Shell* (SSH) [42]. Apesar de toda a utilidade do protocolo, este revelou-se pouco eficaz na interface com o manipulador através da interface RViz. Os *delays* consideráveis no controlo e monitorização do manipulador foram uma limitação permanente da demonstração 1.

Durante a realização da primeira demonstração, foi evidente a otimização das trajetórias. Em movimentos no espaço cartesiano, o manipulador passou a alcançar a posição pretendida na mesma redundância que a definida para o ponto de aproximação. Desta forma a trajetória entre pontos era a mais simples verificando-se, muito raramente - apenas quando o erro de posição associado ao ponto de aproximação era mais significativo - uma trajetória que envolvia manobras mais complexas por parte do manipulador. Da mesma forma, a fluidez do movimento entre os vários pontos da aplicação de demonstração também foi evidente.

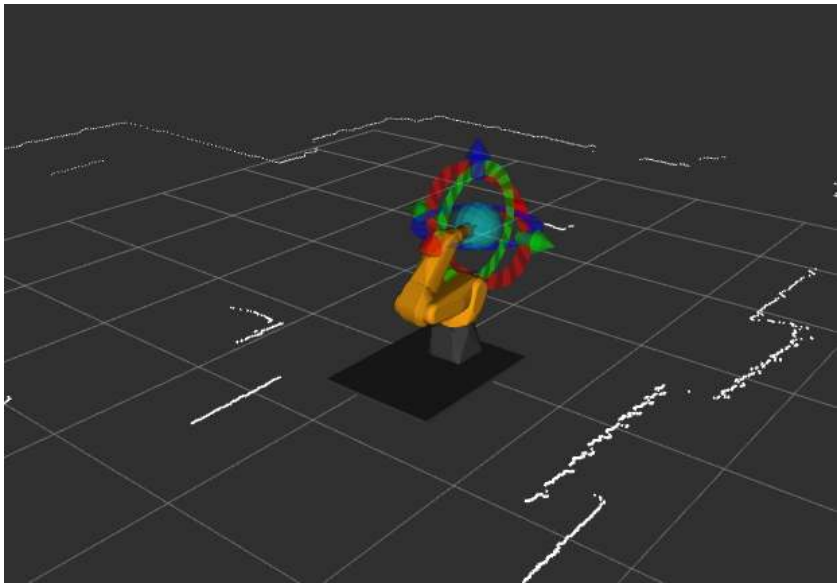


Figura 7.2: Navegação virtual do sistema

No decorrer da aplicação percebeu-se, claramente, que os dispositivos de segurança existentes no sistema robótico não eram suficientes para o tipo de robô e para as suas aplicações. O facto do manipulador industrial não colaborativo se poder movimentar a qualquer momento, com a possibilidade de haver humanos nas proximidades constitui um perigo grave e eminente que deveria ser reduzido.

Numa célula robotizada onde manipuladores industriais possam interagir com humanos, são requeridos dispositivos de segurança ativos como o caso das barreiras físicas ou ópticas, tapetes de segurança, etc. Na impossibilidade da instalação destes dispositivos de segurança, optou-se pela utilização de LRF's e pela integração dos seus dados na arquitetura de *software* para segurança (capítulo 6).

7.2 Demonstração 2 (Demo 2)

A demo 2 é uma aplicação onde se testará toda a arquitetura global, já incluindo o *software* relativo à segurança. De forma a perceber os avanços aplicativos que este módulo acrescenta ao sistema, manteve-se a aplicação de demonstração da demo 1 introduzindo, nesta fase, a navegação real do sistema (figura 7.1). As diferenças para a demo 1 passam também pelo controlo remoto do PC a bordo do sistema. Enquanto que na demo 1 se utilizou a execução remota de comandos através de um PC remoto em comunicação com o PC de controlo por intermédio do protocolo SSH, na demo 2 utilizou-se ROS distribuído [43].

Os resultados obtidos evidenciam as melhorias ao nível da segurança. As operações de manipulação - que na demo 1 se tinham mostrado perigosas para as pessoas que eventualmente se encontrassem junto ao sistema - tornaram-se perfeitamente compatíveis com a presença humana. A integração de dois níveis de severidade no alarme de segurança também resultou positivamente, permitindo a interação do manipulador com humanos sem comprometer a sua segurança.

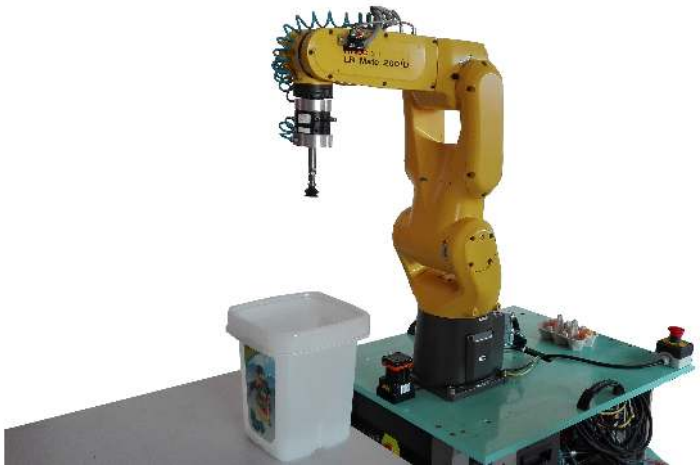


Figura 7.3: Posição *zero-hardware* do manipulador com a quinta junta a 90 graus

	Demonstração 1	**Demonstração 2**
Controlo remoto	Protocolo SSH	Protocolo SSH e ROS distribuído
Navegação	Virtual	Real
Dispositivos de segurança ativa	Inexistentes	Sensores laser 2D

Tabela 7.1: Principais diferenças entre a demonstração 1 e 2

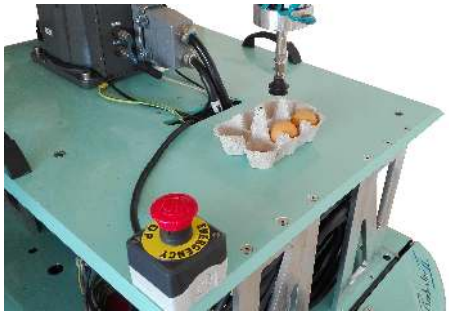
A utilização do ROS distribuído também trouxe vantagens ao nível da interface com o utilizador. Os longos *delays* associados à utilização do primeiro método foram eliminados, tornando a visualização do estado do manipulador e da posição do sistema no referencial global fluida e em tempo real, diminuindo também o tempo de resposta no envio de ordens para o PC principal.

O método da execução remota de comandos, apesar de não ser tão eficaz, continuou a ser fundamental. O facto da *launch file* - responsável pela execução dos nodos de leitura de dados dos LRF - necessitar do nome das portas USB do computador de bordo que lhes estão associadas, requer que o ficheiro seja lançado a partir do computador de bordo. Desta forma, o referido método torna-se uma ferramenta indispensável.

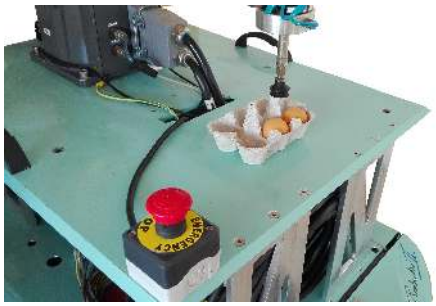
Notou-se que, devido à falta de percepção, o número de aplicações possíveis de concretizar com o sistema é reduzido. Assim, o protótipo carece de sensores de perceção que deverão ser instalados em trabalhos futuros. A autonomia do sistema também é outra prioridade de desenvolvimento, dado que a necessidade de utilizar cabos e mangueiras de ar ligadas ao protótipo não é, de todo, prática.

7.3 Parametrização da unidade de segurança

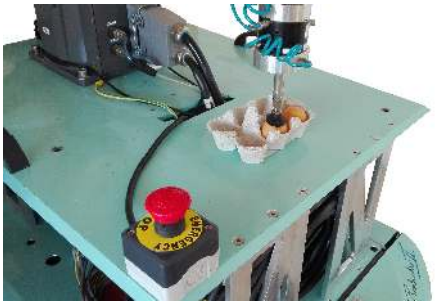
Durante a realização da Demo 2 apercebeu-se que, por vezes, durante operações de manipulação, o manipulador poderia interferir com o plano de varrimento dos sensor instalado na zona frontal do sistema robótico. Se esse caso se verificasse e se a distância ao sensor fosse inferior à definida pelo utilizador, seria despoletado um alarme de severidade elevada que resultaria na interrupção do movimento do manipulador. Nesta situação, o



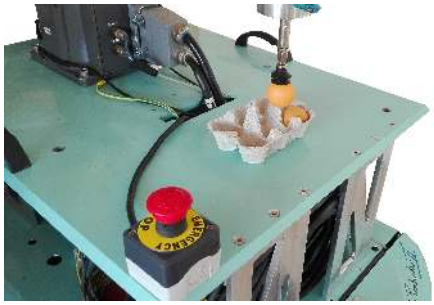
(a) Posição *default*



(b) Posição de aproximação



(c) Posição de recolha



(d) Posição de afastamento

Figura 7.4: Posições percorridas pelo manipulador para recolha de um objeto localizado na superfície de trabalho do Robonuc

manipulador ficaria preso nesta posição, pois o seu movimento era impossibilitado pela verificação do estado de alarme.

De forma a evitar essa situação adicionou-se à arquitetura de *software* desenvolvida um conjunto de parâmetros que gerem a situação de alarme.

1. `robonuc_safety`

   Este é o parâmetro geral de segurança, uma vez que o seu valor seja "0"toda a segurança associada aos sensores laser é ignorada.

2. `safetyl0`

   Parâmetro de limitação de segurança. Se for ajustado a "0"a segurança associada ao LRF Hokuyo URG (instalado atrás do sistema) é ignorada.

3. `safetyl1`

   Parâmetro de limitação de segurança. Se for ajustado a "0"a segurança associada ao LRF Hokuyo UTM é ignorada. Note-se que este é o sensor instalado na frente do sistema e que pode detetar o manipulador no seu campo de visão.

Os parâmetros enumerados são criados durante a execução da *launch file* com os seus valores padrão todos a "1". O utilizador poderá alterar o valor dos parâmetros, parametrizando o comando de execução da *launch file* (`roslaunch robonuc robotic_system.launch safety:=true laserb:=true laserf:=false`) ou poderá ser o próprio *software*, nomeadamente o nodo `vs_fanuc_client`, a encarregar-se da sua

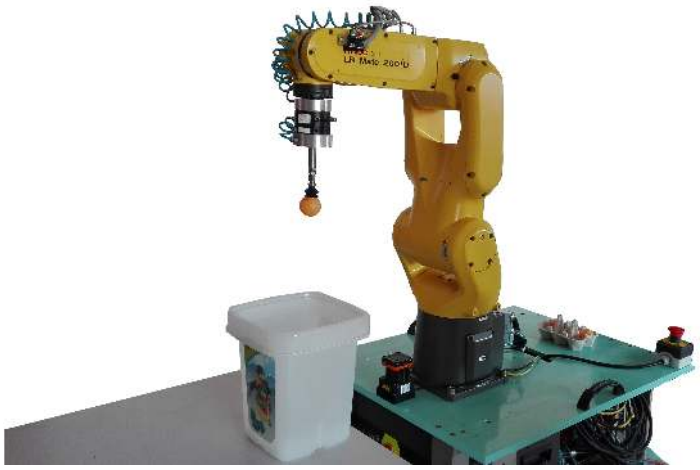


Figura 7.5: Posição do robô aquando a deposição do objeto

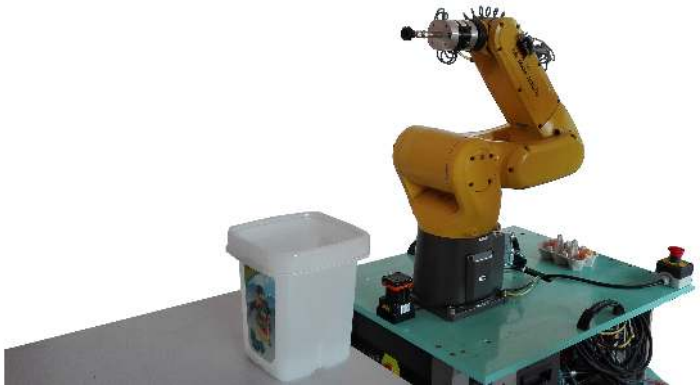


Figura 7.6: Posição de navegação final assinalando o fim de operações de manipulação

alteração se houver a previsão que as movimentações do manipulador entrem no campo de visão dos sensores.

Da mesma forma, se os dados dos sensores estiverem a ser usados para segurança da navegação e se o sistema necessitar de se aproximar de um obstáculo, os parâmetros poderão ser alterados no sentido de permitir que o sistema cumpra com as tarefas propostas pelo utilizador.

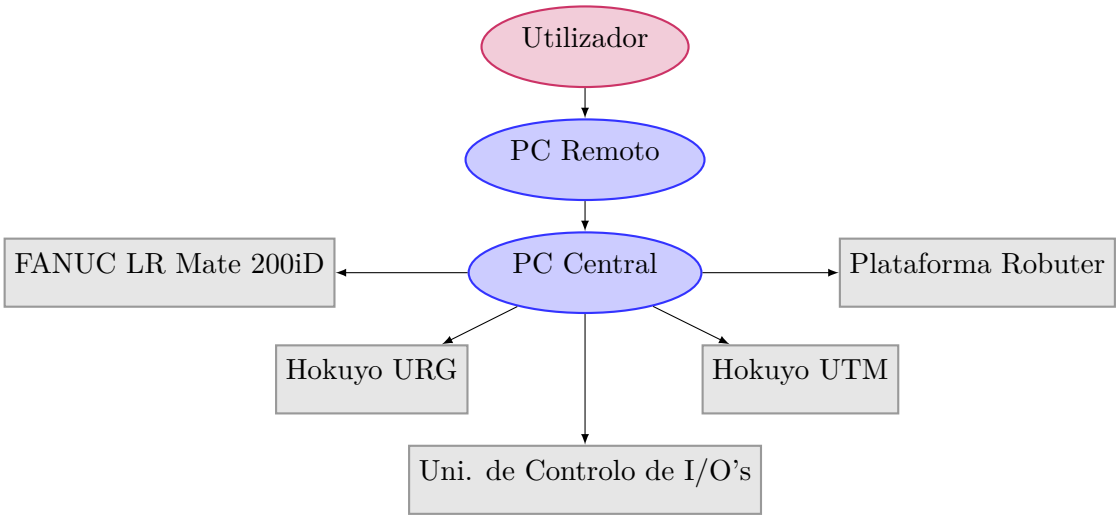


Figura 7.7: Arquitetura do sistema de controlo do sistema robótico integrado Robonuc

Capítulo 8

Conclusões e Trabalho Futuro

8.1 Conclusões

No decorrer deste projeto fez-se uma intervenção mecânica no sentido de integrar o manipulador FANUC LR Mate 200iD na Plataforma Robuter para manipulação móvel. As simulações efetuadas foram um importante indicador da robustez do sistema, validando a resistência mecânica da estrutura à cedência. No que diz respeito à instalação elétrica, deixou-se o sistema preparado para a instalação de um inversor de corrente, de forma a assegurar a autonomia elétrica do protótipo.

Foram ainda efetuadas várias configurações ao nível dos modelos URDF do sistema, e ao nível do movimento do manipulador. Estas contribuíram para a segurança do sistema contra colisões e para a fiabilidade do planeamento de trajetórias e sua fluidez durante a execução.

Ao nível do *software*, foi desenvolvida uma arquitetura descentralizada onde é assegurada a comunicação entre todo o *hardware* até agora integrado no sistema, nomeadamente a plataforma, o manipulador, unidades de controlo e os LRF's. As ferramentas de *software* utilizadas e descritas corresponderam às expetativas, na medida em que permitiram a resolução de todos os problemas e a efetivação de todas as funcionalidades pretendidas de uma forma eficaz. No que diz respeito ao *software*, destaca-se também a interface com o utilizador através do teclado e todo o processamento associado à interação do sistema com o utilizador.

Foi ainda resolvida a limitação do controlo de I/O's, um problema que já persistia de trabalhos anteriores e que, com a introdução de uma unidade de controlo adicional e alguma eletrónica, ficou resolvido. Com esta funcionalidade acrescentada, não só é possível o controlo do *gripper* do manipulador como também a sua troca a bordo da plataforma ou mesmo interagir com o manipulador através de I/O's.

Dado que o sistema robótico não dispunha de mecanismos de segurança ativa e, não obstante a segurança ser um dos principais aspetos a ter em conta no desenvolvimento de um protótipo, a introdução dos LRF's como dispositivos de segurança ativa foi uma contribuição válida no projeto. A interligação dos dados dos sensores à arquitetura principal, com vários níveis de severidade a corresponder a diferentes limitações de movimento, permite a interação segura de humanos com o sistema, aumentando consideravelmente o seu número de aplicações.

As aplicações de teste do sistema global contribuíram em larga escala para o desenvolvimento e melhoria do projeto. Durante esta fase surgiram alguns pontos fracos ao nível

do *software*; como tal, foram implementadas algumas soluções de forma a preencher essas limitações. A alteração do método de controlo remoto ou a parametrização da unidade de segurança são os melhores exemplos destas soluções.

Numa visão geral sobre o trabalho, pode dizer-se que os objetivos principais definidos foram todos cumpridos. Em alguns aspetos, como a segurança ou a otimização do planeamento e movimento do manipulador, pode mesmo dizer-se que foram ultrapassados, visto que foram realizadas algumas contribuições para sistema robótico que não estavam planeadas nos objetivos. Com a observação das aplicações de demostração e estudo, é comprovada a viabilidade do protótipo e o potencial das suas aplicações. Será portanto um projeto que continuará a ser desenvolvido no contexto dos trabalhos futuros.

8.2 Trabalho futuro

Tendo este projeto começado com a realização desta dissertação, existe ainda muito trabalho a ser desenvolvido em termos futuros. As primeiras intervenções passam pela autonomização do sistema e pela integração de sensores de perceção e segurança.

8.2.1 Autonomia

No que diz respeito à autonomia, este sistema ainda não se pode classificar de autónomo, dado que para operar usando todas as suas funcionalidades precisa de estar ligado a fontes externas. Para evitar tal situação, a solução passa pela implementação/alteração de dois componentes, nomeadamente:

1. Inversor de corrente

   Como a plataforma dispõe de 4 baterias com uma tensão total de 48 VDC - responsáveis pela alimentação da plataforma móvel - através da utilização de um inversor de corrente DC-AC consegue-se uma tensão de 230VDC. Desta forma, é possível assegurar a alimentação do manipulador FANUC descartando-se assim a necessidade de um cabo de alimentação ligado ao sistema.

2. Ar Comprimido a bordo/*gripper* elétrico

   Como se pode perceber neste trabalho, apesar de ser possível alteração do *gripper* do robô, neste momento é utilizada uma ferramenta que utiliza ar comprimido. Como tal, o recurso a uma fonte de ar comprimido externa é indispensável. É possível contornar o problema através da utilização de uma fonte de ar comprimido a bordo, como um compressor de reduzidas dimensões ou com a utilização de um tipo de *gripper* elétrico que não necessite de ar comprimido para o seu acionamento.

8.2.2 Perceção

Em operações de manipulação num ambiente não estruturado, sem o conhecimento preciso da posição do sistema no espaço (tal como acontece na atualidade onde é usada apenas a odometria para leitura da posição da plataforma), o uso da visão/perceção é essencial. Não obstante esse facto, e mesmo com o uso dos lasers de varrimento 2D, o protótipo não possui qualquer tipo de sensor que seja útil o que diz respeito à percepção do meio envolvente para manipulação. Deste modo, trabalhos futuros deverão ser realizados no sentido de preencher essa lacuna, algumas opções podem ser implementadas:

1. Câmara na ponta do manipulador

   Um dos sensores mais utilizados na área da visão é a câmara. Esta pode ser uma excelente forma de percepção dependendo dos objetos de trabalho com que se está a operar, do processamento da imagem e, claro, das suas caraterísticas técnicas. De facto, tem-se tornado numa ferramenta cada vez mais utilizada na industria como resposta às necessidades da indústria 4.0 e da produção autónoma. Com a sua implementação, este protótipo beneficiaria de inúmeras novas aplicações na parte da manipulação, pelo que se aconselha vivamente a sua aplicação em trabalhos futuros.

2. Sensor Kinetic para Percepção/Segurança

   Uma outra forma de percepção que se poderia integrar no sistema é o sensor Kinetic. Ao contrário da câmara e dos lasers, permite uma leitura 3D do espaço através da projeção de uma malha de pontos. Da mesma forma que a câmara, acrescentaria um grande conjunto de aplicações ao sistema não só no que diz respeito à manipulação mas também no campo da segurança. Note-se que o único dispositivo de segurança ativa são os *lasers*, como tal a utilização deste sensor iria reforçar o campo da segurança ativa, um aspeto central da conceção de qualquer equipamento.

8.2.3 Tratamento mais eficaz dos dados provenientes dos LRF's

Apesar de se ter parametrizado os dados dos sensores de forma a manualmente evitar que o próprio manipulador acione o alarme de segurança, pode usar-se um método automático onde não seja preciso ignorar todos os dados do mesmo sensor. Sabendo a amplitude angular do sensor, a sua resolução angular, o número de pontos que o sensor retorna e a sua distância ao sensor, pode converter-se os seus dados para coordenadas polares. A partir de outra conversão, pode-se chegar aos mesmos valores em coordenadas cartesianas.

Através da subscrição do tópico `/joint_states` é possível saber, em tempo real, o valor das 6 juntas do manipulador que, com recurso à sua cinemática direta, poderá dar a posição da sua ponta em coordenadas cartesianas.

Ora, o objetivo deste trabalho é verificar automaticamente - caso se detete a oclusão do sensor - se essa oclusão foi provocada pela ponta do robô. Desta forma, poderá detetar-se a aproximação de humanos do sistema robótico ao mesmo tempo que operações de manipulação estejam a ocorrer no plano de varrimento do sensor. À partida, prevê-se a dificuldade de a oclusão ser provocada pelos elos do robô e não propriamente pela sua ponta pelo que, neste caso, a verificação será mais difícil de efetuar.

A dissertação de mestrado de Jorge Almeida [44] contém trabalho que poderá ser utilizado como ponto de partida para levar a bom porto esta tarefa.

Os trabalhos futuros neste campo passam ainda pela utilização dos dados dos sensores para navegação e/ou localização do sistema. Tal como as operações de manipulação, as de navegação são igualmente perigosas para pessoas que eventualmente estejam próximas do sistema ou para a própria segurança do equipamento. Assim, além da possível utilização dos parâmetros de segurança publicados pelo nodo `vs_platform_sim` para navegação, poderão ser criados outros algoritmos de navegação com base, por exemplo, no espaço livre. Os dados poderão ainda ser utilizados para a localização do sistema num referencial

global. A fusão e comparação de modelos CAD com os dados dos sensores poderá ser uma metodologia interessante na exploração deste campo.

Referências

[1] Oliver Brock. *A Historical Perspective*. Mar. de 2012. URL: http://www.mobilemanipulation.org/index.php/about (acedido em 20/01/2017).

[2] Lei Cui et al. «Challenges and Solutions for Autonomous Robotic Mobile Manipulation for Outdoor Sample Collection». en. Em: *Journal of Electrical and Electronic Engineering* 3.5 (dez. de 2015), p. 156. ISSN: 2329-1605.

[3] *KUKA to use youBot in manipulation competition*. URL: https://www.robots.com/articles/viewing/kuka-to-use-youbot-in-manipulation-competition (acedido em 23/01/2017).

[4] Tiago Simões. «Integração de ROS-Industrial num robô FANUC para flexibilizar atividades de cooperação». Tese de mestrado. Universidade de Aveiro, 2016.

[5] A. HENTOUT et al. «Mobile Manipulation: A Case Study». Em: (2012).

[6] *Robô industrial FANUC LRMate 200iD - Fanuc*. URL: http://www.fanuc.eu/uk/en/robots/robot-filter-page/lrmate-series/lrmate-200-id (acedido em 15/02/2017).

[7] FANUC CORPORATION. *FEATURES Application system Load / unload from ROBODRILL Operating space*. 2012.

[8] *Controlador de robô FANUC R-30iB - Fanuc*. URL: http://www.fanuc.eu/uk/en/robots/accessories/robot-controller-and-connectivity (acedido em 05/03/2017).

[9] Bruno Vieira. «Retrofitting of the Robuter Platform to an AGV with Visual Guidance». Tese de mestrado. Universidade de Aveiro, 2017.

[10] *MRPT - Empowering C++ development in robotics*. URL: http://www.mrpt.org (acedido em 12/01/2017).

[11] *RDS 4*. URL: https://www.microsoft.com/en-us (acedido em 12/01/2017).

[12] *CARMEN*. URL: http://carmen.sourceforge.net/home.html (acedido em 12/01/2017).

[13] L. Sanchez Crespo e A. Mahtani A. Martinez E. Fernandez. «Learning ROS for Robotics Programming». Em: vol. 1.2nd Ed. (2015).

[14] Morgan Quigley et al. «ROS: an open-source Robot Operating System». Em: *ICRA workshop on open source software*. Vol. 3. Kobe, 2009, p. 5.

[15] *ROS/Introduction - ROS Wiki*. URL: http://wiki.ros.org/ROS/Introduction (acedido em 12/01/2017).

[16] *Services - ROS Wiki*. URL: http://wiki.ros.org/Services (acedido em 18/02/2017).

[17] David Silva. «Multisensor Calibration and Data Fusion Using LIDAR and Vision». Tese de mestrado. Universidade de Aveiro, 2016.

[18] *tf - ROS Wiki*. URL: http://wiki.ros.org/tf#static_transform_publisher (acedido em 18/03/2017).

[19] *tf/Tutorials/Adding a frame (C++) - ROS Wiki*. URL: http://wiki.ros.org/tf/Tutorials/Adding (acedido em 18/03/2017).

[20] *Parameter Server - ROS Wiki*. URL: http://wiki.ros.org/Parameter%20Server (acedido em 29/04/2017).

[21] Lentin Joseph. *Mastering ROS for robotics programming*. Packt Publishing Ltd, 2015. (Acedido em 13/01/2017).

[22] *ROS-Industrial core meta-package (http://wiki.ros.org/industrial_core)*. Jun. de 2017. URL: https://github.com/ros-industrial/industrial_core (acedido em 17/01/2017).

[23] G. A. vd Hoorn. *ROS-Industrial Fanuc meta-package. http://ros.org/wiki/fanuc*. Abr. de 2015. URL: https://github.com/gavanderhoorn/fanuc (acedido em 20/01/2017).

[24] *fanuc/Tutorials/hydro/Installation - ROS Wiki*. URL: http://wiki.ros.org/fanuc/Tutorials/hydro/Installation (acedido em 01/02/2017).

[25] *FAQs | MoveIt!* URL: http://moveit.ros.org/documentation/faqs/ (acedido em 14/01/2017).

[26] Sachin Chitta, Ioan Sucan e Steve Cousins. «MoveIt! [ROS Topics]». Em: *IEEE Robotics & Automation Magazine* 19.1 (mar. de 2012). ISSN: 1070-9932. (Acedido em 04/02/2017).

[27] *Experimental packages for Fanuc manipulators within ROS-Industrial (http://wiki.ros.org/fanuc_experimental)*. Abr. de 2017. URL: https://github.com/ros-industrial/fanuc_experimental (acedido em 17/01/2017).

[28] *Concepts | MoveIt!* URL: http://moveit.ros.org/documentation/concepts/ (acedido em 14/01/2017).

[29] Dinco Trading Co. *Aluminium Alloy 5083 - H111 Sheet*. URL: http://www.dinco.ae/files/aluminium-alloy-5083-data-sheet.pdf.

[30] *urdf - ROS Wiki*. URL: http://wiki.ros.org/urdf (acedido em 15/02/2017).

[31] *Fanuc LR-Mate 200iD - SOLIDWORKS,STEP / IGES - 3D CAD model - GrabCAD*. URL: https://grabcad.com/library/fanuc-lr-mate-200id-1 (acedido em 02/03/2017).

[32] Adília Maria Lúcia Teixeira Gomes Marinho. «Os quaterniões e suas aplicações». Tese de doutoramento. 2013. (Acedido em 01/04/2017).

[33] Martin Baker. *Maths - Conversion Quaternion to Matrix*. URL: http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix (acedido em 01/04/2017).

[34] Martin Baker. *Maths - Conversion Matrix to Quaternion*. URL: http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion (acedido em 01/04/2017).

[35] Sourangsu Banerji. «Study and Development of a Data Acquisition & Control (DAQ) System using TCP/Modbus Protocol». Em: *arXiv preprint arXiv:1307.1751* (2013). URL: https://arxiv.org/abs/1307.1751 (acedido em 16/01/2017).

[36] Fred Dirla. *Modbus Application Protocol Specification V1.1b*. 2006.

[37] Vitor Silva. *Interface Gráfica com o manipulador Fanuc LRMate200iD através do protocolo ModBus*. Jun. de 2017. URL: https://github.com/VSilvaa/PARI_Project (acedido em 11/01/2017).

[38] G. A. vd Hoorn. *fanuc_ros_cgio: A simple CGI-like interface to the IO ports on Fanuc robot controllers*. Jun. de 2017. URL: https://github.com/gavanderhoorn/fanuc_ros_cgio (acedido em 14/04/2017).

[39] *Arduino LEONARDO ETH*. URL: http://www.arduino.org/products/boards/arduino-leonardo-eth (acedido em 05/04/2017).

[40] José Gomes Ferreira. *Segurança em automação industrial*. Fev. de 2017. (Acedido em 23/02/2017).

[41] *hokuyo_node: A ROS node to provide access to SCIP 2.0-compliant Hokuyo laser range finders*. Mar. de 2017. URL: https://github.com/ros-drivers/hokuyo_node (acedido em 06/05/2017).

[42] *SSH Protocol - Secure Remote Login and File Transfer*. URL: https://www.ssh.com/ssh/protocol (acedido em 02/06/2017).

[43] *ROS/Tutorials/MultipleMachines - ROS Wiki*. URL: http://wiki.ros.org/ROS/Tutorials/MultipleMachines (acedido em 06/06/2017).

[44] Jorge Almeida. «Target tracking using laser range finder with occlusion». Tese de mestrado. Universidade de Aveiro, 2010.

Apêndices

Apêndice A

Desenhos de definição das placas estruturais

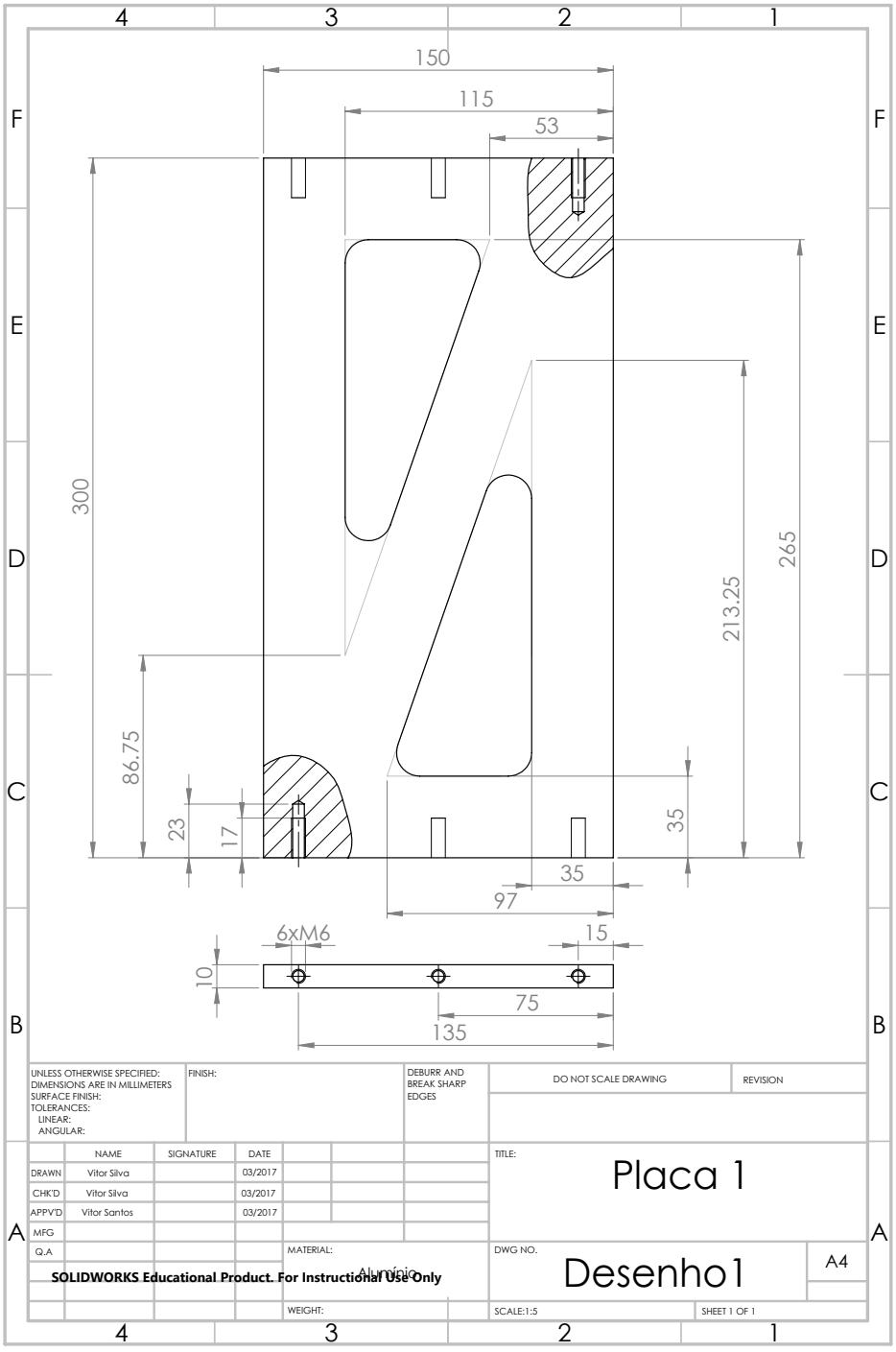


Figura A.1: Desenho de definição da placa nr. 1

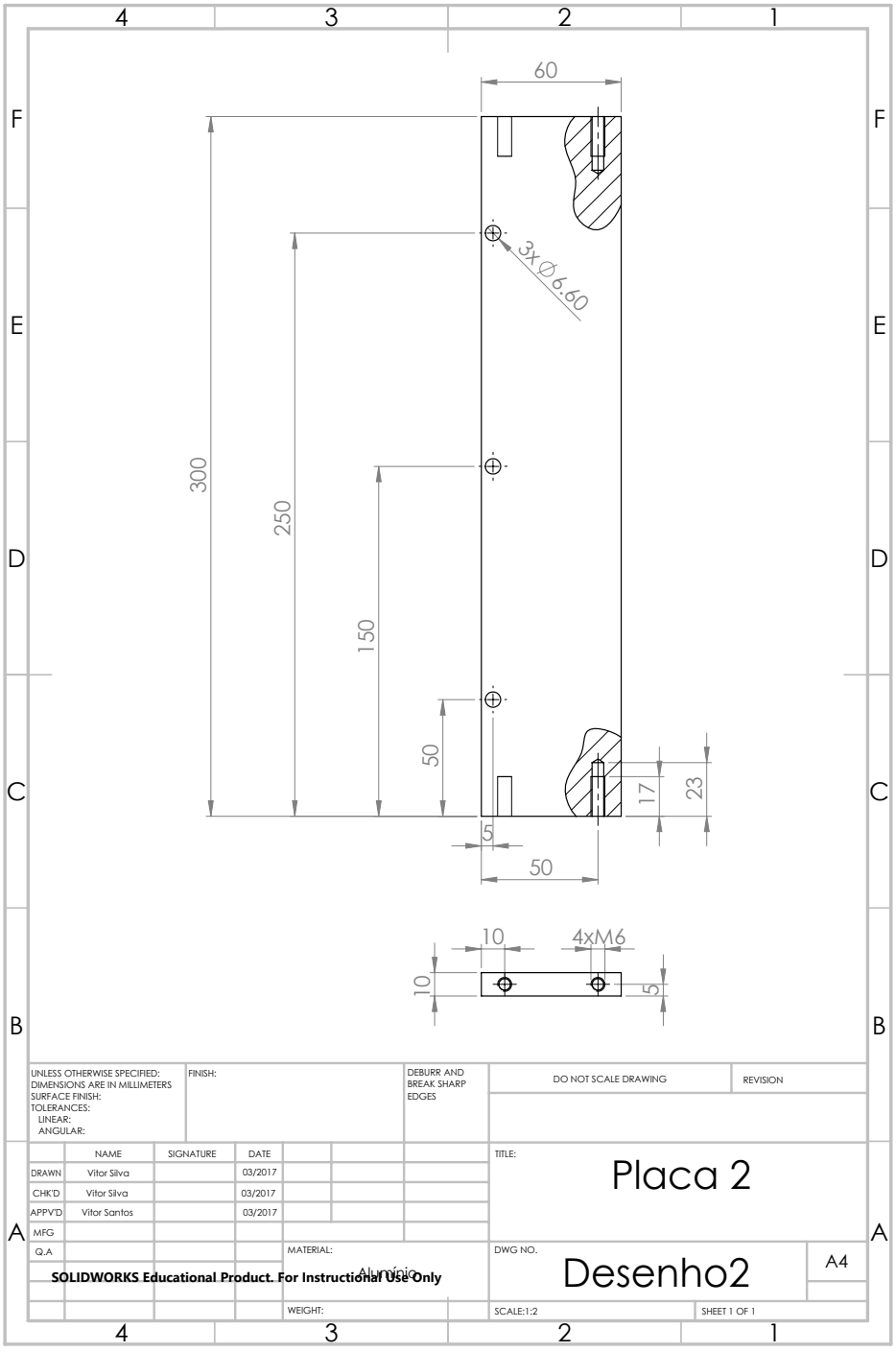


Figura A.2: Desenho de definição da placa nr. 2

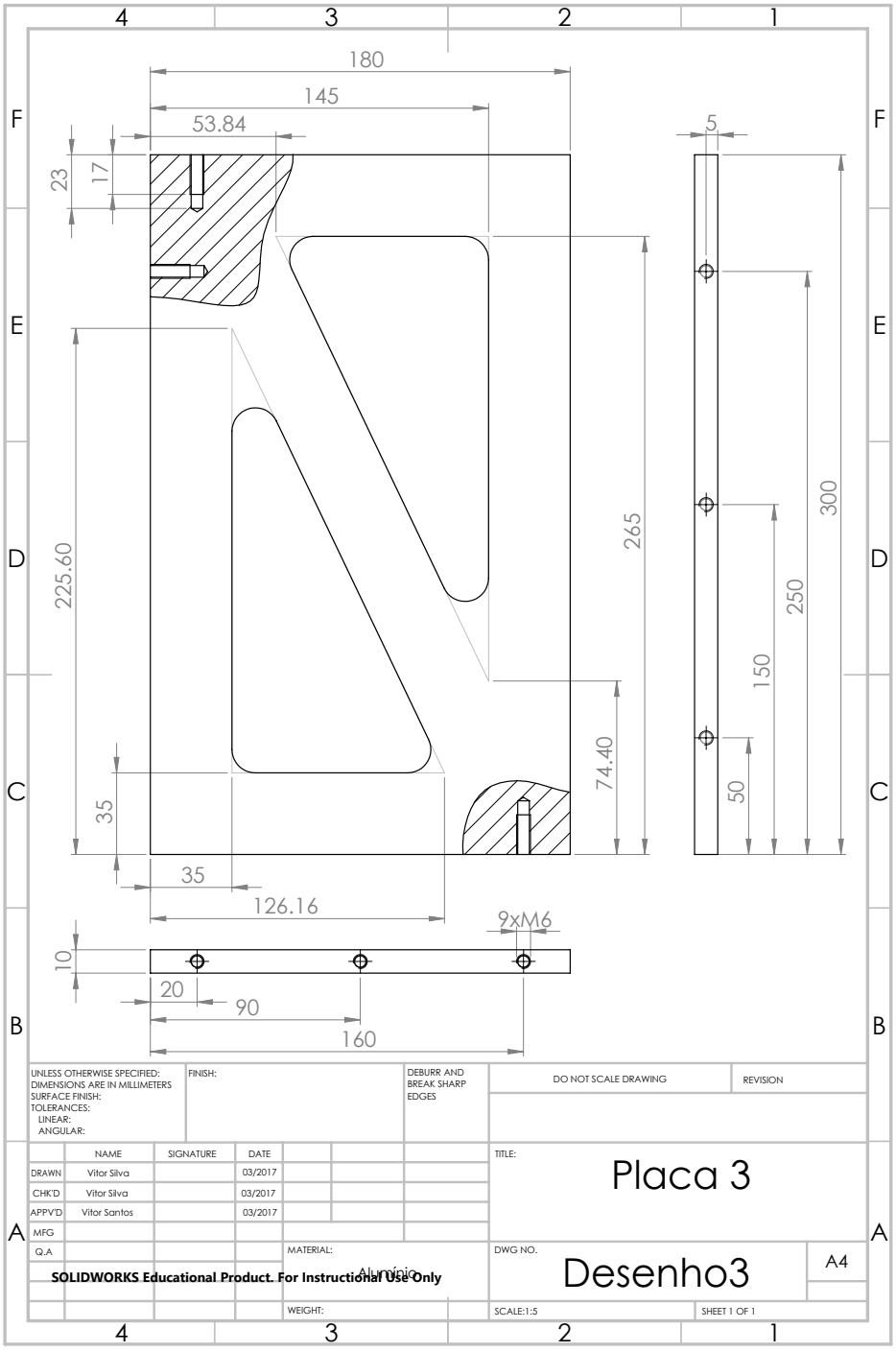


Figura A.3: Desenho de definição da placa nr. 3

Apêndice B

Circuito completo de controlo de I/O's

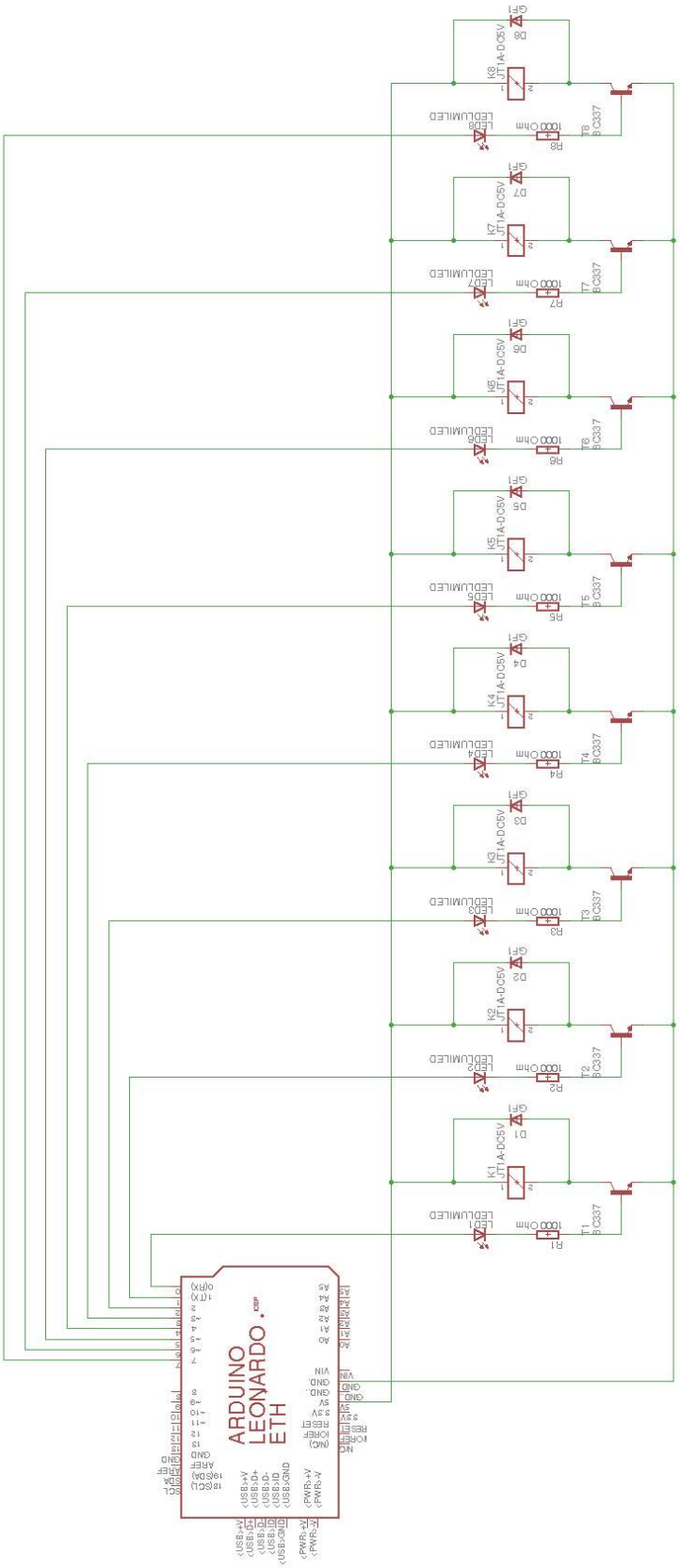


Figura B.1: Esquema elétrico da placa de relés

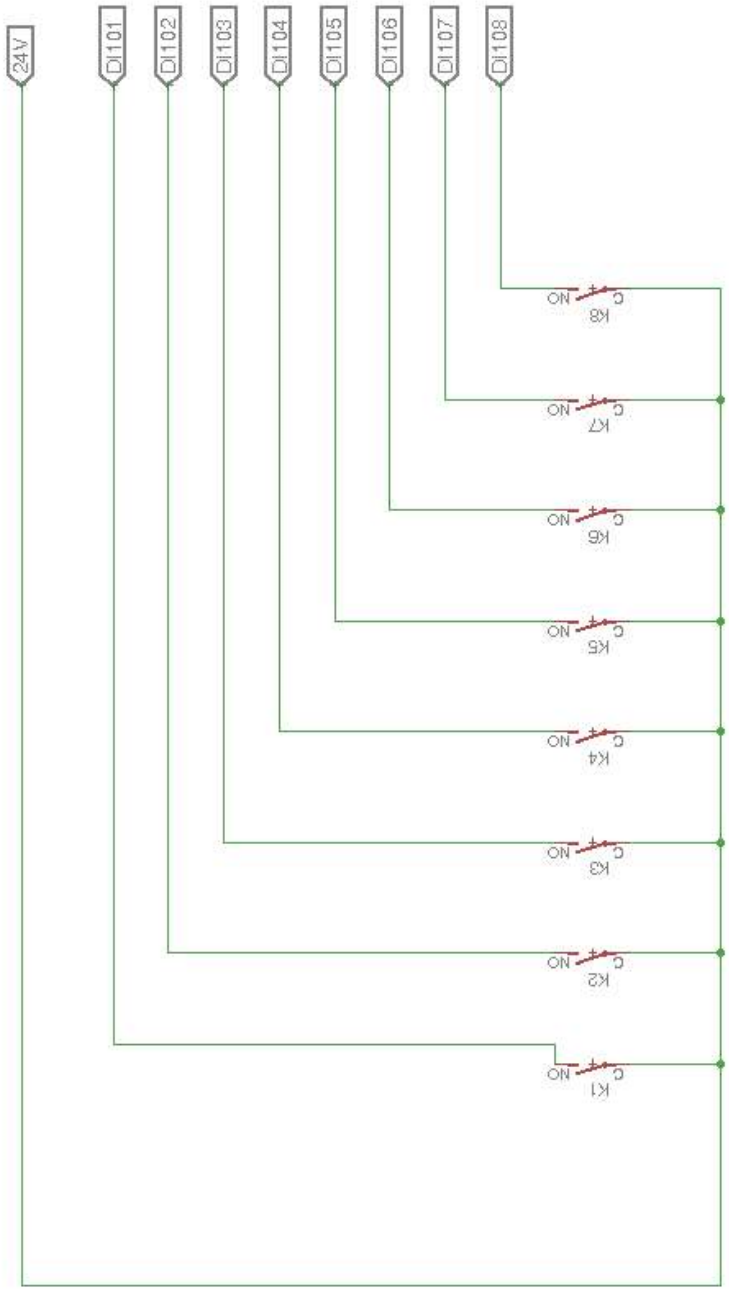


Figura B.2: Ligação dos relés ao manipulador

Apêndice C

Manuais de instalação e execução da arquitetura global da API

C.1 Instalação de *software*

Admitindo que o ROS já está previamente instalado na Unidade de controlo, os passos a seguir para a instalação de toda a arquitetura de *software* de *interface* com o sistema é a seguinte:

1. Ativar C++ 11 no compilador CMake através da inserção do do comando no ficheiro CMakeLists.txt no diretório /home/robuter/catkin_ws/src;
2. Instalar a *metapackage* MoveIt! inserindo dos comandos `$sudo apt-get install ros-kinetic-moveit; source /opt/ros/kinetic/setup.bash`;
3. Instalar as *packages* associadas ao ROS-I e suas dependências pela seguinte ordem: *driver_common*, *industrial_core*, *moveit_resources-master* e *moveit-kinetic-devel*;
4. Instalar as *packages* associadas aos sensores Hokuyo e suas dependências pela seguinte ordem: *image_common* e *hokuyo_node-indigo-devel*.

C.2 Procedimento normal para execução da arquitetura global de *software*

1. Ligar o sistema robótico no disjuntor para o efeito;
2. Ligar o manipulador FANUC no interruptor no controlador;
3. Executar o programa ROS, em modo automático, a partir do controlador do robô;
4. Ligar o LRF Hokuyo URG de forma a que fique associado à porta ttyACM0 do PC de bordo;
5. Da mesma forma, ligar o LRF Hokuyo UTM de forma a que fique associado à porta ttyACM1;
6. Estabelecer ligação TCP/IP com a rede Robonuc;

7. Estabelecer a ligação do PC remote ao PC de bordo utilizando o protocolo ssh: `$ ssh -Y robuter@192.168.0.123`;
8. Correr o roscore no terminal remoto do PC do sistema: `$ roscore`;
9. Numa nova *shell*, repetir o passo 7. e executar o comando `$ export ROS_IP=192.168.0.123` seguido do comando de inicialização dos sensores Hokuyo: `$ roslaunch robonuc laser_setup.launch`;
10. Num novo terminal do PC remoto, exportar à variável de ambiente que permite o funcionamento do ROS distribuído: `$ export ROS_MASTER_URI=http://192.168.0.123:11311`;
11. Especificar o IP remoto através do comando: `$ export ROS_IP=192.168.0.26`;
12. Executar o comando `$ roslaunch fanuc_lrmate200id_moveit_config moveit_planning_execution.launch sim:=false robot_ip:=192.168.0.231`
13. Finalmente, num novo terminal, repetir o passo 7. e executar o comando `$ export ROS_IP=192.168.0.123` seguido do comando `$ roslaunch robonuc robotic_system.launch safety:=true laserb:=true laserf:=true` completando o lançamento de toda a arquitetura de *software*;

C.3 Lista de comandos disponíveis

A lista de comandos que o nodo `vs_keyboard` está preparado para receber é a seguinte:

1. Comando "1" - Movimento do manipulador para a posição *default* dada pelo ID enviado no próximo argumento;
2. Comando "2" - Movimento do manipulador para a posição cartesiana dada pelos próximos argumentos. A orientação *default* e a orientação;
3. Comando "3" - Movimento do manipulador para a posição definida pelos valores de juntas dados nos argumentos seguintes;
4. Comando "4" - Pedido da posição cartesiana da ponta do robô no referencial da sua base;
5. Comando "5" - Pedido da posição cartesiana da ponta do robô em refação ao referencial global;
6. Comando "6" - Execução da Demo de demostração;
7. Comando "7" - Controlo de I/O's;
8. Comando "8" - Navegação do sistema robótico controlada pelas setas do teclado.

Apêndice D

Resolução de problemas

Problema: Aquando a tentativa de execução do programa ROS.TP o surge o erro *TPIF-013 Other program is running*

Solução: Pressionar FCTN > ABORT (ALL) duas vezes, na consola do robô.

Problema: *ERROR connecting: No route to host* [vs_IO_client-4] process has finished cleanly

Solução: Verificar a ligação Ethernet do Arduino ao router ou switch do sistema robótico.

Problema: *ERROR connecting: Connection timed out [vs_IO_client-4] process has finished cleanly*

Solução: Verificar a ligação Ethernet do Arduino e/ou reiniciar a unidade de controlo de I/O's.

Problema: A interface RViz de controlo do manipulador não inicia e/ou não funciona corretamente.

Solução: Verificar a ligação Ethernet do PC ao controlador e certificar-se que o programa ROS.TP está em execução, verificando que a informação *12345 I RSTA Waiting for ROS state prox* e *12345 I RREL Waiting for ROS traj relay* aparece na consola. Após a conexão ser corretamente estabelecia deverá surgir a informação *12345 I RSTA Connected 12345 I RREL Connected.*

Problema: *Exception thrown while opening Hokuyo. Failed to open port: /dev/ttyACM0* ou *Failed to open port: /dev/ttyACM1*

Solução: Verificar a ligação USB dos sensores ao PC de bordo. Certificar-se ainda que o LRF Hokuyo URG está associado à porta *ttyACM0* e o Hokuyo UTM à porta *ttyACM1.*

Problema: Os relés da unidade de controlo atracam e desatracam mas os I/O's do robô não alteram o seu estado.

Solução: Certificar-se que em MENU>SETUP>BGLogic o programa MONIO está em execução e/ou verificar a ligação da placa de controlo de I/O's ao controlador do manipulador.