Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Tiago de Almeida Tavares

Tele-Operação Integrada da Plataforma ROBONUC para Bin-Picking Móvel

Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Tiago de Almeida Tavares

Tele-Operação Integrada da Plataforma ROBONUC para Bin-Picking Móvel

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado com Agregação do Departamento de Engenharia Mecânicada Universidade de Aveiro.

O júri / The jury

Presidente / President	**Prof. Doutor José Paulo Oliveira Santos** Professor Auxiliar da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor Vítor Manuel Ferreira dos Santos** Professor Associado com Agregação da Universidade de Aveiro (orientador)
	Prof. Doutor Pedro Mariano Simões Neto Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (arguente principal)

Agradecimentos / Acknowledgements

"Há Gente que fica na história da história da gente". Quero por isso, expressar o meu profundo agradecimento a todos, os que que me acompanharam e ajudaram nestes últimos 5 anos.
Aos meus pais pelo apoio incondicional e pela motivação durante todo o meu percurso académico.
À Inês pela compreensão, incentivo, apoio nas horas difíceis e especialmente pela ajuda e correção do meu português.
Aos meus amigos Almeidinha, Pinho e Bernardo por todo o tempo de ajuda e companheirismo durante estes 5 anos. Por todas aquelas tardes de estudo intensivo, em que se questionava e se procurava uma resposta pela origem do universo.
Ao professor Vítor Santos pela orientação, motivação e conhecimento transmitido nestes últimos dois anos.
À minha família e amigos em geral pela disponibilidade e incentivo.

Palavras-chave Manipulação móvel; Bin-picking; Navegação; Integração; ROS; Máquina de estados; Cinemática

Resumo

A manipulação móvel é uma área atualmente em crescimento quer na investigação, quer em aplicações para a indústria. Este tipo de sistemas são de grande complexidade, uma vez que necessitam de conjugar diferentes disciplinas.

A presente dissertação consiste na integração de uma plataforma móvel com um manipulador FANUC de forma a originar o sistema total (nomeado como ROBONUC). O principal objetivo deste trabalho foca-se assim em integrar os dois sistemas em ambiente *ROS*, de modo a conceber uma solução de tele-operação para execução de tarefas de *bin-picking*.

Antes de desenvolver a arquitetura para esse fim, foi necessário compreender as ferramentas existentes e calibrar todos os componentes inerentes.

Assim, a solução final resultou numa maquina de estados semi-automática para a realização de tarefas de *bin-picking*.

Foi também estudada a problemática da formulação para movimentação simultânea (plataforma com manipulador), concebidos e simulados dois modelos de cinemática integrada e foi efetuado um estudo da possibilidade de implementação futura desses modelos no sistema ROBONUC.

Resultados experimentais demonstram a adequação e viabilidade das soluções desenvolvidas.

Keywords Mobile manipulation; Bin-picking; Navigation; Integration; ROS; State machine; Kinematics

Abstract Mobile manipulation is an area of current growth on the researching and industrial fields. This type of systems are very complex, since they need to combine different disciplines.
The present dissertation consists of the integration of a mobile platform with a FANUC manipulator, in order to create the total system named ROBONUC. The main objective of this work is to integrate the two systems within the *ROS* environment in order to design a solution, which includes the possibility of tele-operation to perform *bin-picking* tasks.
Before developing the architecture for this purpose, it was necessary to perceive the tools already developed and to calibrate all the inherent components.
Thus, the final solution resulted in a semi-automatic state machine to perform *bin-picking* tasks.
It was also studied the formulation of simultaneous movement planning (platform with manipulator), conceived and simulated two models of integrated kinematics, and it was made a study about the possibility of future implementation of these models in the ROBONUC system.
Experimental results demonstrate the appropriateness and feasibility of the developed solutions.

Índice

1 Introdução **1**
- 1.1 Manipulação móvel e a indústria . . . 2
  - 1.1.1 AIMM: "Autonomous Industrial Mobile Manipulation" . . . 2
  - 1.1.2 Importância da adaptação e configuração . . . 3
  - 1.1.3 Autonomia partilhada com recurso à tele-operação . . . 3
  - 1.1.4 Desafios a superar . . . 4
- 1.2 Bin-picking . . . 5
- 1.3 Motivação e objetivos . . . 5
- 1.4 Estrutura da Dissertação . . . 6

2 Revisão do Estado da Arte **7**
- 2.1 Trabalhos Relacionados . . . 7
  - 2.1.1 Manipulação móvel para tarefas de Bin-Piking . . . 8
  - 2.1.2 Histórico de trabalhos para conceção do ROBONUC . . . 8
- 2.2 Aplicações industrias de manipulação móvel . . . 11

3 Infraestrutura Experimental: Hardware **13**
- 3.1 Plataforma móvel . . . 13
  - 3.1.1 CPU1- Mini PC GIGABYTE . . . 14
  - 3.1.2 CPU2- Arduino Leonardo ETH . . . 14
  - 3.1.3 CPU3- Arduino Micro . . . 15
  - 3.1.4 Lasers Hokuyo . . . 16
  - 3.1.5 Controlador XBox . . . 17
  - 3.1.6 Router Asus . . . 18
  - 3.1.7 Arquitetura de comunicação dos sistemas apresentados . . . 18
- 3.2 Manipulador Róbotico . . . 19
  - 3.2.1 Robô FANUC LR Mate 200iD . . . 19
  - 3.2.2 Sensor Kinect . . . 19
  - 3.2.3 Sensor Laser DT20Hi, CPU4 e CPU5 . . . 20
  - 3.2.4 Arquitetura dos sistemas apresentados . . . 21
- 3.3 Integração do Hardware . . . 22

4 Infraestrutura Experimental: Software **25**
- 4.1 ROS - Robot Operating System . . . 25
- 4.2 Plataforma Móvel . . . 27
  - 4.2.1 Cinemática . . . 27
  - 4.2.2 Localização . . . 28

4.2.3 Navegação Assistida . . . 30
4.2.4 Resumo da arquitetura total . . . 32
4.3 Manipulador Robótico para Bin-Picking . . . 34
4.3.1 Segmentação da Nuvem de Pontos . . . 34
4.3.2 Execução de Movimentos . . . 36
4.3.3 Ativação IO's . . . 37
4.3.4 Fluxo do Processo global de Bin-Picking . . . 38

5 Calibração **41**
5.1 Manipulador-plataforma . . . 41
5.2 Lasers Hokuyo-Manipulador . . . 43
5.3 Câmara-Manipulador . . . 45
5.3.1 Calibração dos Parâmetros Intrínsecos . . . 45
5.3.2 Calibração dos Parâmetros Extrínsecos . . . 47
5.4 Laser 1D-Manipulador . . . 49
5.5 Sistema Total Calibrado e Integrado . . . 50

6 Arquitetura Integrada **53**
6.1 Filosofia dos estados . . . 53
6.2 Controlo entre estados . . . 54
6.3 Estado 1: Navegação . . . 56
6.4 Estado 2: Aproximação . . . 57
6.5 Estado 3: Orientação . . . 59
6.6 Estado 4: Bin-Picking . . . 62
6.7 Controlo Remoto . . . 66
6.7.1 Interface . . . 66
6.7.2 Controlador Interativo . . . 68
6.7.3 ROS Distribuído . . . 68

7 Cinemática Integrada **71**
7.1 Problemática e Trabalhos relacionados . . . 71
7.2 Modelos de cinemática propostos . . . 74
7.2.1 Modelação cinemática . . . 74
7.2.2 Cinemática diferencial . . . 78
7.2.3 Manipulabilidade . . . 79
7.2.4 Implementação prática . . . 79
7.3 Aplicabilidade da cinemática integrada no ROBONUC . . . 88

8 Testes e Resultados **91**
8.1 Controlo Remoto . . . 91
8.2 Ambiente 1 . . . 91
8.3 Ambiente 2 . . . 94
8.4 Análise dos estados propostos em diferentes velocidades . . . 98
8.5 Análise de uma trajetória "reta" . . . 99
8.6 Demonstração de trajeto sinusoidal . . . 99
8.7 Conclusão . . . 102

9 Conclusões e Trabalho Futuro **103**
9.1 Conclusões . . . 103
9.2 Trabalho Futuro . . . 104

A Execução dos processos desenvolvidos **113**
A.1 *Setup* inicial . . . 113
A.2 Calibração extrínseca: Kinect . . . 114
A.3 *Launch* Files . . . 114
A.3.1 integrated_referee.launch . . . 114
A.3.2 camera_extr_calibration.launch . . . 116
A.3.3 display_total_urdf.launch . . . 117
A.4 Cinemática integrada . . . 117
A.5 Demonstração Sinusoide . . . 118

B Ficheiros Auxiliares **119**
B.1 Diagrama de Navegação . . . 119
B.2 Árvore de transformações do ROBONUC . . . 119

Lista de Tabelas

3.1 Características do CPU1 [30]. 14
3.2 Características do CPU2 [22]. 15
3.3 Características do CPU3 [32]. 16
3.4 Características do laser Hokuyo UTM-30LX [33]. 17
3.5 Características do laser Hokuyo URG-04LX-UG01 [34]. 17
3.6 Características da *Microsoft Kinect* [38]. 20
3.7 Características do sensor *SICK DT20HI* [39]. 21
3.8 Tabela representativa dos cálculos de potência. 22
3.9 Características do Inversor 48 V (DC-AC) [41]. 23

6.1 Descrição da função dos botões do comando *XBox* da figura 6.2. 68

7.1 Tabela dos parâmetros D-H usados na abordagem 1. 75
7.2 Tabela dos parâmetros D-H usados na abordagem 2. 78
7.3 Tabela referente aos valores usados para comprimentos dos elos. 79

8.1 Tabela Representativa do sucesso de cada estado em cada experiência efetuada no ambiente 1. 92
8.2 Tabela que descreve o sucesso de cada estado em cada experiência efetuada no ambiente 2. (* sucesso do estado à segunda tentativa) 95
8.3 Representação das velocidades testadas para cada grupo T_i. 98

Lista de Figuras

1.1 Exemplos de trabalhos realizados em diferentes áreas [3]. 1
1.2 Arquitetura base dos *AIMM* [3]. 2
1.3 Infraestrutura exemplo de bin-picking [11]. 5

2.1 Cenário encontrado em [19]. 8
2.2 Representação reconversão elétrica do Robuter II realizada em [20]. 9
2.3 Representação minimalista da integração realizada em [21]. 9
2.4 Representação do processo *Bin-Picking* concebido em [13]. 10
2.5 Representação do processo de navegação realizado em [22]. 10
2.6 KUKA Mobile Robotik iiwa - KMR iiwa [24]. 11
2.7 Exemplos de aplicações industriais [28], [29]. 12

3.1 Plataforma ROBONUC. 13
3.2 CPU1: Mini-PC GIGABYTE GB-BKi7A-7500 [30]. 14
3.3 CPU2: *Arduíno Leonardo ETH* [31]. 15
3.4 CPU3: *Arduíno Micro* [32]. 15
3.5 *Hokuyo UTM-30LX* [33]. 16
3.6 *Hokuyo URG-04LX-UG01* [34]. 17
3.7 *GamePad Microsoft XBox 360* [35]. 18
3.8 *Router Asus RT-AC51U* [36]. 18
3.9 Diagrama de comunicações da plataforma móvel [22]. 19
3.10 *FANUC LR Mate 200iD* [37]. 19
3.11 *Micosoft Kinect* [38]. 20
3.12 *SICK DT20HI* [39]. 20
3.13 Arduíno UNO [40]. 21
3.14 Arquitetura dos componentes do manipulador e respetivas comunicações. . 21
3.15 Inversor de tensão (DC-AC) instalado [41]. 22
3.16 *HUB* UH700 [42]. 23
3.17 Representação do sistema após montagem de todo o *hardware* necessário. 24

4.1 Logótipo `ROS Melodic Morenia` [45]. 26
4.2 Conceitos básicos inerentes ao `ROS` [46]. 26
4.3 Estado de transições de um servidor de ações [48]. 27
4.4 Cinemática do robô com locomoção diferencial [50]. 28
4.5 Diagrama de leitura dos *encoders* [22]. 29
4.6 Sistemas de coordenadas necessários para o `HectorSLAM` [22]. 30
4.7 Trajetória calculada pelo modo automático. 31
4.8 Diagrama de funcionamento do modo semi-automático [22]. 32

4.9 Diagrama de funcionamento do modo automático e semi-automático [22]. 33
4.10 Nuvens de pontos pós-tratamento. 35
4.11 Pontos de aproximação e transformada laser-*end effector*. 36
4.12 Arquitetura `Moveit` [59]. 37
4.13 Circuito externo para ativar IO's. 38
4.14 Arquitetura esquemática do processo de *Bin-Picking* [13]. 39

5.1 Representação visual do modelo URDF da plataforma. 42
5.2 Representação visual do modelo URDF (plataforma + manipulador + *gripper*). 43
5.3 Representação do processo de calibração dos lasers Hokuyo e dos referenciais necessários. 44
5.4 Representação visual do modelo URDF completo e respetivos referenciais associados. 45
5.5 Representação do processo de calibração intrínseca. 46
5.6 Representação das melhorias após o processo de calibração intrínseca, comparativamente aos parâmetros usados no trabalho anterior. 46
5.7 Representação das melhorias após o processo de calibração intrínseca, comparativamente aos parâmetros padrão. 47
5.8 Representação do processo de calibração extrínseca. 48
5.9 Representação do processo de calibração do laser 1D. 50
5.10 Árvore de transformações do modelo total. 51
5.11 URDF do sistema total integrado. 52

6.1 Diagrama de estados. 54
6.2 Identificação dos botões do comando *XBOX 360* [22]. (Descrição na tabela 6.1.) 54
6.3 Representação das posições do manipulador para os diferentes estados. . . 55
6.4 Fluxograma de funcionamento do `integrated_referee`. 56
6.5 Representação simplista da arquitetura inerente ao estado 1. 57
6.6 Representação da arquitetura inerente ao estado 2. 58
6.7 Fluxograma inerente ao servidor de ações correspondente ao estado 2. . . 59
6.8 Exemplo da posição da plataforma após a aproximação da bancada. . . . 60
6.9 Representação da orientação pretendida da plataforma móvel. 60
6.10 Representação do processo de orientação da plataforma móvel. 61
6.11 Representação da arquitetura inerente ao estado 3. 62
6.12 Fluxograma inerente ao servidor de ações, correspondente ao estado 3. . . 63
6.13 Fluxograma da tarefa de *bin-picking* concebida em [13]. 64
6.14 Representação da arquitetura inerente ao estado 4. 65
6.15 Fluxograma inerente ao servidor de ações correspondente ao estado 4. . . 66
6.16 Interface para controlo remoto. 67
6.17 Representação de exemplos de informação apresentada na interface gráfica, durante o processo de *bin-picking*. 67

7.1 Representação de cooperação de manipuladores móveis na realização de tarefas [86]. 74
7.2 Representação ilustrativa das variáveis usadas na abordagem 1. 75

7.3 Sistemas de coordenadas definidos para o manipulador robótico na abordagem 1. 76
7.4 Sistemas de coordenadas definidos para o sistema total na abordagem 2. . 77
7.5 Representação do sistema na posição *hardware* para a abordagem 1. . . . 80
7.6 Representação do sistema na posição *hardware* para a abordagem 2. . . . 80
7.7 Representação da trajetória 1 definida pela abordagem 1. 81
7.8 Representação da trajetória 1 definida pela abordagem 2. 82
7.9 Representação da posição final do sistema, para a trajetória 1 definida pela abordagem 1. 82
7.10 Representação da posição final do sistema, para a trajetória 1 definida pela abordagem 2. 83
7.11 Representação dos valores de manipulabilidade do sistema total ao longo da trajetória 1, para as diferentes abordagens. 83
7.12 Representação da trajetória 2 definida pela abordagem 1. 84
7.13 Representação da trajetória 2 definida pela abordagem 2. 85
7.14 Representação da posição final do sistema, para a trajetória 2 definida pela abordagem 1. 85
7.15 Representação da posição final do sistema, para a trajetória 2 definida pela abordagem 2. 86
7.16 Representação dos valores de manipulabilidade do sistema total ao longo da trajetória 2, para as diferentes abordagens. 86
7.17 Representação dos valores de manipulabilidade do braço robótico ao longo da trajetória 1, para as diferentes abordagens. 88
7.18 Representação dos valores de manipulabilidade do braço robótico ao longo da trajetória 2, para as diferentes abordagens. 88
7.19 Representação da arquitetura atual para controlo do manipulador FANUC por parte do `moveit`. 90

8.1 Representação ilustrativa da sequência de estados de uma tarefa completa. 92
8.2 Representação ilustrativa da falta de alcance num determinado teste, do ambiente 1. 93
8.3 Representação da colisão no teste 7 no ambiente 1. 93
8.4 Representação do ambiente 2. 94
8.5 Representação da posição dos objetos na bancada presente no ambiente 2. 94
8.6 Representação do modo de navegação automático no ambiente 2. 95
8.7 Representação das falhas do modo de navegação automático no ambiente 2. 96
8.8 Representação do processo de *bin-picking* à segunda tentativa. 97
8.9 Representação da nuvem de pontos obtida com os parâmetros intrínsecos de *default*. 97
8.10 Representação da orientação inicial e final da plataforma, com velocidades definidas em T_6. 98
8.11 Representação dos 5 testes de hodometria para uma trajetória retilínea. . 99
8.12 Representação da posição inicial e final da trajetória realizada num dos testes. 100
8.13 Fluxograma inerente ao nó `move_sinusoide.py`. 101
8.14 Representação da sinusoide resultante no ambiente utilizado. 101

8.15 Representação da sinusoide resultante do movimento integrado do sistema total. 102

A.1 Representação do menu do *teach pendant* para executar o *Background logic.* 114

Capítulo 1

Introdução

O termo ”manipulação móvel” nasceu nos anos noventa, quando alguns investigadores decidiram inserir um manipulador robótico numa plataforma móvel, ou seja um dispositivo que permite o desempenho de tarefas que exigem habilidades de locomoção e manipulação [1]. Inicialmente, plataformas móveis e braços robóticos foram estudados separadamente, mas com o avanço da investigação, estes sistemas tendem para sistemas mecânicos autónomos que combinam a mobilidade e manipulação [2]. Desde então, esta área da robótica tem tido um bom progresso devido à realização de inúmeros trabalhos de pesquisa e desenvolvimento, em diferentes áreas, tais como área dos serviços (profissional e/ou doméstico), espacial, militar e industrial (exemplos na Figura 1.1).

Figura 1.1: Exemplos de trabalhos realizados em diferentes áreas [3].

Esta área continua na linha da frente da investigação e pode ser vista como um ”Grande Desafio” para robótica. A implementação bem-sucedida de robôs autónomos para executar tarefas complexas de manipulação em ambientes não estruturados, além de exigir avanços significativos em uma variedade de áreas de robótica e visão computacional, também exigirá que essas novas tecnologias sejam integradas em uma única, robusta plataforma robótica autónoma [4].

1.1 Manipulação móvel e a indústria

As capacidades de manipulação móvel são fundamentais para muitas novas aplicações da robótica em diversas áreas, inclusive na área da indústria. A introdução de um robô para ajudar o ser humano, aumenta de forma exponencial o desempenho geral das tarefas, uma vez que reduzirá a fadiga, aumentará a precisão e melhorará a sua qualidade final [5]. Na indústria, os robôs são amplamente utilizados para executar tarefas 4D (*Dumb*, *Dangerous*, *Dull* e/ou *Dirty*) com certos benefícios adicionais, tais como o espaço de trabalho estendido, maior flexibilidade operacional e maior eficácia. A montagem, a inspeção, o *bin-picking*, assim como diversos serviços em ambientes perigosos/prejudiciais à saúde humana, são alguns exemplos dessas aplicações industriais [6].

1.1.1 AIMM: "Autonomous Industrial Mobile Manipulation"

Com o passar do tempo, surgiu a definição de *AIMM* (*Autonomous Industrial Mobile Manipulation*), na qual a arquitetura base deste tipo de sistemas pode ser visualizada na figura 1.2.

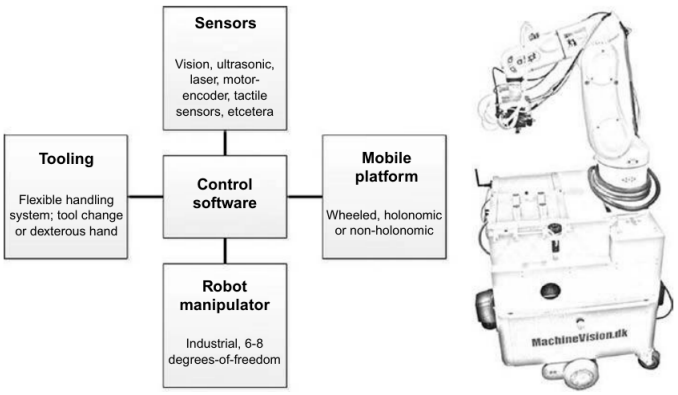


Figura 1.2: Arquitetura base dos *AIMM* [3].

Esta tecnologia requer a integração de tecnologias e conceitos, que usualmente são tratados de forma independente [3]:

Autonomia refere-se à capacidade de controlar e realizar tarefas versáteis, sem contínua intervenção humana. Tal competência, exige diversos mecanismos para extrair informações do ambiente envolvente, de modo a evitar acidentes, recuperar falhas e modificações de planos com base no *feedback* de tempo de execução.

Industrial é o conceito relacionado com o domínio de aplicação e ambiente de trabalho do sistema robótico. Este termo refere ambientes de fabricação típicos, de parcialmente a totalmente estruturados, onde robôs estão presentes na execução de enumeras tarefas.

Móvel descreve a capacidade de o robô se mover livremente em determinados ambientes, contribuindo para um elevada flexibilidade. Além disso, este termo remete para o mapeamento e localização no ambiente inserido.

Manipulação implica trabalho mecânico para modificar o arranjo ou a posição de objetos.

Numa célula de trabalho, que contem um robô industrial estacionário, a localização precisa dos objetos é tipicamente conhecida, assim, a colisão, caminhos livres e manipulação podem ser programados previamente, uma vez que todos os movimentos são repetitivos. Contrariamente, na manipulação móvel, a localização precisa do sistema do robô no ambiente inserido, não é conhecida. Apesar das enumeras técnicas de localização adotadas (como por exemplo, scanners a laser), estas continuam a apresentar erros na ordem dos centímetros, portanto, muitas das vezes é necessário calibrar o sistema do robô em relação ao ambiente de trabalho. Algumas abordagens usam câmaras para identificar marcadores fixos na cena [7].

Para a navegação fiável, em ambientes onde existe a presença de pessoas, o sistema de navegação do robô deve ser capaz de lidar com ambientes dinâmicos. Deve garantir a segurança dos trabalhadores bem como preservar o ambiente que o rodeia. De modo a especificar requisitos para um sistema fiável de navegação, as diferentes possibilidades de implementação do sistema de navegação de um robô móvel devem ser analisadas. Além disso, o grau de confiança deve ser avaliado para cada solução [7].

1.1.2 Importância da adaptação e configuração

De modo a superar todos os desafios em ambientes industriais, de uma forma autónoma, a adaptação é um aspeto central (diretamente relacionado com a autonomia), ou seja o sistema deve ”aprender” e ”raciocinar” de modo a adaptar-se a uma situação particular (desconhecida) sem instruções humanas. Caso os ambientes em causa sejam bem estruturados e apresentem configurações estáticas, a adaptação é frequentemente uma tarefa relativamente simples, podendo ser utilizadas tecnologias como localização e mapeamento simultâneos (SLAM), planeamento de movimento e prevenção de obstáculos.

Uma outra característica essencial nos Robôs *AIMM* é a sua configuração, isto é, uma mudança no sistema robótico realizada pelo operador, quando o sistema não estiver no modo operacional. Um operador/trabalhador, com um mínimo de conhecimento, deve ser capaz de configurar ou até mesmo auxiliar robôs *AIMM* para tarefas e aplicações típicas de fabricação. Este tipo de ações são extremamente relevantes, uma vez que os ambientes de operação, podem ser relativamente pouco estruturados, o que leva a imprevistos durante as operações.

1.1.3 Autonomia partilhada com recurso à tele-operação

O nível de autonomia de um robô está diretamente relacionado com a previsibilidade da tarefa e do ambiente inserido. Na execução de tarefas previsíveis em ambientes previsíveis, os robôs trabalham de forma autónoma, enquanto que para robôs onde a variação é alta, a tele-operação, ainda é a solução mais aceite, de modo a tomar decisões ou de modo a que o robô atue como uma extensão ao operador humano. Muitas aplicações na indústria são padronizadas e exigem altos níveis de concentração e destreza manual, assim a sinergia entre humano-robô pode aumentar significativamente o desempenho geral das tarefas. Surge assim, o conceito de ”autonomia partilhada” ou seja, um meio-termo, que combina sequências autónomas pelo robô com a contribuição de um operador humano

para a tomada de decisões de alto nível, reduzindo a carga cognitiva e convergindo para sistemas mais fidedignos [8].

Na tele-operação, o robô pode ser controlado apenas por dispositivos de entrada (como um *joystick* por exemplo), no entanto, algumas tarefas podem ser automatizadas para facilitar o controlo por parte do operador. Se o robô não conseguir gerir a tarefa de forma autónoma, o operador pode controlar ou auxiliar manualmente o robô de modo a que consiga concluir a tarefa desejada. Existem diversos projetos já realizados que usam esta metodologia de modo a dar resposta a ambientes pouco estruturados, especialmente em robôs de serviços (domésticos) como é o caso de um TSR (Tele-operated Service Robot) em [9].

Para alem desta abordagem, existem diversos robôs que apenas são tele-operados, sem qualquer tipo de autonomia, em que o controlo está um pouco mais desenvolvido tecnologicamente, uma vez já se substituem os *joysticks* por equipamentos mais avançados, como por exemplo sistemas de realidade virtual conjugada com sensores no corpo do operador de modo a que o robô emite os movimentos do operador (em [10]). Este tipo de robôs, são bastante utilizados, em diversas áreas: a militar e espacial; a medicina de modo a realizar operações á distancia; prestadores de serviços de modo a auxiliar pessoas com dificuldades motoras; e até mesmo na industria, por exemplo na manipulação de objetos pesados.

1.1.4 Desafios a superar

Os maiores desafios para o desenvolvimento e implementação de *AIMM* têm sido as questões de credibilidade (segurança), economia, e de configuração.

Em primeira instância, a tecnologia *AIMM* nunca será aceite pela indústria se não for segura, quer em relação aos seres humanos quer em relação á maquinaria/equipamento. Muitas abordagens diferentes têm vindo a ser propostas para a segurança humana, tais como o uso de diferentes sensores e algoritmos para deteção humana e uso de robôs leves controlados por torque. No entanto, estas tecnologias precisam de ser governamental e/ou industrialmente padronizadas e aprovadas. Atualmente, a segurança pode ser assegurada por outros meios, para alem dos sensores avançados, tais como a restrição da presença humana dentro da área de trabalho. No entanto, isso não é uma solução a longo prazo [3].

Em segunda instância, as propriedades físicas do sistema AIMM, ou seja, o design do sistema e arquitetura, poderá ser uma possível barreira. Como a tecnologia *AIMM* é uma solução de automação versátil, é desejável maximizar a flexibilidade do sistema, no entanto, não é adequado criar um sistema universal e multi-funcional que resolva todas as tarefas e aplicações industriais. Este problema pode ser abordado tendo em conta uma personalização em massa, ou seja, o uso da modulação e configuração. Uma vez que estes tipo de produtos são dispendiosos, utilizar uma lógica de design modular e plataformas *AIMM* padronizadas, os sistemas podem ser facilmente reconfigurados de modo a dar resposta a várias tarefas [3].

Em terceira instância, a configuração e interação trabalhador-robô. O objetivo principal para os robôs *AIMM* é apresentar uma tecnologia com uma programação simples e flexível de modo a que a interação homem-robô seja facilitada. Portanto, é necessário pensar em interações sem esforço através de interfaces multimodais e linguagens naturais. O trabalhador sem muito esforço, poderá ser capaz de atribuir missões complexas a robôs

AIMM de uma maneira rápida e fiável, fazendo uso de ferramentas de software para configuração semi-automática, ou sistemas funcionais que usem lógicas de aprendizagem e assistência [3].

Desta forma, a investigação associada à manipulação móvel aumenta progressivamente a autonomia, robustez e o número de aplicações de sistemas robotizados, ao mesmo tempo que reduz gradualmente a dependência de informações prévias sobre o ambiente inserido e as tarefas a desenvolver. Este processo será gradual, e permitirá desenvolver interfaces intuitivas, de modo a facilitar a interação homem-robô, e eventualmente, levará a um melhor desempenho e a um menor custo.

1.2 Bin-picking

A técnica usada por um robô para capturar objetos dispostos aleatoriamente dentro de uma caixa ou palete, representada na figura 1.3, é designada de *bin-picking*.

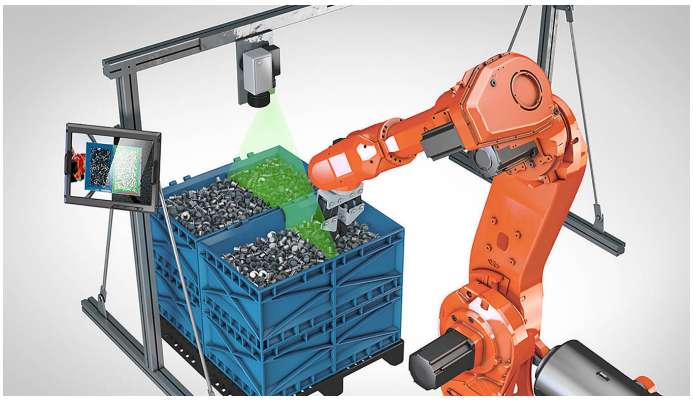


Figura 1.3: Infraestrutura exemplo de bin-picking [11].

Com auxilio de sistemas de visão e sensores, o robô analisa uma superfície, deteta os objetos a serem apanhados e move-os para um local desejado. No entanto, o processo de deteção de objetos e identificação da sua posição correta, é uma tarefa bastante vulgar quando realizada por um humano, mas de extrema dificuldade quando realizada por um robô, o que torna esta tarefa um dos grandes desafios na área de robótica industrial, devido à complexidade da perceção [12].

Uma vez que o robô necessita de localizar a peça num ambiente não estruturado, onde os objetos estão em posições e orientação aleatórias, esta técnica, requer um equilíbrio entre a destreza do manipulador, a visão artificial, o software, e o poder de processamento, de modo a extrair as peças sem colisões ou danos [13].

1.3 Motivação e objetivos

Atualmente, a manipulação móvel e *bin picking* são áreas com grande foco devido à necessidade de conjugar diferentes disciplinas, resultando assim, em tarefas de extrema complexidade. Este trabalho visa o culminar destas duas áreas, o que, de certa forma, aparenta um ótimo desafio, visto que reúne várias competências da engenharia. A existência de um manipulador robótico (FANUC) e de uma plataforma móvel no laboratório

(LAR), permitiu a idealização deste projeto. Estes dois sistemas apresentam algumas ferramentas desenvolvidas em projetos anteriores, e pretende-se aproveita-las de modo a integrar os sistemas. Após a sua integração, este sistema poderá ser usado para estudo de problemas e desafios mais complexos que não seriam possíveis sem a conceção da sua arquitetura integrada.

Numa primeira fase, é necessário compreender os sistemas individualizados e as suas ferramentas, para posteriormente instalar e calibrar os mesmos. Posto isto, além de conceber, poderá ser necessário alterar as ferramentas existente de forma a atingir uma integração funcional robusta, e nesse seguimento, será necessário desenvolver aplicações ilustrativas para explorar as funcionalidades desejadas. Para o efeito, deve-se ter em conta o grau de autonomia do conjunto, e se necessário, adotar estratégias de forma a aumentar a sua independência energética. Por fim, será abordado a possibilidade de conceber um modelo de cinemática "global", bem como a forma de implementar uma solução desse tipo com as ferramentas disponíveis.

1.4 Estrutura da Dissertação

A dissertação é composta por nove capítulos, e será organizada da seguinte forma:

1. **Introdução -** O presente capitulo, apresenta uma breve introdução dos temas inerentes á dissertação e o seu enquadramento com a industria. É apresentado o problema e são enumerados os objetivos propostos.

2. **Estado da Arte -** Descreve um projeto desenvolvido no âmbito de manipulação móvel para *bin-picking*, alguns trabalhos com contribuição para o projeto atual, bem como algumas soluções de manipulação móvel comercializadas para aplicações industriais.

3. **Infraestrutura Experimental: Hardware -** Introduz e descreve o *hardware* usado para o projeto.

4. **Infraestrutura Experimental: Software -** Apresenta e descreve todo o *software* inerente ao projeto.

5. **Calibração -** Descreve as técnicas de calibração entre cada sistema de coordenadas envolvidos.

6. **Arquitetura Integrada -** Descreve todo o processo de integração do *software* e o funcionamento do sistema integrado.

7. **Cinemática Integrada -** Apresenta-se os problemas da modelação cinemática e dinâmica do sistema integrado, e propõem-se duas soluções de modelos cinemáticos para o sistema ROBONUC. Expõem-se também ferramentas que podem possibilitar ou não, essa integração.

8. **Testes e Resultados -** Descreve todos os testes efetuados de modo a demonstrar e a avaliar o trabalho desenvolvido.

9. **Conclusões e Trabalho Futuro -** Apresenta conclusões do trabalho desenvolvido e algumas sugestões para trabalhos futuros inseridos neste tema.

Capítulo 2

Revisão do Estado da Arte

A manipulação móvel, na industria ou na ciência, é uma área em constante crescimento que tem vindo a ser alvo de apostas ao longo dos anos. Este capitulo descreve e enumera alguns trabalhos desenvolvidos cientificamente e industrialmente, e expõe os trabalhos diretamente relacionados com o hardware utilizado na presente dissertação.

2.1 Trabalhos Relacionados

Sendo a manipulação móvel, uma área de investigação bastante desafiante que conjuga varias áreas da robótica, encontram-se na literatura um número alargado de trabalhos referentes a este tema.

Em [8], o objetivo consiste em realizar tarefas de *pick-and-place*, em bancadas de trabalho, de forma autónoma e/ou semi-autónoma, ou seja, o manipulador móvel também pode ser tele-operado. Para uma melhor perceção do ambiente, e de modo a evitar colisões, o ambiente é reconstruido tridimensionalmente usando *voxel grids*. Salienta-se o uso da técnica *inverse Dynamic Reachability Maps (iDRM)*, desenvolvida em [14], de modo a que a base móvel atingia uma posição capaz de maximizar a manipulabilidade, aumentando assim a eficiência da tarefa.

Muitos projetos, têm vindo a ser desenvolvidos em robôs de serviços, uma vez que os ambientes não-estruturado e a constante presença de humanos, representam um desafio acrescido. São exemplos destes projetos, o [15], que faz uso de um PR2 (robô atualmente muito usado para tarefas de manipulação móvel), e [16].

Para tarefas exteriores, com características adversas, também se encontram alguns trabalhos, como o caso de [17] e [18]. No entanto, como era de esperar, devido à natureza do ambiente, estes robôs apresentam uma maior dependência da tele-operação.

Apesar dos enumeros trabalhos encontrados, o uso de sistemas independentes continua a prevalecer, isto é, a navegação e a manipulação são tarefas distintas. O facto do sistema global ser excessivamente redundante pode dificultar a implementação de uma cinemática global do sistema, de modo a que os sistemas atuem em simultâneo para um dado ponto objetivo. Em alternativa, o uso de alternância de sistemas, de modo a que exista uma cooperação entre eles, poderia ser aplicada, no entanto, algoritmos de otimização como o *iDRM*, são uma alternativa e aparentam apresentar uma maior eficiência.

Realça-se que, na sua maioria, os trabalhos encontrados referentes a esta área, são

desenvolvidos em ambiente *ROS*, e as plataformas móveis usadas são omnidirecionais.

2.1.1 Manipulação móvel para tarefas de Bin-Piking

Salienta-se o trabalho [19], que consistiu em desenvolver um robô de serviços móvel e autónomo para tarefas de bin-piking, no qual o cenário encontrado está representado na figura 2.1. Para o desenvolvimento do projeto foi utilizado o sistema operativo ROS.

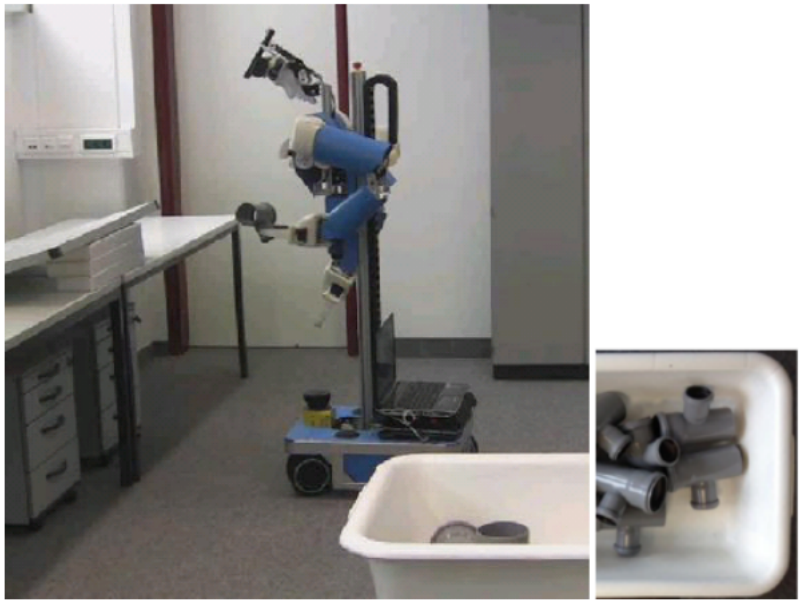


Figura 2.1: Cenário encontrado em [19].

A tarefa de *bin-picking* é dividida na fase cognitiva, em que o robô explora a caixa com objetos e reconhece os objetos superiores, a fase de *picking*, onde o robô agarra o objeto com a cota mais elevada e a fase de "largada", onde o robô coloca o objeto no local final. Para navegação o robô utiliza tecnicas tradicionais como o *Adaptive Monte Carlo Localization* utilizando um *Laser-Range Finder* (LRF). De modo a posicionar o robô para a concretização total do *picking*, a caixa localiza-se sempre numa posição predefinida no mapa global, e utiliza-se um laser 2D para o posicionamento mais preciso. No reconhecimento de objetos é usado um algoritmo que compara a nuvem de pontos adquirida (pela *Microsoft Kinect*), com formas simples (esferas, cilindros, planos) de modo a detetar as suas posições para posterior *grasping*.

2.1.2 Histórico de trabalhos para conceção do ROBONUC

Ao longo dos últimos anos, foram realizadas algumas dissertações do LAR de modo a desenvolver o projeto "ROBONUC" da melhor forma possível. São exemplos os seguintes trabalhos apresentados.

Reconversão da Plataforma Robuter num AGV com Guiamento Visual

Nesta dissertação, Bruno Vieira, em 2017, fez um *retrofitting* da plataforma Robuter II, com o objetivo de substituir toda a parte de controlo por uma arquitetura baseada em ROS. Foi realizada uma intervenção elétrica (figura 2.2), e instalados alguns equipamentos de controlo (apresentados num capitulo mais á frente). Neste trabalho, obteve-se um veículo capaz de navegar de forma manual, e foi desenvolvido um algoritmo simples de visão artificial para navegação autónoma [20].

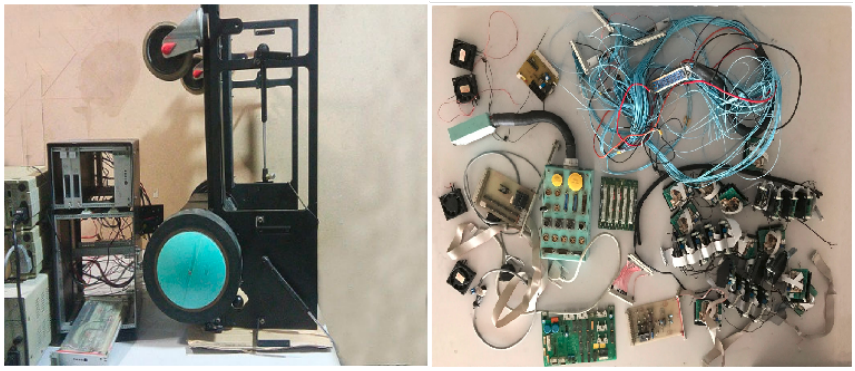


Figura 2.2: Representação reconversão elétrica do Robuter II realizada em [20].

Integração de Manipulador FANUC na plataforma Robuter para Manipulação Móvel

Nesta dissertação, Vítor Silva, em 2017, focou-se essencialmente na integração física dos sistemas, figura 2.3. Idealizou e construiu uma estrutura com resistência mecânica para que a integração dos dois sistemas fosse possível. Para esta validação, foi realizada uma interface com o utilizador através do teclado para atuação nos dois sistemas. Devido às limitações do ROS-FANUC foi necessário conceber uma solução externa de controlo de I/O's para atuação do *gripper* do manipulador [21].

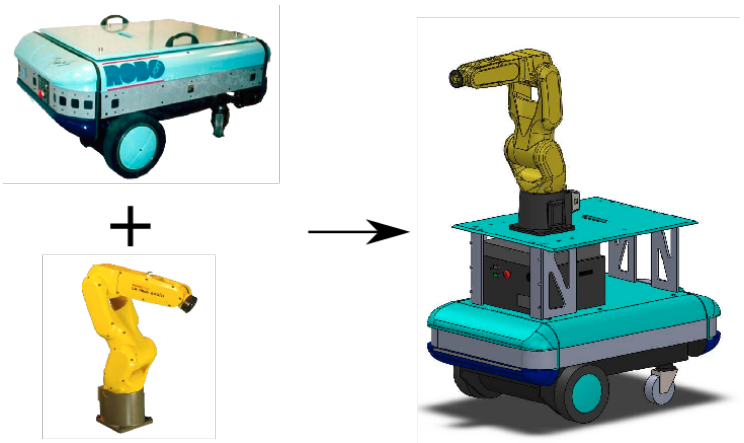


Figura 2.3: Representação minimalista da integração realizada em [21].

Bin-Picking de Precisão usando um Sensor 3D e um Sensor Laser 1D

Em 2018, Joana Mota desenvolveu uma solução para execução de um processo de bin-picking de objetos pequenos e frágeis (figura 2.4) sem os partir ou deformar. Para o efeito, necessitou de integrar um Manipulador FANUC, com um sensor 3D (kinect), e um sensor laser 1D para medição precisa de distancias [13].

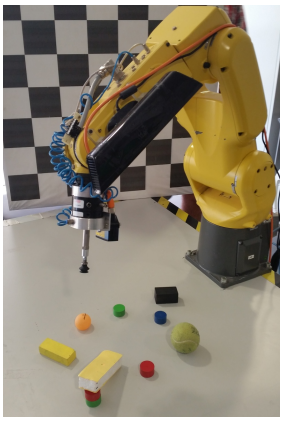


Figura 2.4: Representação do processo *Bin-Picking* concebido em [13].

Navegação Assistida e Semi-Autónoma da Plataforma ROBONUC

Luís Sarmento, em 2018, implementou um sistema de controlo remoto para a plataforma móvel, com mecanismos de autonomia suficientes para atuar preventivamente. Com base nos leituras dos lasers, e nas áreas (geométricas) de risco pré-definidas, a maquina comutava de estado em caso de risco de colisão. O mapa do local a navegar é reconstruido á medida que a plataforma é tele-operada, e apresenta mecanismos de localização da plataforma implementados, como é o caso da hodometria. Em suma, no final da dissertação, a plataforma poderia navegar de forma semi-automática ou automática [22]. O resultado da interface final pode ser visualizada na figura 2.5.

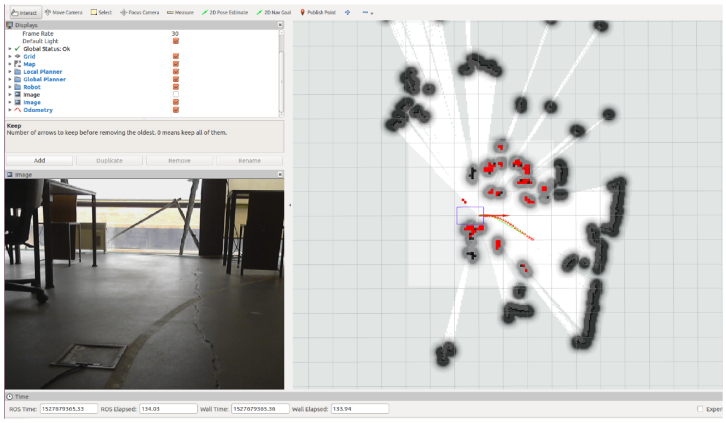


Figura 2.5: Representação do processo de navegação realizado em [22].

2.2 Aplicações industrias de manipulação móvel

Ao longo dos anos, a manipulação móvel na industria tem vindo a ser introduzida por algumas entidades, especialmente pela marca *KUKA*. Em 2013 a *KUKA* lançou o seu primeiro robô móvel para o mercado, o *youbot*, que executava tarefas de *pick-and-place*. O operador tinha duas formas de indicar qual o objeto a apanhar pelo o robô, selecionava um objeto contemplado na base de dados com o respetivo modelo 3D, ou escolhia um ponto através da nuvem de pontos fornecida pela *kinect* do robô [23].

Recentemente, a *KUKA* lançou o *KUKA Mobile Robotik iiwa* (figura 2.6), um robô que segue numa lógica *pick-and-place*, preparado para frequentar locais movimentados devido ao seu sistema de segurança. Durante o movimento, caso detete obstáculos no caminho planeado e na hipótese de haver espaço livre, este consegue contornar-los. O robô utiliza um mapa (previamente inserido) do ambiente que frequenta de modo a melhorar o seu sistema de localização [24].

Figura 2.6: KUKA Mobile Robotik iiwa - KMR iiwa [24].

Com o avanço da investigação, vários institutos de investigação têm vindo a destacar-se na área de manipulação móvel. Um destes casos, o *INESC-TEC* [25], tem vindo a desenvolver vários projetos nesta área, realçando-se projetos como o Stamina [26], o CARLoS [27] e o mais recente, representado na figura 2.7a, o ColRobot [28].

Uma vez que este tipo de sistemas começa a ter uma grande procura, devido à sua versatilidade, também têm vindo a surgir novas empresas que o desenvolve. Um dos exemplos é a *ClearPathRobotics* [29], uma empresa que oferece soluções robóticas (de *software* ou *hardware*), e integra os sistemas que o cliente desejar em ambiente *ROS*. Na figura 2.7b é apresentado um exemplo desenvolvido pela *ClearPathRobotics*.

(a)

(b)

Figura 2.7: Exemplos de aplicações industriais [28], [29].

Capítulo 3

Infraestrutura Experimental: Hardware

A plataforma *RONONUC*, resultou da conversão da plataforma *ROBUTER II* de modo a ser controlado com sistema `ROS` e integrado com o manipulador FANUC. Deste modo, foi necessário implementar melhorias estruturais, e substituir/acrescentar alguns componentes. Assim, este capitulo tem como propósito enumerar os sistemas físicos inerentes à plataforma móvel e ao manipulador, de forma a compreender o seu funcionamento geral. Para informações mais específicas, pode consultar-se os documentos [20], [21], [22] e [13].

3.1 Plataforma móvel

De modo a que a plataforma *RONONUC* possua tração nas duas rodas dianteiras, dispõem de um motor elétrico em cada roda, no qual, acoplado, apresenta um *encoder* e um travão. Com o intuito de oferecer estabilidade à plataforma, esta possui complementarmente, duas rodas traseiras giratórias de menor raio, como se pode observar na imagem 3.1.

Figura 3.1: Plataforma ROBONUC.

Para processar toda a informação necessária, a unidade de processamento original,

foi substituída por três unidades, com funções distintas, designadas por CPU1, CPU2 e CPU3.

3.1.1 CPU1- Mini PC GIGABYTE

O CPU1 é a unidade principal de processamento, uma vez que é aqui que todos os processos `ROS` são executados e processados. Devido a uma sobrecarga de energia, esta unidade necessitou de ser substituída por outra semelhante, e deste modo os sistemas operativos foram consequentemente atualizados para versões mais recentes, o `LINUX` e o `ROS`. Trata-se de um mini-PC GIGABYTE GB-BKi7A-7500, apresentado na figura 3.2, com as características referidas na tabela 3.1 .

Figura 3.2: CPU1: Mini-PC GIGABYTE GB-BKi7A-7500 [30].

Tabela 3.1: Características do CPU1 [30].

Especificações	Detalhes
Chipset	Intel® H110
Processador	Intel® Core™ i7-7500U Dual-Core, 2.7 GHz with Turbo up to 3.5 GHz
Armazenamento	SSD 2.5" Kingston V300 120 GB MLC SATA
Memória	RAM SO-DIMM Crucial Ballistic 8GB DDR4-2400MHz
Gráficos	Intel® HD Graphics 620
Network	Intel® Dual Band Wireless-AC 3168 (M.2 2230)
	Realtek ALC255
Alimentação	19V input, 65W
Dimensões	119.4 X 112.6 X 34.4 [mm]

3.1.2 CPU2- Arduino Leonardo ETH

O CPU2, figura 3.3, é responsável pelo controlo da velocidade da plataforma. Trata-se de um micro-controlador *Arduino Leonardo ETH* e as suas especificações estão presentes na tabela 3.2.

Figura 3.3: CPU2: *Arduíno Leonardo ETH* [31].

Tabela 3.2: Características do CPU2 [22].

Especificações	Detalhes
Processador	802.3 10/100 Mbit/s
Micro-controlador	ATmega32u4
Arquitetura	AVR
Tensão de Operação	5V
Memória flash	32 KB das quais 4 KB usadas pelo *bootloader*
SRAM	2.5Kb
Velocidade do Relógio	16 MHz
Pinos Analógicos I/O	12
Pinos Digitais I/O	20
Canais PWM	7
EEPROM	1 KB
Interface	RS232, SPI, I^2C, Ethernet

3.1.3 CPU3- Arduino Micro

Uma vez que o CPU2 não tem a capacidade de resposta requerida, foi necessário adicionar um CPU3 de modo a contar os pulsos provenientes dos *encoders*, e enviá-los para o CPU2 a uma taxa de 20 Hz [20]. Para o efeito foi escolhido um Arduíno Micro (figura 3.4) com as especificações presentes na tabela 3.3.

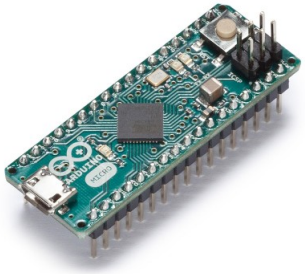


Figura 3.4: CPU3: *Arduíno Micro* [32].

Tabela 3.3: Características do CPU3 [32].

Especificações	Detalhes
Micro-controlador	ATmega32U4
Tensão de Operação	5V
Memória flash	32 KB das quais 4 KB usadas pelo *bootloader*
SRAM	2.5Kb
Velocidade do Relógio	16 MHz
Pinos Analógicos I/O	12
Pinos Digitais I/O	20
Canais PWM	7
EEPROM	1 KB
Interface	RS232, SPI, I^2C

3.1.4 Lasers Hokuyo

Em cima da plataforma, existem dois lasers *rangefinder* (LRF) de modo a auxiliar a navegação do ROBONUC.

O laser dianteiro, é um laser *Hokuyo UTM-30LX* (figura 3.5), o laser traseiro um *Hokuyo URG-04LX-UG01* (figura 3.6), com as características apresentadas nas tabelas 3.4 e 3.5, respetivamente.

Figura 3.5: *Hokuyo UTM-30LX* [33].

Tabela 3.4: Características do laser Hokuyo UTM-30LX [33].

Especificações	Detalhes
Fonte de Alimentação	12VCC±10%
Fonte Luminosa	Díodo laser semicondutor (λ=905nm), classe 1 de segurança
Área de Medição	0.1 a 30m, 270^o
Precisão	0.1 a 10m: ±30mm
	10 a 30mm: ±50mm
Resolução Angular	Ângulo de passo: ≈0.25^o
Tempo de Leitura	25ms/scan

Figura 3.6: *Hokuyo URG-04LX-UG01* [34].

Tabela 3.5: Características do laser Hokuyo URG-04LX-UG01 [34].

Especificações	Detalhes
Fonte de Alimentação	5VCC±5%
Fonte Luminosa	Díodo laser semicondutor (λ=785nm), classe 1 de segurança
Área de Medição	20 a 5600mm, 240^o
Precisão	60 a 1,000mm: ±30mm
	1,000 a 4,095mm: ±3%
Resolução Angular	Ângulo de passo: ≈0.36^o
Tempo de Leitura	100ms/scan

3.1.5 Controlador XBox

A plataforma pode ser controlada através do comando XBox 360, apresentado na figura 3.7, que tem um alcance máximo de 9 m. Este dispositivo permite ao utilizador uma rápida adaptação e, como já verificado em [22] e [20], um controlo preciso.

Figura 3.7: *GamePad Microsoft XBox 360* [35].

3.1.6 Router Asus

De modo a estabelecer comunicação entre CPUs, a plataforma contém um *router* (figura 3.8), que permite também o seu controlo remotamente por Secure Shell (SSH) através de uma rede wi-fi.

Figura 3.8: *Router Asus RT-AC51U* [36].

3.1.7 Arquitetura de comunicação dos sistemas apresentados

A figura 3.9 apresenta a arquitetura de comunicação dos sistemas descritos. Como se pode observar o CPU1 estabelece ligação via Ethernet com o CPU2, e entre o CPU2 e o CPU3 é utilizado o protocolo I^2C.

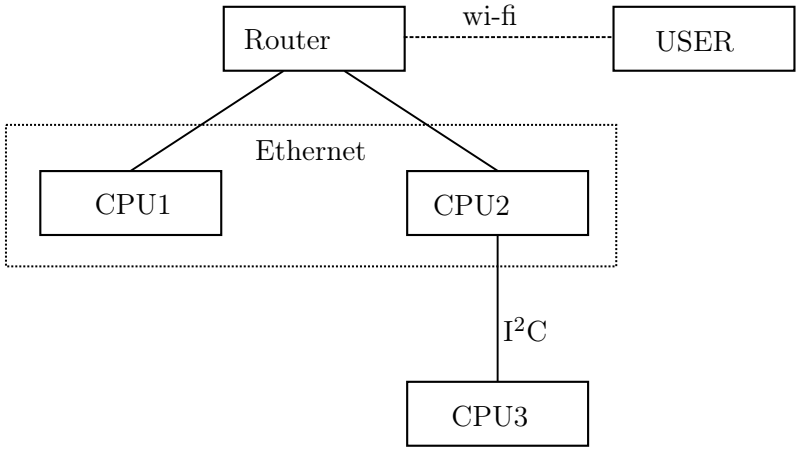


Figura 3.9: Diagrama de comunicações da plataforma móvel [22].

3.2 Manipulador Róbotico

3.2.1 Robô FANUC LR Mate 200iD

O ROBONUC apresenta um braço robótico, FANUC LR Mate 200iD presente na figura 3.10. Este manipulador dispõe de um alcance de 717 mm, semelhante ao de um braço humano. Tem uma capacidade de carga de 7 kg, capaz de atingir velocidades de 4 ms^{-1} e uma repetibilidade de ± 0.018 mm. Apresenta um peso de apenas 25 kg, o que facilita a sua integração com a plataforma móvel [37].

Figura 3.10: *FANUC LR Mate 200iD* [37].

3.2.2 Sensor Kinect

Para a perceção de objetos e obstáculos durante tarefas de manipulação, o robô *Fanuc* tem acoplado ao seu quarto elo um sensor *Kinect*, figura 3.11. Este dispositivo é capaz de combinar informação de cor (RGB) com pixeis de profundidade. Contém uma câmara RGB, um sensor de profundidade, um microfone, e um acelerómetro de 3 eixos [38]. Encontram-se na tabela 3.6 as especificações mais relevantes para o trabalho.

Tabela 3.6: Características da *Microsoft Kinect* [38].

Especificações	Detalhes
Sensor de profundidade	min 800mm e max 4000mm
Ângulo de visão	43º vertical por 57º horizontal
Inclinação vertical	±27º
FPS	30 FPS
Resolução de profundidade	640x480, 320x240 ou 80x60
Resolução de cor	1280x960 a 12FPS, 640x480 a 15FPS ou 640x480 a 30 FPS

Figura 3.11: *Micosoft Kinect* [38].

3.2.3 Sensor Laser DT20Hi, CPU4 e CPU5

De modo a melhorar a precisão no processo de *picking* de objetos, foi acoplado junto do *gripper* do manipulador, um sensor laser de precisão. Este sensor, *SICK DT20HI* presente na imagem 3.12, é um sensor 1D que permite medir distâncias com precisão sem recorrer a contacto, independente da rugosidade do alvo. A tabela 3.7 apresenta as suas especificações.

Figura 3.12: *SICK DT20HI* [39].

Para adquirir, de modo mais preciso, a informação proveniente do sensor laser, concebeu-se (em [13]) um circuito para condicionar e converter o sinal. Assim, para o seu processamento escolheu-se um Arduíno UNO (CPU4), que após a calibração do laser, filtra o sinal analógico e envia a medição por porta série. Esta unidade de processamento pode ser visualizada na figura 3.13.

Salienta-se também, que de modo a controlar os IO's do manipulador FANUC, foi necessário conceber (em [21]) um circuito externo com um outro Arduíno UNO (CPU5).

Tabela 3.7: Características do sensor *SICK DT20HI* [39].

Especificações	Detalhes
Alcance de medição	100-1000 mm
Resolução	1000 μm
Linearidade	$\pm$6 mm
Tempo de resposta	2,5ms ; 10 ms ou 40ms (configurável)
Alimentação	10-30V
Saída analógica	4-20mA
Resolução analógica de saída	12bit

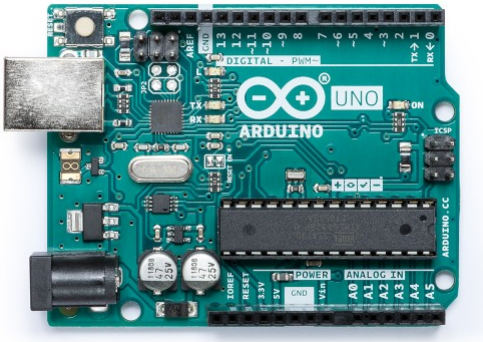


Figura 3.13: Arduíno UNO [40].

3.2.4 Arquitetura dos sistemas apresentados

Para que se perceba melhor a interação dos componentes necessários à tarefa de *Bin-Picking*, apresenta-se na figura 3.14 um esquema da arquitetura presente e as suas comunicações.

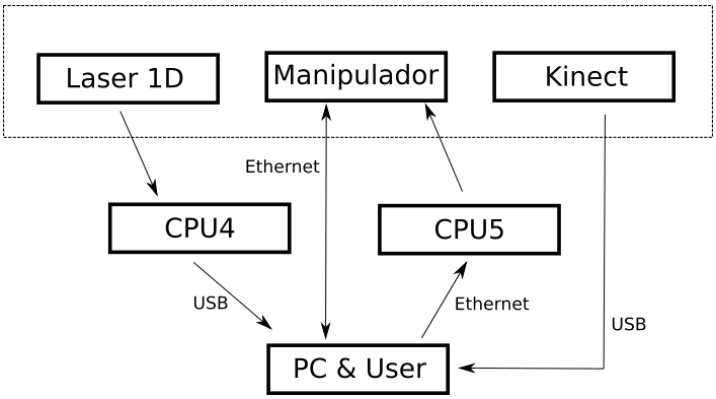


Figura 3.14: Arquitetura dos componentes do manipulador e respetivas comunicações.

3.3 Integração do Hardware

Para um maior proveito do equipamento desenvolvido, foi necessário adotar estratégias que permitem o seu bom desempenho. Assim, para reduzir a dependência energética do manipulador móvel, foi instalado um inversor de tensão na plataforma.

Uma vez que à saída das baterias da plataforma móvel existe uma diferença de potencial de 48 V, o inversor escolhido teria obrigatoriamente de aceitar essa tensão como entrada.

Sabendo que ${P_{aparente}}^2 = {P_{ativa}}^2 + {P_{reativa}}^2$ e assumindo a Potência Reativa como nula, uma vez que se usou o fator de potência mais desfavorável (igual a 1), resulta em $P_{aparente} = P_{ativa}$. Desta forma, a potência ativa em W, é dada pelo produto entre a voltagem e a amperagem consumida de cada equipamento.

Na tabela 3.8 são apresentadas as tensões, correntes e potências que cada dispositivo consome. Após um somatório de todas as potências individuais, o sistema necessita de uma potência total de aproximadamente 1330 W.

Tabela 3.8: Tabela representativa dos cálculos de potência.

Equipamento	**Voltagem (V)**	**Amperagem (A)**	**Potência (W)**
Kinect	12	1,08	12,96
Sensor Laser 1D	12	1	12
Hokuyo UTM-30LX	12	1	12
Mini-Pc	19	3,42	64,98
Fanuc	200	6	1200
Hub	12	2	24
Total			1325,94

Possivelmente um inversor de 1500 W poderia assegurar as necessidades energéticas, contudo, salienta-se que este tipo de equipamentos podem ter oscilações de ±10 %. Assim, a potência fornecida poderia variar entre 1350 e 1650 W, o que representaria um limite muito próximo da potência necessária. Posto isto, optou-se por um inversor de 2000 W de forma a que em projetos futuros o sistema tenha capacidade suficiente para integrar novos equipamentos. O inversor escolhido pode ser visualizado na figura 3.15 e contêm as características apresentadas na tabela 3.9,

Figura 3.15: Inversor de tensão (DC-AC) instalado [41].

Tabela 3.9: Características do Inversor 48 V (DC-AC) [41].

Especificações	Detalhes
Modelo	179-3339
Potência nominal	2000 W
Tensão de entrada	48 V
Tensão de saída	230 V
Frequência de saída	50 Hz ±3
Onda de saída	Onda sinusoidal pura
Dimensões	372 × 230 × 108 mm
Peso	4850 g

Uma vez que o computador do robô (CPU1) não tem capacidade de efetuar todas as ligações USB necessárias com os equipamentos presentes, foi necessário instalar um *HUB USB*. Optou-se pelo *HUB* apresentado na figura 3.16 que contêm 7 portas *USB* 3.0 e alimentação externa de forma a responder às necessidades energéticas de todos os equipamentos.

Figura 3.16: *HUB* UH700 [42].

Desta forma, foi possível a sua integração física do sistema, isto é, a instalação de todos os equipamentos na mesma plataforma (conforme representado na figura 3.17). É de salientar que para o uso da corrente alternada fornecida pelo inversor, foi necessária a instalação de transformadores de tensão (230AC para 12DC) para o laser dianteiro e para o sensor laser 1D.

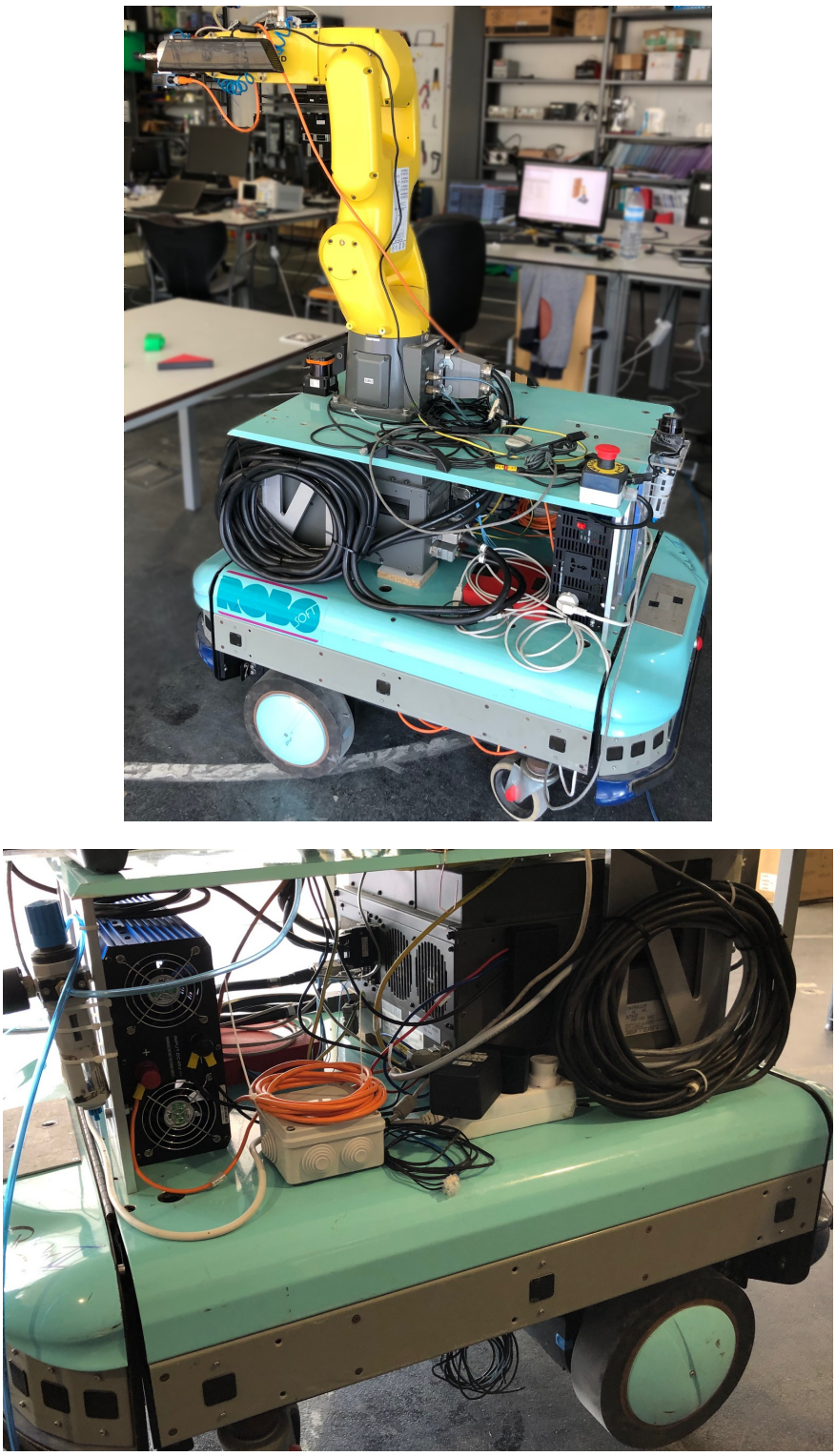


Figura 3.17: Representação do sistema após montagem de todo o *hardware* necessário.

Capítulo 4

Infraestrutura Experimental: Software

De modo a desenvolver o projeto é necessário compreender as ferramentas aplicadas a cada um dos sistemas, para uma posterior integração eficaz. Neste capitulo introduz-se e descreve-se todo o software inerente aos subsistemas referidos anteriormente.

4.1 ROS - Robot Operating System

A complexidade do desenvolvimento de ferramentas robustas para a robótica, necessita de diferentes equipamentos, com diferentes comunicações e por vezes programas em diferentes linguagens, o que os tornam difíceis de escrever e monitorizar [43]. Deste modo, surgiu o `ROS`, uma plataforma de desenvolvimento de código aberto, que combina todas as bibliotecas, ferramentas e convenções com o objetivo de simplificar a tarefa de desenvolvimento para diversas aplicações robóticas [44].

Este sistema operativo providencia uma gestão de pacotes de modo a simplificar e a incentivar a reutilização de código, de forma a que seja possível construir rapidamente um software que inclua funcionalidades sofisticadas, tais como planeamento de trajetórias ou reconhecimento de objetos, sem que seja necessário ter conhecimentos profundos na área de desenvolvimento de *software*. Devido à sua arquitetura modular, existe a possibilidade da reutilização de módulos e bibliotecas de outros projetos na realização de novos, o que contribui significativamente para um melhor tempo e eficácia de desenvolvimento. Padrões mecatrónicos para *AIMM*, ou seja, padrões de interface de software e hardware, são importantes não só em termos de desenvolvimento e integração, mas também no *benchmarking* do sistema.

Como existem diversas distribuições de `ROS`, optou-se por utilizar a versão mais recente, `ROS Melodic Morenia` (figura 4.1), apesar das ferramentas até agora concebidas para o ROBONUC serem projetadas numa versão mais antiga. No entanto, esta migração deve ser realizada.

De modo a evitar paragens de funcionamento, o `ROS` adota uma filosofia descentralizada em que, caso alguma `thread` do código principal deixar de funcionar, não impede o contínuo funcionamento dos restantes processos.

Existem quatro conceitos fundamentais que necessitam de ser clarificados para a entender o funcionamento do `ROS`.

Figura 4.1: Logótipo `ROS Melodic Morenia` [45].

Nós: São processos ou executáveis que "correm" para uma certa aplicação. É comum a existência de diversos nós em simultâneo, de modo a que cada um tenha uma tarefa específica. Existe troca de informação síncrona ou assíncrona, e cada nó pode comunicar com outros através de tópicos, serviços ou parâmetros.

Mensagens: São a forma mais usada na comunicação entre nós. As mensagens são publicadas em tópicos e podem ser lidas por outros nós. De modo a que as mensagens sejam reconhecidas por todos os nós, independentemente da linguagem de programação usada, o programador pode usar tipos de mensagens *standard*, ou definir outros tipos mais adequados a uma determinada aplicação. Assim, as mensagens podem resultar em combinações de strings, variáveis booleanas, inteiras, flutuantes, ou até estruturas.

Tópicos: São os "canais"onde são publicadas as mensagens pelos nós. Designa-se um nó publicador, quando este envia uma mensagem através de um certo tópico, e um nó subscritor quando este lê uma mensagem de um tópico. Esta comunicação não é limitada quanto ao número de nós, ou seja, podem existir vários nós publicadores bem como subscritores do mesmo tópico. Para que estas comunicações (unidirecionais) se concretizem, o tópico necessita de estar associado ao mesmo tipo de mensagem que os nós publicam e leem.

Serviços: Providenciam um tipo de comunicação síncrona de pedido-resposta, que é definida por um par de mensagens, uma de pedido e outra de resposta. Um nó pode oferecer um serviço, e um nó cliente pode chamar esse serviço enviando uma mensagem e esperando pela sua resposta.

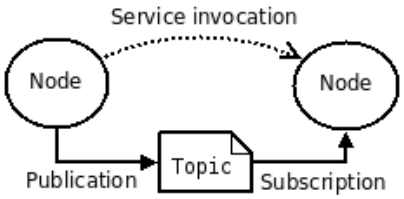


Figura 4.2: Conceitos básicos inerentes ao `ROS` [46].

O conceito de publicador-subscritor vem dar outra versatilidade às aplicações desenvolvidas em `ROS` uma vez que o publicador não necessita de especificar quem vai receber as mensagens, apenas publica as mensagens num tópico e quem pretender essa informação pode consultá-la. Deste modo, o modelo usado permite uma escalabilidade do sistema, muito mais facilmente do que o paradigma cliente-servidor [47].

Ações

As ações são semelhantes aos serviços, uma vez que permitem um pedido para obter uma resposta. Existem três especificações das ações, o *Goal*, o *Feedback* e o *Result*, ou seja um pedido que é definido pelo *Goal*, uma motorização da ação através do *Feedback*, e um resultado final após a conclusão.

Apesar desta filosofia geral de funcionamento, existem dois tipos de clientes e servidores, os simples e os "não-simples". Podemos optar por criar um `SimpleActionServer` ou um `ActionServer`, em que a principal diferença reside no facto de num servidor simples apenas um *Goal* pode ter um estado ativo em cada momento, colocando todos os outros pedidos em espera. Caso aceite um novo *Goal* enquanto outro esteja a decorrer, a ação será publicada com sucesso mesmo não tendo sido executada até ao fim. Um servidor "não-simples" pode ter vários estados, como é demonstrado na figura 4.3.

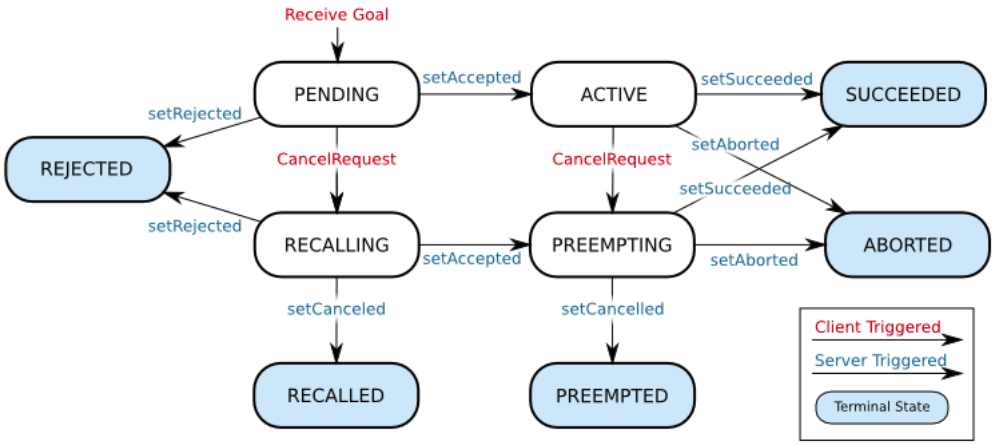


Figura 4.3: Estado de transições de um servidor de ações [48].

4.2 Plataforma Móvel

Os problemas de navegação na robótica móvel podem ser sumariados com base em três perguntas: "onde estou?", "onde vou?" e "como devo lá chegar?" [49]. A primeira questão remete para saber a sua localização atual com base no que "vê" e no que já viu anteriormente. A segunda e terceira questão são essencialmente questões de especificações de objetivos e de planeamento de trajetórias para atingir o local desejado.

4.2.1 Cinemática

Como já foi referido, a locomoção da plataforma móvel é realizada pelas duas rodas dianteiras. Como cada roda possui um motor acoplado, é possível realizar o controlo individual da velocidade de cada roda e caso se deseje um movimento curvilíneo basta conferir velocidades distintas a cada uma das rodas. Na figura 4.4 são apresentadas as variáveis consideradas para os cálculos, das quais V_r e V_l designam as velocidades lineares de cada uma das rodas, R o raio de casa Roda e L a largura da plataforma.

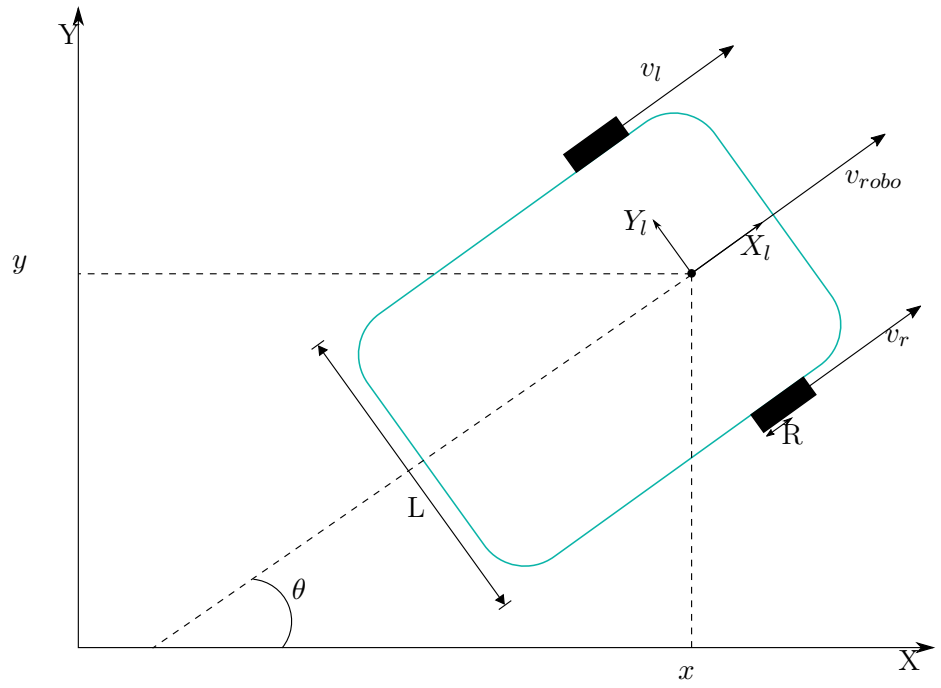


Figura 4.4: Cinemática do robô com locomoção diferencial [50].

Assim, com base nas deduções em [22] pode-se definir a variação da posição, em relação ao referencial global, com o sistema de equações 4.2, em que $\Delta x_l, \Delta y_l, \Delta\theta_l$ estão definidos em 4.1.

$$\begin{cases} \Delta x_l = \frac{R}{2}(v_r + v_l) \\ \Delta y_l = 0 \\ \Delta\theta_l = \frac{R}{L}(v_r + v_l) \end{cases} \tag{4.1}$$

$$\begin{cases} \Delta x = \Delta x_l \cdot \cos(\theta) - \Delta y_l \cdot \sin(\theta) \\ \Delta y = \Delta x_l \cdot \sin(\theta) + \Delta y_l \cdot \cos(\theta) \\ \Delta\theta = \Delta\theta_l \end{cases} \tag{4.2}$$

4.2.2 Localização

Hodometria

Das diversas técnicas existentes para efetuar a localização de um robô e tendo em conta os equipamentos instalados na plataforma, *encoders* e lasers, optou-se por explorar duas técnicas bastante conhecidas, a hodometria e a correspondência de modelos.

Fundamentalmente, a hodometria consiste na integração de informação incremental ao longo do tempo, permitindo a determinação do deslocamento e velocidade das rodas. Como se pode observar na figura 4.5, os pulsos provenientes dos *encoders* são contados pelo CPU3 (arduíno) para posteriormente, enviar essas leituras para a unidade principal de processamento (CPU1). Com base nesta informação é possível calcular a hodometria

através das equações 4.1 e 4.2. Uma vez que o ROS fornece ferramentas para o cálculo e publicação da hodometria, foi criado um nó (odom_node) de forma a subscrever os dados provenientes do arduíno e a publicar a hodometria. Esta técnica de hodometria, como não previne eventuais deslizamentos das rodas e os erros são comutativos, seria mais adequada para localização em pequenas distância. Deste modo, para combater estas incorreções, o nó referido, subscreve as transformadas provenientes pelo mapeamento e corrige a sua posição quando a transformada da hodometria se distância da transformada do mapeamento.

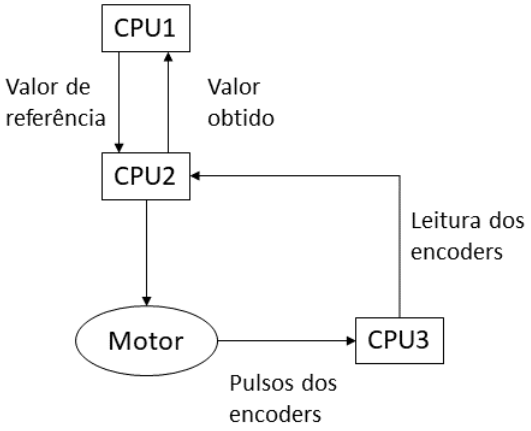


Figura 4.5: Diagrama de leitura dos *encoders* [22].

Mapeamento

Tendo em conta os objetivos do projeto, optou-se por apenas explorar a criação de um mapa em tempo real, uma vez que o robô seria semi-autónomo e não completamente autónomo. Assim, para facilitar a navegação por parte do utilizador o principal objetivo do mapeamento é conhecer os obstáculos do meio envolvente.

Para efetuar o mapeamento é usado um sistema LIDAR (*Light Detection and Ranging*). Uma vez que se usam dois lasers Hokuyo para o efeito, existe um pacote ROS, hokuyo_node [51], que permite realizar a leitura de ambos os lasers. Deste modo, como cada um dos lasers cria o seu próprio mapa, foi necessário utilizar um pacote ROS, ira_laser_tools, que possibilita a fusão a informação de vários lasers. Para o seu uso basta indicar a transformada de um sistema de coordenadas laser para outro, e o mapa resultante será dado no sistema de coordenadas do laser definido como "pai".

Das diferentes técnicas disponíveis em ROS, para efetuar o mapeamento utilizou-se o HectorSLAM [52], uma vez que, para sua aplicação é recomendado utilizar um *Laser Rangefinder* (LRF) de alta frequência (como é o caso do laser UTM-30LX presente na zona frontal da plataforma) e não necessita dos dados da hodometria contrariamente a outras técnicas. Para implementação do HectorSLAM é necessário definir alguns sistemas de coordenadas bidimensionais, como apresentados na figura 4.6. Nessa conformidade, para simplificar sistemas coordenadas definiu-se que os referenciais base_footprint e laser_link se encontram sobrepostos. Quando a plataforma se desloca, o HectorSLAM utiliza a infor-

mação adquirida e compara-a com informação antiga de modo a encontrar semelhanças e a estimar a posição. Esta técnica apresenta algumas limitações comprovadas em [22], nomeadamente em corredores onde os lasers só detetam as paredes laterais, bem como em espaços simétricos ou envidraçados.

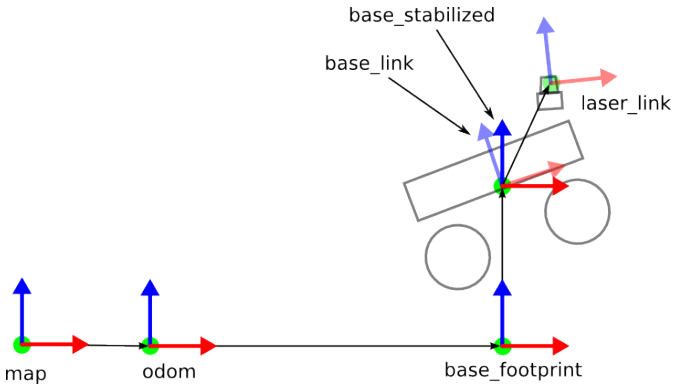


Figura 4.6: Sistemas de coordenadas necessários para o `HectorSLAM` [22].

O facto de usar a informação fundida pelo pacote `ira_laser_tools` definido num só sistema de coordenadas laser, leva a uma interpretação incorreta por parte do `HectorSLAM`. Deste modo apenas foi utilizado a informação proveniente do laser dianteiro para realizar o mapeamento, uma vez que este apresenta informação mais fiável face ao laser traseiro. Alternativamente poderia optar-se por criar dois mapas baseados em cada um dos lasers, e posteriormente fundir essa informação. É de salientar, que esta abordagem não foi utilizada devido à necessidade de maior exigência computacional, e da diferença de precisão entre os dois lasers, uma vez que o laser traseiro poderia dificultar a compreensão do mapa devido ao elevado ruído.

4.2.3 Navegação Assistida

A plataforma móvel apresenta dois modos de navegação, um completamente autónomo e outro semi-manual.

Modo Automático

O primeiro modo, automático, foi implementado usando o pacote `teb_local_planner` [53], complementar ao pacote `navigation` [54]. Para gerar uma trajetória inicial até uma certa posição objetivo, o pacote `navigation` necessita de informações relativas à hodometria bem como dados laser e respetivas transformadas (geométricas). Esta trajetória é otimizada pelo `teb_local_planner` de modo a manter distâncias de segurança aos obstáculos e minimizar o tempo de execução, garantindo as limitações cinemáticas da plataforma, como por exemplo velocidade máxima e aceleração. Uma vez que para a utilização destes pacotes, é necessário fornecer um mapa prévio do local de navegação, recorreu-se ao pacote `HectorSlam`. Este pacote permite a criação de um mapa em tempo real à medida que a plataforma se move, e onde o espaço desconhecido, neste projeto, é assumido como espaço livre.

Na figura 4.7 pode-se ver o mapa construído pelo `HectorSlam` e a trajetória calculada pelo modo automático a fim de atingir o ponto e orientação pedida.

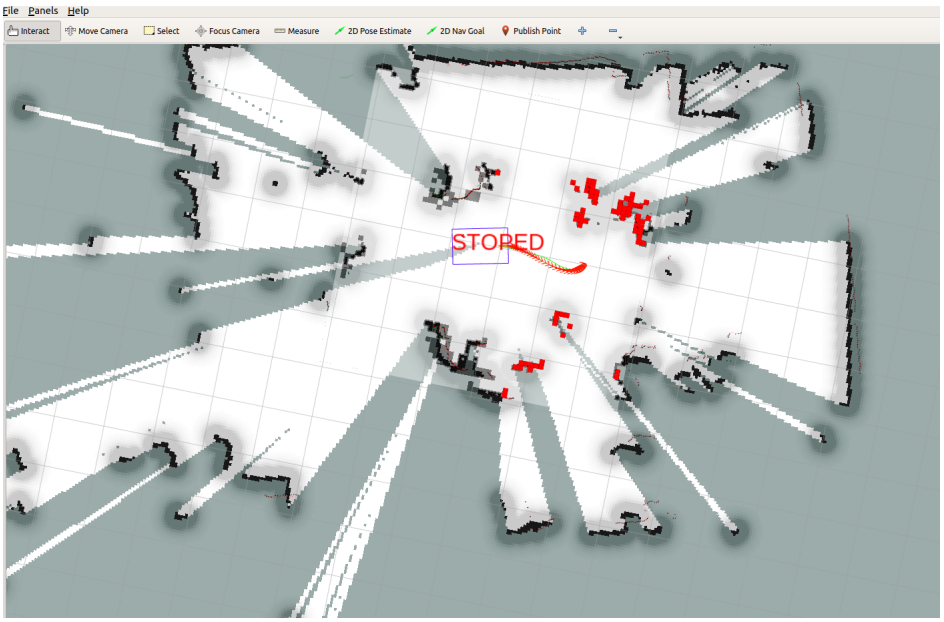


Figura 4.7: Trajetória calculada pelo modo automático.

O desempenho do uso destas ferramentas para o mapeamento está dependente da capacidade de processamento do computador, velocidades angulares e inversões de velocidades. Como se pode concluir em [22], deve-se assim, evitar inversões repentinas de velocidade e mudanças de direção a velocidades angulares elevadas.

Modo Semi-Automático

O modo semi-automático, ou semi-manual, consiste na integração do modo de navegação autónoma quando a plataforma é controlada manualmente, ou seja, caso exista risco de colisão por parte do operador, o modo automático será ativado.

De modo a perceber o momento de comutação do modo manual para semi-automático deve-se ter em conta a distância de paragem e as zonas de risco consideradas. A distância de paragem é o espaço percorrido pela plataforma desde que deteta um obstáculo até se imobilizar. Assim sendo, esta distância é função da sua velocidade linear, ou em caso de movimentos circulares, da velocidade angular.

As zonas de risco são zonas geométricas da plataforma que previnem colisões com os obstáculos do meio envolvente. Para o efeito, foram adotadas zonas de geometrias simples, de fácil aplicação, que dependem da velocidade do movimento. Assim, em locais mais estreitos, como portas, a plataforma é obrigada a movimentar-se a baixa velocidade.

Em suma, no modo semi-automático, o utilizador indica a direção em que pretende deslocar a plataforma com o comando *XBox* e um nó com a função de árbitro, `r_hybrid`, avalia se o movimento é seguro tendo em conta os dados provenientes dos lasers. Caso este nó considere a manobra perigosa a velocidade passa a ser definida pelo o nó que controla o modo automático.

Para melhor compreensão apresenta-se, na figura 4.8 o diagrama de funcionamento do modo semi-automático.

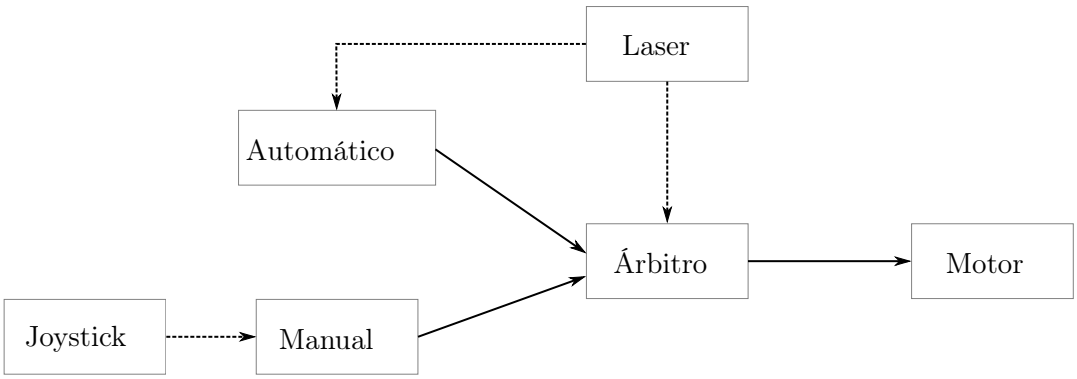


Figura 4.8: Diagrama de funcionamento do modo semi-automático [22].

4.2.4 Resumo da arquitetura total

Para uma melhor perceção da arquitetura inerente às funcionalidades apresentadas, apresentar-se-á o diagrama total, bem como a marcação dos sub-diagramas para cada função. O diagrama de funcionamento do modo automático e semi-automático pode ser visualizado na figura 4.9 ou no anexo B (em maior dimensão).

O bloco numerado como 1, na figura 4.9 contém os nós e tópicos necessários ao modo automático, do pacote `navigation`. O nó principal deste modo, `move_base`, subscreve as informações dos lasers através do tópico `/scan`, o mapa através do tópico `/map` e da localização do robô através do tópico `/tf`. Deste modo, quando existe uma posição alvo indicada pelo utilizador, este nó envia as velocidades que a plataforma deve ir tomando para atingir a posição desejada, para o tópico `/cmd_vel`.

O bloco 2 representa as leituras dos lasers, ou seja o nó `hokuyo_node` está a receber as informações do hardware e a publicá-las em linguagem `ROS` para o tópico `/scan`.

O bloco 3 está associado às leituras do comando XBox. O nó `joy_node` recebe os sinais do comando, e publica essa informação no tópico `/joy`.

Desta forma, o nó `r_hybrid` subscreve os dados provenientes do comando e do modo automático, decidindo qual o modo que prevalece. Após essa decisão, este nó publica as mensagens da velocidade (correspondente ao modo em execução) no tópico `/navi_comands`.

O bloco 4, responsável pelo controlo da velocidade da plataforma, recebe as velocidades pretendidas pelo nó `r_hybrid` e envia-as para o hardware. A informação devolvida pelo tópico `/pid_data` é subscrita pelo nó `/odom_node`, que calcula a hodometria.

Este nó pertence ao bloco 5 e é responsável pelo registo dos referenciais da plataforma no tópico `/tf`. Estes referenciais permitem estabelecer a localização relativa a um referencial global, localizado no ponto onde o sistema iniciou.

O bloco 6 é responsável pelo construção do mapa em tempo real, sendo necessário as informações dos lasers através do tópico `/scan` e das transformações existentes entre referenciais. Desta forma, o nó `/hector_mapping` publica o mapa no tópico `/map` para que este seja utilizado pelo modo automático (bloco 1).

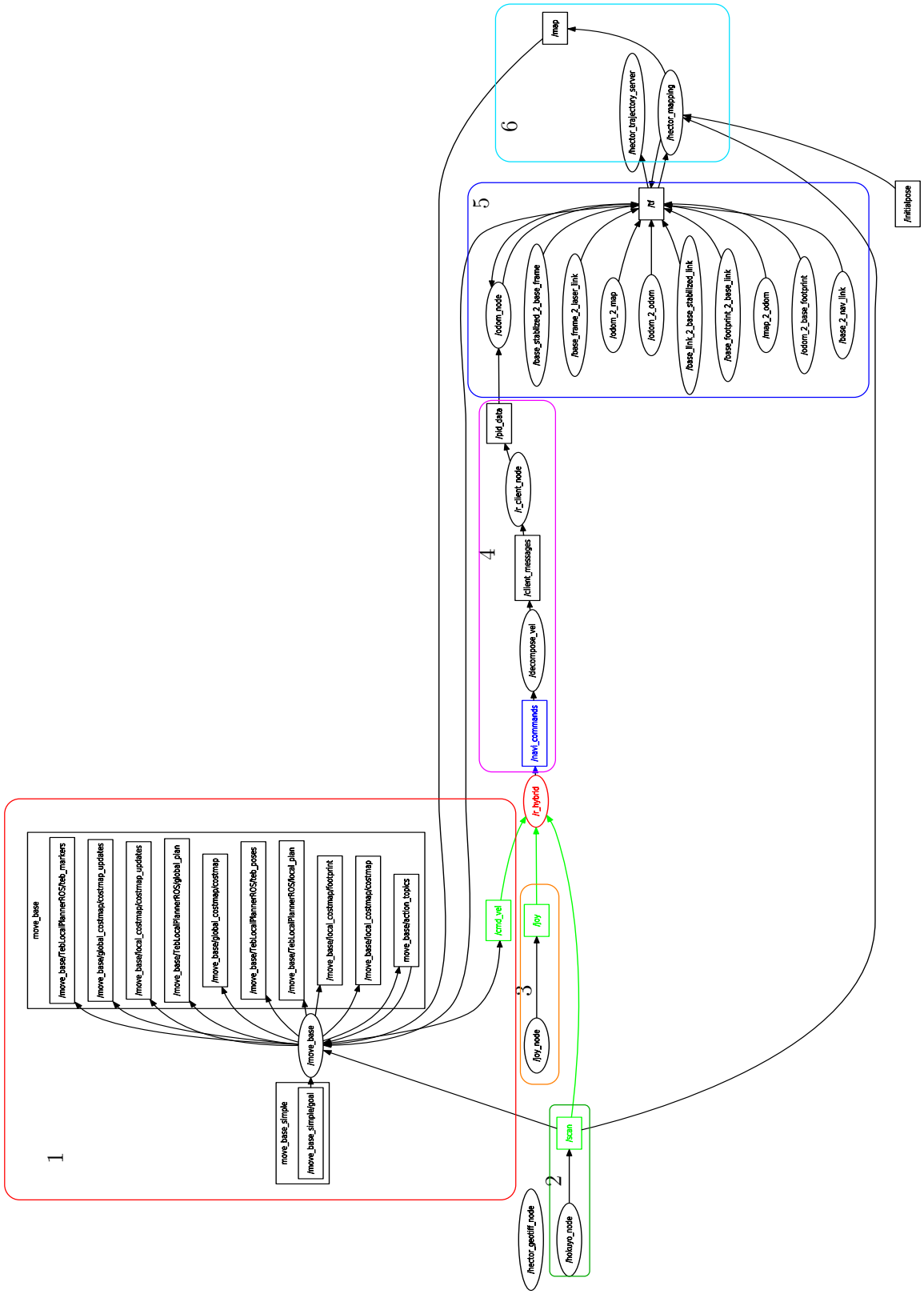


Figura 4.9: Diagrama de funcionamento do modo automático e semi-automático [22].

4.3 Manipulador Robótico para Bin-Picking

De modo a executar um processo de *bin-picking*, é necessário realizar várias etapas de forma a obter os centróides e as normais dos objetos presentes e assim proceder ao *grasping*. Para o efeito, utilizou-se a seguinte metodologia [13]:

1. Remoção de pontos indesejados da nuvem de pontos adquirida, como por exemplo os pontos no chão ou pontos da base do robô;
2. Remoção dos pontos da mesa onde os objetos estão sobrepostos.
3. Agrupamento (*clustering*) da nuvem de pontos para separar os diferentes objetos;
4. Determinação do centróide e da normal de cada objeto.

4.3.1 Segmentação da Nuvem de Pontos

O nó `objDetection` é responsável por subscrever um dos tópicos publicados pela kinect, `camera/depth_registered/points`, e após o tratamento publica as coordenadas de um centróide e a normal da sua superfície, no tópico `cloud_centroid` e `cloud_centroid_normal`, respetivamente. A fim de realizar esse tratamento, salienta-se que para a aquisição de imagem é necessário a instalação do *software* `Open Natural Iteraction (OpenNI)` [55], e para facilitar a segmentação desejada ferramentas da `Point Cloud Library (PCL)` [56] demonstraram-se bastante úteis.

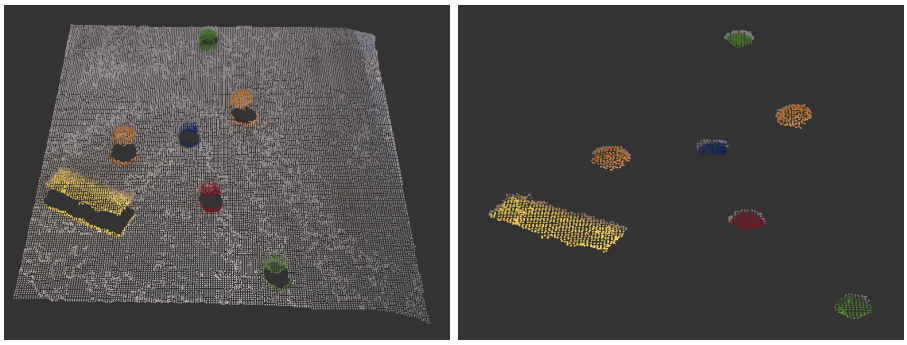
Para remover os pontos indesejados, não pertencentes à mesa, utilizou-se um filtro de profundidade denominado por *PassThrough* que remove os pontos distanciados a mais de 0.9 m, ao longo do eixo Z, da câmara.

Após esta remoção, constatou-se que, apesar da diminuição de alguns pontos, a nuvem adquirida continuava pesada computaconalmente. Para salvar algum espaço no disco, e flexibilizar o processo, reduziu-se o número de pontos através de um filtro `VoxelGrid`, que aceita como entrada as dimensões do cubo que servirá de filtro. Para este caso escolheu-se um filtro com uma geometria cúbica, com aresta de 2 mm, e o resultado pode ser visualizado na figura 4.10a.

O passo seguinte consiste em identificar e remover o fundo correspondente à superfície da mesa. Para esse fim, utilizou-se a função `SACSegmentation` que encontra um plano na cena adquirida. Apesar deste método ser bastante eficaz, após a sua aplicação foi necessário utilizar o filtro `RadiusOutlierRemoval` de modo a remover alguns pontos isolados ainda presentes. Desta forma, pontos que não contenham pelo menos 45 pontos vizinhos num raio de 20 mm serão eliminados.

Tendo apenas os pontos pertencentes às superfícies dos objetos, é necessário dividi-los em diferentes nuvens de pontos (figura 4.10b), com o intuito de conseguir processar cada superfície individualmente. Deste modo, utilizou-se o método `Eucledian Cluster Ectraction`, que identifica diferentes superfícies na cena caso os objetos não se encontrem em contacto entre eles. Para sua parametrização, definiu-se que os objetos teriam que ter uma distância de tolerância de cerca de 35 mm, e que o número de pontos da respetiva nuvem de cada objeto, estaria contido entre 100 e 2500.

O passo final da segmentação consiste na determinação do centróide e da normal de cada objeto. A classe `compute3DCentroid` estima o valor das coordenadas do centróide (X, Y e Z) retornando-as na forma de um vetor 3D, possibilitando assim a determinação



(a) Nuvem após o filtro `VoxelGrid` [13]. (b) Objetos segmentados [13].

Figura 4.10: Nuvens de pontos pós-tratamento.

do índice do ponto, no respetivo *cluster*. Uma vez que a classe `NormalEstimation` retorna os vetores perpendiculares aos pontos da superfície, o conhecimento do índice do centróide do objeto torna-se imprescindível para a associação do centróide ao seu vetor normal.

O nó `pointTFtransfer`, subscreve os tópicos `/cloud_centroid_normal` e `/cloud_centroid`, que contêm informação do centróide e do vetor normal de um objeto, vistos no referencial da câmara, mais especificamente no `camera_rgb_optical_frame`. Assim sendo, este nó tem como função calcular os pontos de aproximação do *end effector*, vistos no referencial do manipulador, `robot_base_link`.

Existem 2 pontos de aproximação necessários, um para medição com o laser 1D e outro para aproximação do *"picking"*, designados na figura 4.11 como "aprox_point1" e "aprox_point2", respetivamente. O segundo ponto é calculado com base na equação 4.3, em que A_p é o ponto de aproximação, C representa as coordenadas do centróide, $\vec{n}$ o vetor normal à superfície, e k simboliza a distância de um ponto ao outro, definido como 300 mm.

$$A_p = C + k \times \vec{n} \tag{4.3}$$

Uma vez que a orientação da ponta do robô nesse ponto necessita de ter uma direção contrária à normal da superfície, basta apenas calcular o vector $\overrightarrow{A_pC}$, em que $\overrightarrow{A_pC} = (x_C - x_{A_p}, y_C - y_{A_p}, z_C - z_{A_p})$ e normalizá-lo. Para descrever a orientação da ponta do manipulador, basta calcular os ângulos de *Euler* (*Roll, Pitch e Yaw*) com base no vetor $\overrightarrow{A_pC}$ já conhecido. Salienta-se que para o ângulo *Roll* definiu-se um valor fixo de 180°, dado que o nosso robô usa uma ventosa para a apanha de objetos, este ângulo pode tomar qualquer valor, uma vez que representa a rotação em torno do eixo Z do *"end effector"*.

Para a leitura da distância do sensor laser, o seu referencial deve ser coincidente com o ponto de aproximação calculado anteriormente. Desta forma é necessário calcular a posição do *end effector* para que o laser execute o processo de medição nesse ponto. Para o efeito basta apenas fazer uso do pacote `tf` [57], em que é possível publicar e ler transformações entre referenciais, usando `tf:TransformBroadcaster` `tf::lookupTransform`, respetivamente. Sabendo a transformada estática do *end effector* para o laser (denominada de "Tf_laser_eef" na figura 4.11), e sabendo que o laser deve estar no ponto de aproximação calculado anteriormente (e com a mesma orientação), consegue-se descobrir a posição e a orientação do *end effector*.

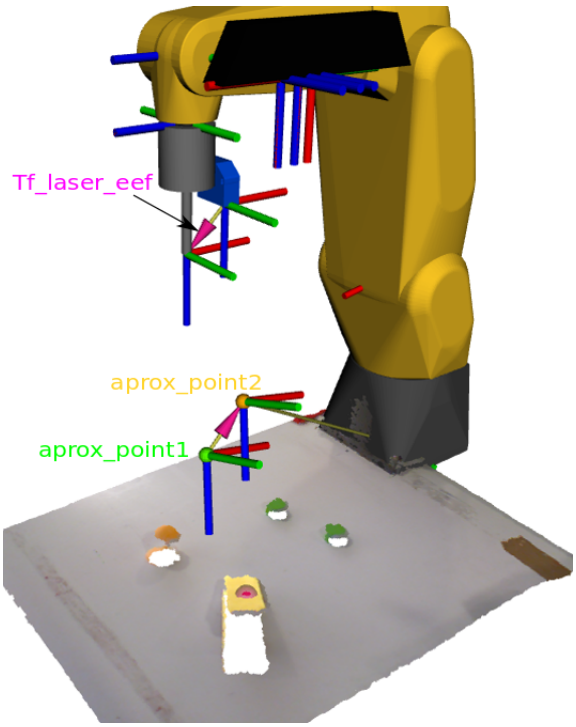


Figura 4.11: Pontos de aproximação e transformada laser-*end effector*.

Após o cálculo dos pontos de aproximação, estes são publicados no tópico `/targets_pose`, para posteriormente serem usados pelo nó principal, `move_fanuc.py`, que controla toda a sequência de tarefas.

4.3.2 Execução de Movimentos

Para a execução dos movimentos do manipulador é necessário a instalação de algumas ferramentas que facilitam a sua implementação prática. São exemplos destas ferramentas, o `ROS Industrial`, `ROS fanuc` e o `Moveit`.

Como as ferramentas existentes não permitem o controlo do ar comprimido para o nosso manipulador, é essencial usar uma alternativa desenvolvida por um antigo aluno em [21].

ROS Industrial e ROS fanuc

`ROS-Industrial` é um projeto de código aberto que alarga capacidades avançadas do `ROS` para a robótica e automação industrial. Este repositório contém livrarias, modelos virtuais (URDF), ferramentas e drivers para hardware industrial [58].

O `ROS-I` inclui meta-pacotes direcionados para vários vendedores industriais, como a ABB, Fanuc, Universal Robots, Motoan, entre outros. No entanto, para controlar um manipulador FANUC basta incluir o meta-pacote `ROS fanuc`, e utilizar os pacotes correspondentes ao modelo do manipulador, neste caso, FANUC LR Mate 200iD, e o pacote `fanuc_driver`. É de realçar que o `ROS fanuc` ainda não apresenta a versão completa para o modelo utilizado, no entanto, fez uso do meta-pacote `fanuc_experimental`, que contém alguns pacotes ainda em desenvolvimento ou em teste.

MoveIt

Para complementar e facilitar o controlo e planeamento de trajetórias do manipulador, optou-se por usar outro meta-pacote, o `ROS MoveIt`.

`Moveit` é um *software open source* que inclui pacotes e ferramentas necessárias à manipulação móvel. Esta ferramenta além de providenciar uma plataforma de uso fácil para desenvolvimento de aplicações robóticas, integra diferentes áreas: manipulação, perceção, cinemática, planeamento de movimento dinâmico, verificação de colisões, navegação e controlo [59].

Para correta configuração do `Moveit` é necessário definir algumas especificações do robô, tais como tipo de juntas, os seus respetivos limites e o modelo URDF do sistema. Após estas definições, é possível criar grupos de planeamento, isto é, ao fornecer um determinado ponto final para a garra, por exemplo, pode-se escolher qual o grupo de juntas a atuar. Apresenta-se na figura 4.12 um esquema da arquitetura usada pelo `Moveit`, para melhor compreensão.

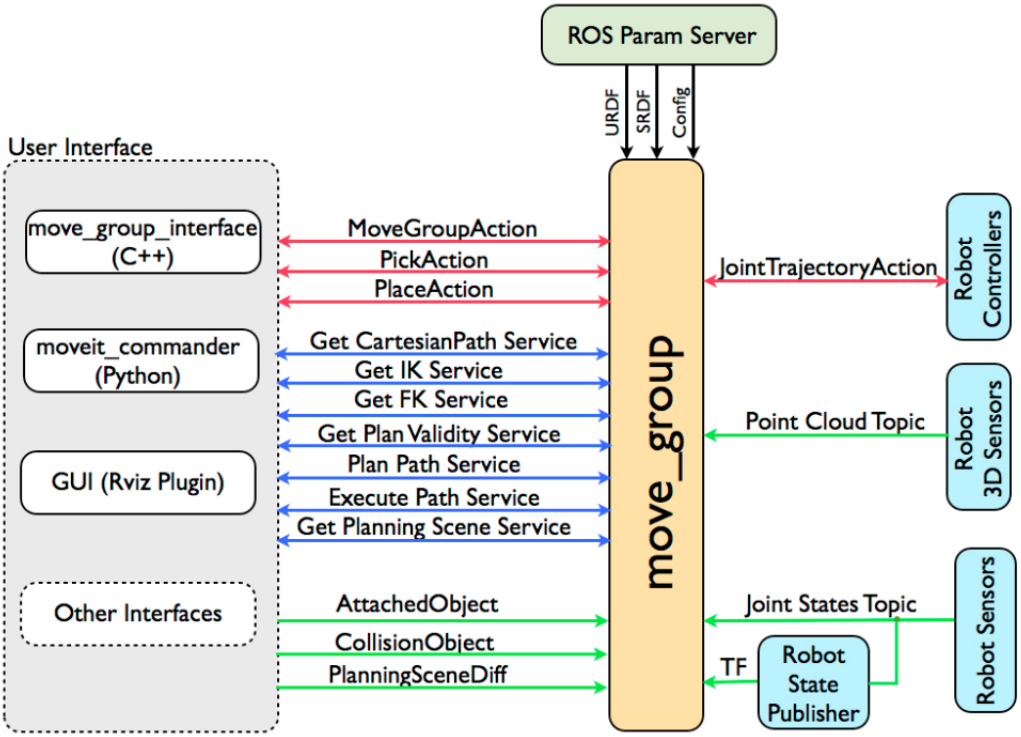


Figura 4.12: Arquitetura `Moveit` [59].

4.3.3 Ativação IO's

Para agarrar um objeto, é necessário ativar a I/O de sucção. Como os pacotes `ROS`, `ROS-Industrial` e `Moveit`, responsáveis por interagir com o manipulador FANUC ainda não estão preparados para esta tarefa, houve necessidade de contornar este problema. Para o efeito, desenvolveu-se uma solução em [21], em que através da utilização de um micro-controlador é possível comandar um circuito com relés que ativam entradas digitais

do manipulador, e consequentemente comutem saídas digitais. O circuito desenvolvido é apresentado na figura 4.13.

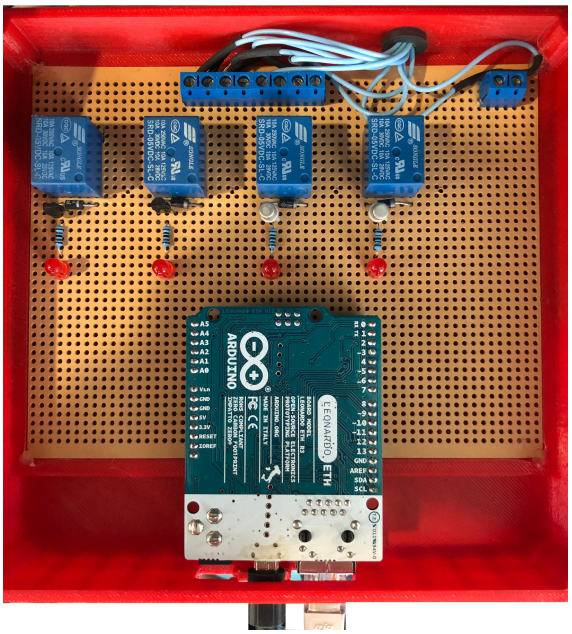


Figura 4.13: Circuito externo para ativar IO's.

A conexão Ethernet computador-micro-controlador é realizada através do protocolo TCP/IP, e para troca de informação utiliza-se o nó `vs_IO_client`.

Como a válvula que controla o ar comprimido é bi-estável, para ativar a sucção é necessário ativar uma saída e desativar outra. Para tal, foi criado um programa TP, designado `MONIO`, de modo a controlar duas saídas consoante um único sinal de entrada. Este programa é executado em *Background Logic* e está constantemente a verificar o estado das entradas digitais, sem interferir com programas principais que corram em simultâneo, como é o caso do `ROS`.

4.3.4 Fluxo do Processo global de Bin-Picking

O nó `move_fanuc.py`, sendo o nó principal, controla toda a sequência de tarefas inerentes ao processo de *Bin-Picking*. Inicialmente coloca o manipulador numa posição pré-definida de forma a que a kinect alcance os objetos sobrepostos na bancada, e posteriormente lança os nós `objDetection` e `pointTFtransfer`, obtendo assim os pontos de aproximação, através dos tópicos referidos anteriormente. Depois de receber a informação necessária, os nós são terminados e posiciona o manipulador para a leitura do sensor laser. De modo a obter a distância é lançado um outro nó, `sensorRS232`, no qual retorna uma média de 10 leituras a partir do momento em que o nó é lançado.

Este nó é terminado, calcula-se o ponto de *grasping* com base nas leituras do laser e no ponto de aproximação do *end effector*, e procede-se ao *picking*. O objeto é largado posteriormente num local pré-definido. Apresenta-se na figura 4.14 uma esquematização do processo global.

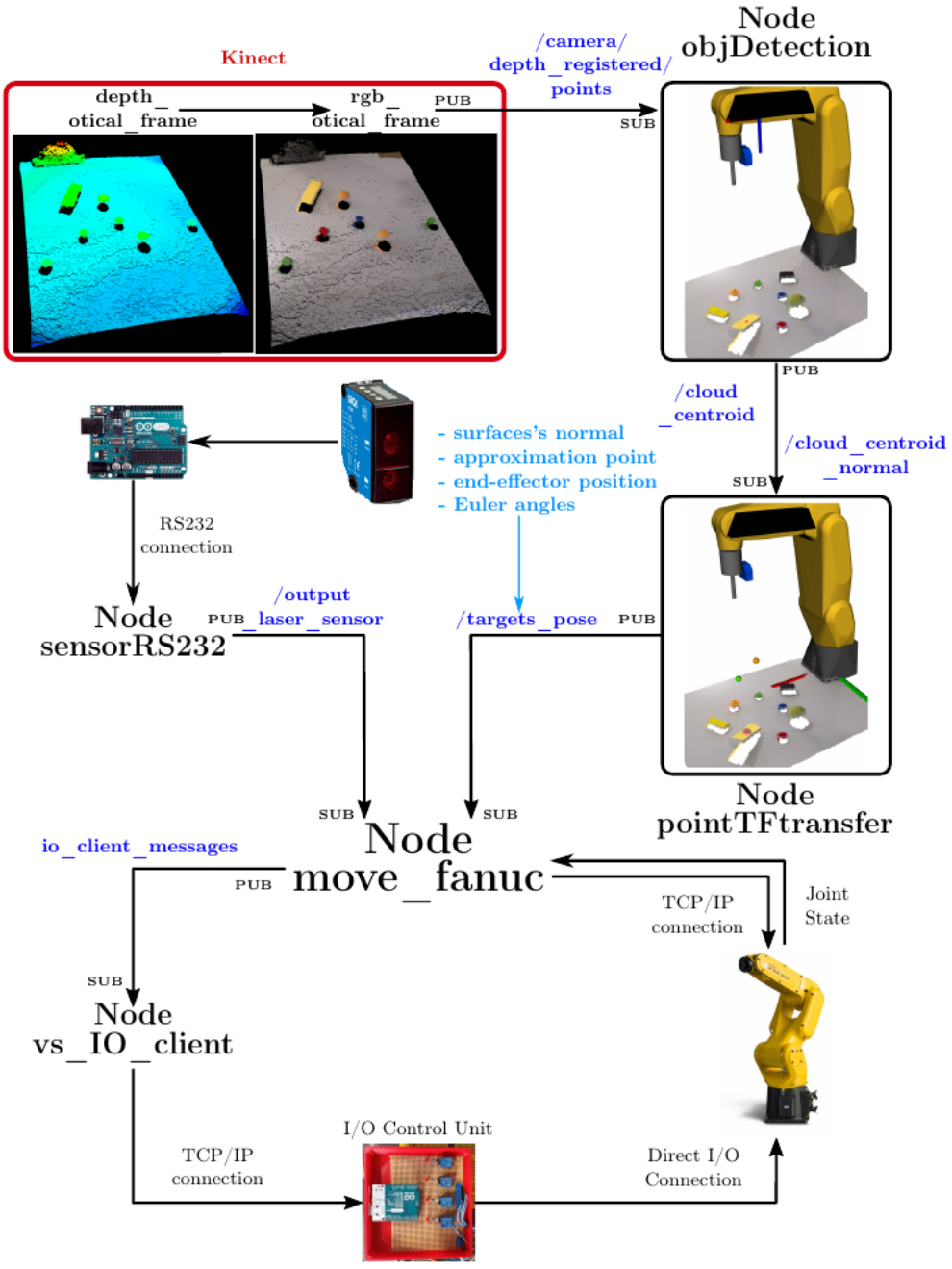


Figura 4.14: Arquitetura esquemática do processo de *Bin-Picking* [13].

Capítulo 5

Calibração

Um dos problemas em robótica industrial consiste na determinação da posição relativa entre dois elementos com referenciais distintos. Matematicamente, esta posição é definida por um vetor de translação ($t_{[3\times1]}$) e por uma matriz de rotação ($R_{[3\times3]}$).

Dado que as calibrações são responsáveis pela determinação das transformações geométricas entre dois sistemas de coordenas, ou seja, uma relação geométrica que traduz a posição e orientação entre referenciais, este capítulo tem como objetivo descrever as técnicas de calibração usadas neste projeto.

A fim de integrar todo o hardware num sistema global, é importante conhecer as transformações entre os componentes presentes. Uma vez que o manipulador e a plataforma podem estar em constante movimento, pode ser necessário conhecer as transformações entre referenciais ao longo do tempo. Para o efeito, usa-se o pacote `tf`, que permite um "rastreamento" dos referenciais após o arranque do sistema, ou seja, é possível saber a relação geométrica entre o referencial A e o referencial B num determinado instante de tempo.

5.1 Manipulador-plataforma

A modelação dos sistemas em `ROS`, tais como robôs, sensores ou ambientes de trabalho, é realizada por ficheiros *Unified Robot Description Format* (URDF). Estes ficheiros apresentam um formato XML e tem como objetivo a descrição cinemática e dinâmica, a representação visual e a definição do modelo de colisões de um robô, ou seja, caracteriza as suas juntas e os seus elos em termos de, por exemplo, o tipo, os limites e dimensões.

Uma vez que a plataforma móvel tem um elevado número de componentes, a sua modelação manual (em formato XML) tem algum grau de complexidade. Para a criação do seu modelo URDF, utilizou-se uma extensão ao `Solidworks`, designada por `sw_urdf_exporter` [60]. Este *add-in* providencia um gestor de propriedades que permite configurar o modelo URDF manualmente através do modelo CAD, assim, é necessário configurar manualmente as propriedades de cada elo e construir a árvore de elos e juntas. Depois de finalizado, é gerado um pacote ROS conforme o *assembly* realizado, com os ficheiros URDF e respetivas *meshes* (para representação mais realista).

Na configuração de juntas do modelo, à exceção das juntas correspondentes às duas rodas diferenciais, todas foram consideradas fixas. Estas juntas, foram consideradas como "contínuas", uma vez que não têm limites associados aos seus eixos de rotação.

Para visualizar e testar o modelo criado, fez-se uso de dois pacotes bastante conhecidos, `joint_state_publisher` e `robot_state_publisher`. O primeiro utiliza o URDF especificado pelo parâmetro lançado como "robot_description", identifica todas as juntas não-fixas e publica uma mensagem do tipo `JointState`, com todos os valores de juntas definidos na interface fornecida pelo mesmo. O segundo pacote utiliza o mesmo parâmetro e as posições de juntas publicadas no tópico `joint_state` de forma a calcular a cinemática direta do robô e a publicar os resultados via `tf`. Na figura 5.1 pode-se visualizar a interface criada pelo `joit_state_publisher` bem como os resultados fornecidos pelo `rviz`.

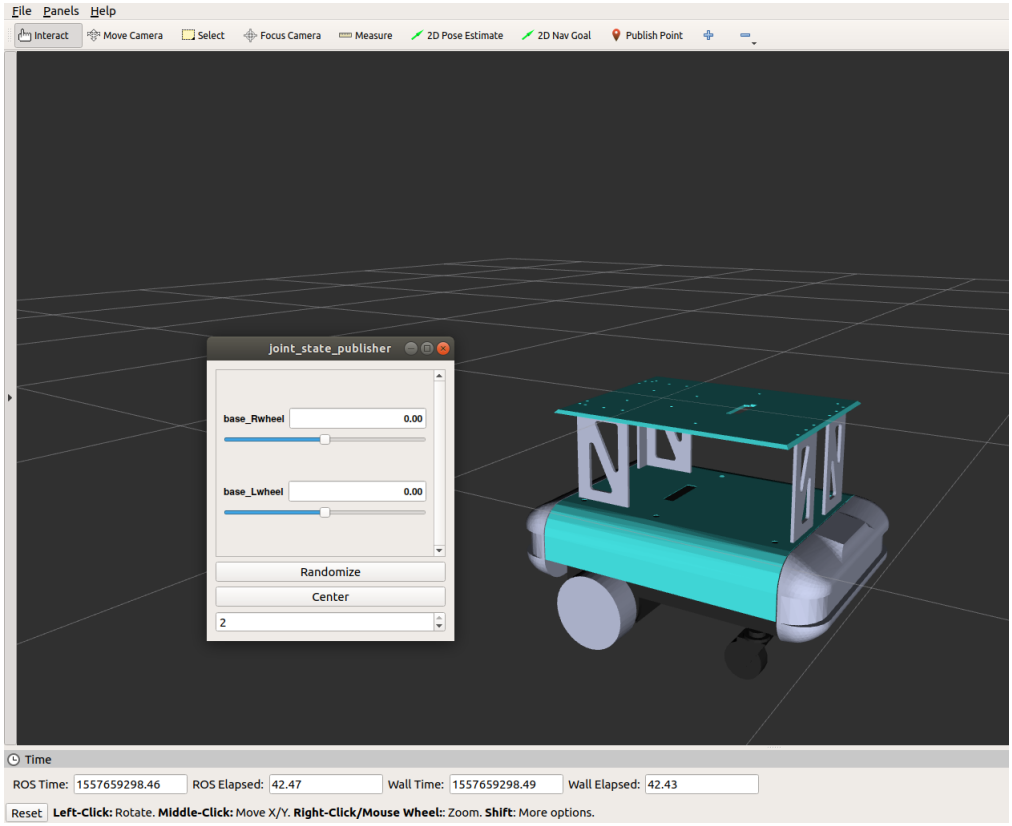


Figura 5.1: Representação visual do modelo URDF da plataforma.

Aquando da criação do modelo da plataforma, teve-se em conta a posterior integração manipulador-plataforma, isto é, localizou-se o referencial associado à placa superficial da plataforma, no local onde é fixada a base do manipulador. Assim, tendo o modelo URDF do manipulador, apenas foi necessário importá-lo e associar uma junta fixa, com transformação e rotação nula, do referencial da placa à base do robô. O URDF do manipulador é fornecido pelo pacote `fanuc_lrmate200id_support`, contido no meta-pacote `fanuc_experimental`, no qual é necessário incorporar o *gripper* usado neste projeto.

Após a integração, o aspeto geral do modelo é apresentado na figura 5.2.

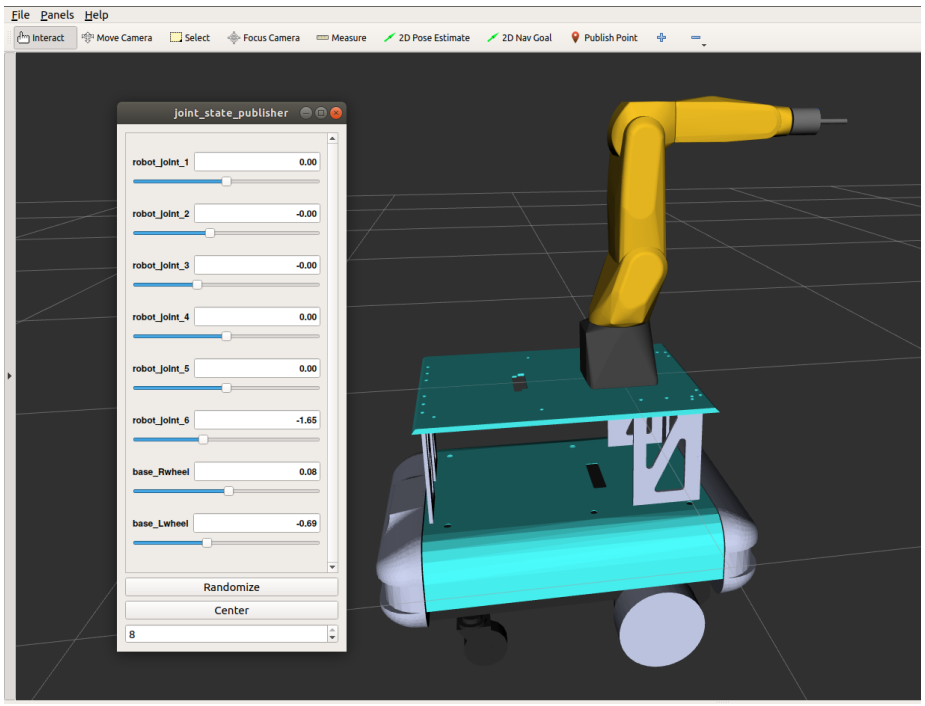


Figura 5.2: Representação visual do modelo URDF (plataforma + manipulador + *gripper*).

5.2 Lasers Hokuyo-Manipulador

Como referido anteriormente, para navegação são usados dois sensores LRF, o laser dianteiro, Hokuyo UTM-30LX, e o traseiro, Hokuyo URG-04-LX-UG01. De modo a incluir os seus modelos no sistema global, é necessário efetuar as respetivas calibrações. O modelo do laser dianteiro pode ser importado do pacote `hector_sensors_description`, enquanto que o do laser traseiro foi importado de [61]. É importante salientar que o modelo URDF do laser dianteiro apenas apresenta um referencial associado à origem do feixe laser, enquanto que o laser traseiro, para além deste referencial, apresenta um referencial associado à base do laser.

De modo a conhecer a transformação geométrica da base do robô (`robot_base_link`) para o laser, a fim de realizar uma correta calibração, deve proceder-se do seguinte modo:

1. Colocar a ponta do *gripper* perpendicular à placa da plataforma, definindo, por exemplo, todas as juntas na posição zero e a quinta junta a $-90°$.
2. Mover o robô apenas em Z, de modo a ser detetado pelo laser.
3. Através da informação adquirida pelo laser, localizar a ponta do *gripper* no referencial do laser, de forma a conhecer a transformação laser-*gripper*, ${}^{laser}T_{gripper}$.
4. Adquirir a posição e orientação da ponta do manipulador relativamente à sua base, ou seja, conhecer a transformação base do robô-*gripper*, ${}^{robot\_base}T_{gripper}$.
5. Calcular a transformação base do manipulador-laser, ${}^{robot\_base}T_{laser}$, com base na equação 5.1 e consequentemente implementa-la no modelo URDF global.

$$^{robot\_base}T_{laser} = ^{robot\_base}T_{gripper} * {^{laser}T_{gripper}}^{-1} \quad (5.1)$$

No entanto, este procedimento não seria eficaz uma vez que os lasers proporcionam erros superiores a 30 mm. Assim, para localizar a cota Z do laser relativamente à base do robô, executaram-se os passos anteriormente enumerados por 1 e 2, como representado na imagem 5.3. Posteriormente, mediu-se a distância do referencial `Robot_base` ao centro do laser de modo a conhecer a diferença entre abcissas, uma vez que não existe desfasamento em Y. Assim, o laser visto do referencial `Robot_base`, apresenta as seguintes coordenadas: X=0.1665 m, Y=0 m e Z=0.055 m (±0.0005 m).

Como o modelo do laser traseiro contém dois referenciais localizados entre si (figura 5.4), basta definir a abcissa entre o referencial da base do laser e o do robô, sendo que as bases estão no mesmo plano e não há desfasamento em Y. Visto que o laser traseiro necessita de estar com o feixe laser direcionado para trás da plataforma, é de destacar que este laser apresenta uma rotação de 180° em Z face ao `Robot_base` e está desfasado em X 0.532 m.

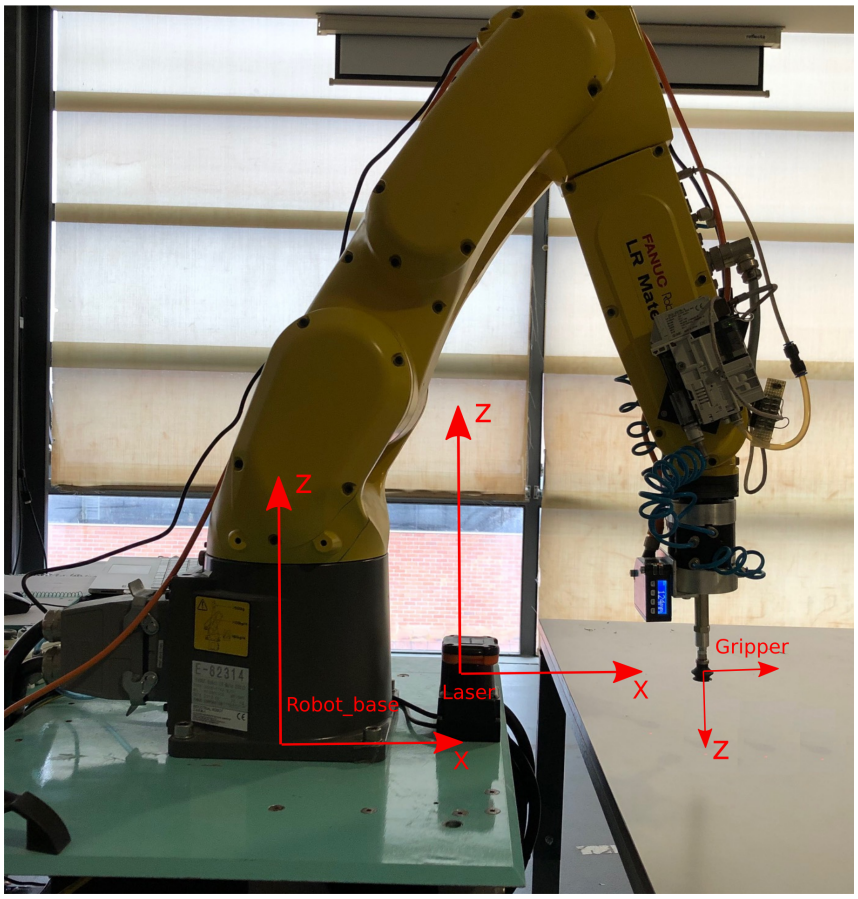


Figura 5.3: Representação do processo de calibração dos lasers Hokuyo e dos referenciais necessários.

Após as suas calibrações, pode-se visualizar a presença dos lasers no modelo total URDF e os seus respetivos referenciais associados na figura 5.4.

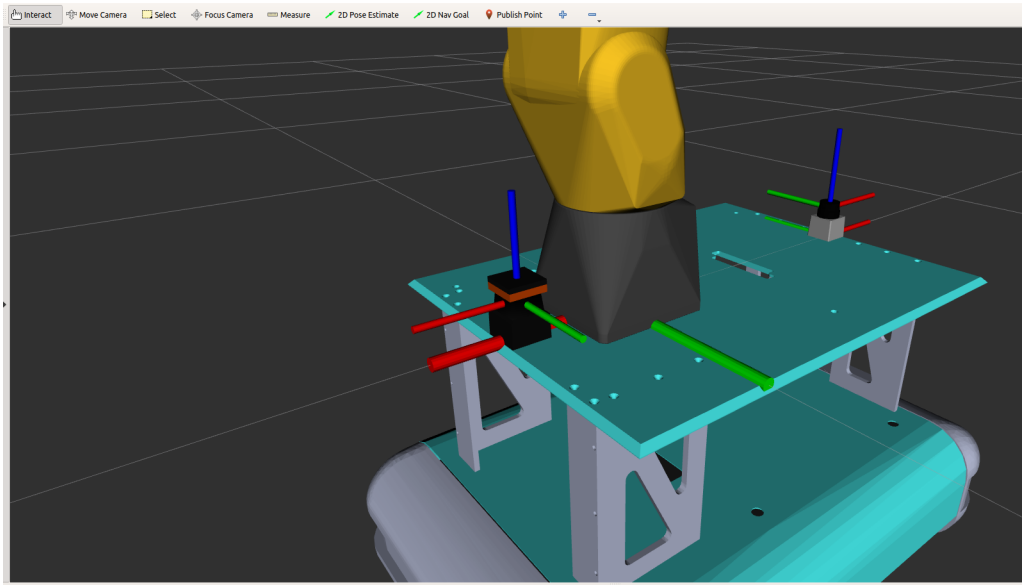


Figura 5.4: Representação visual do modelo URDF completo e respetivos referenciais associados.

5.3 Câmara-Manipulador

A calibração da câmara consiste em duas partes, calibração intrínseca e calibração extrínseca. A primeira representa a transformação de coordenadas métricas para pixels, isto é, do sistemas de coordenadas da câmara 3D para um sistema de coordenadas na imagem 2D. A segunda permite identificar a relação entre os sistemas coordenadas da câmara e do mundo.

5.3.1 Calibração dos Parâmetros Intrínsecos

A calibração intrínseca é responsável pela estimativa de parâmetros de ambas as câmaras presentes, RGB e IR. A distância focal, coordenadas do centro ótico, coeficiente de *"skew"* e coeficiente de distorção da lente, são os parâmetros intrínsecos necessários de quantificar a fim de descrever o correto modelo físico da câmara [62].

Apesar desta calibração não ser estritamente necessária, uma vez que o driver `openni_camera` providencia os parâmetros necessários com algum grau de confiança, é aconselhado realizar uma nova calibração, visto que o intuito é fazer *bin-picking* com elevada precisão.

Dado que esta calibração foi realizada em [13], foram mantidos os mesmos parâmetros, contudo, após alguns testes verificou-se que a calibração não gerava resultados conforme o esperado. Estes erros podem ter origem no incorreto manuseamento da câmara, ou seja, pequenas oscilações de lentes quando sujeito a impactos, por exemplo. Para gerar os ficheiros de calibração necessários, usou-se o nó `cameracalibrator.py` do pacote `camera_calibration` [63], que pode ser usado para calibrar as câmaras monocular e estéreo. Usou-se um tabuleiro de xadrez 8×6 (interior), com quadrados 105.3 mm, como se pode ver na figura 5.5.

Os ficheiros gerados, pertencem ao pacote `robonuc_integration`, e são designados

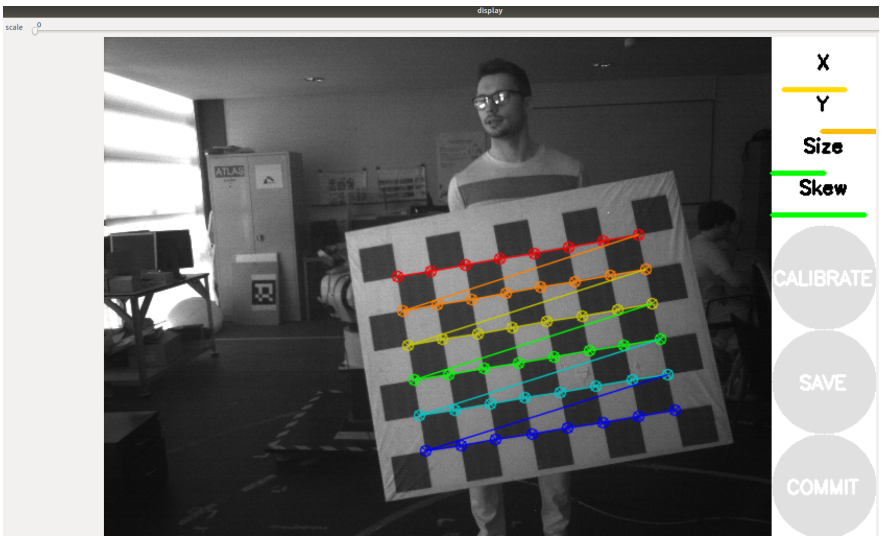
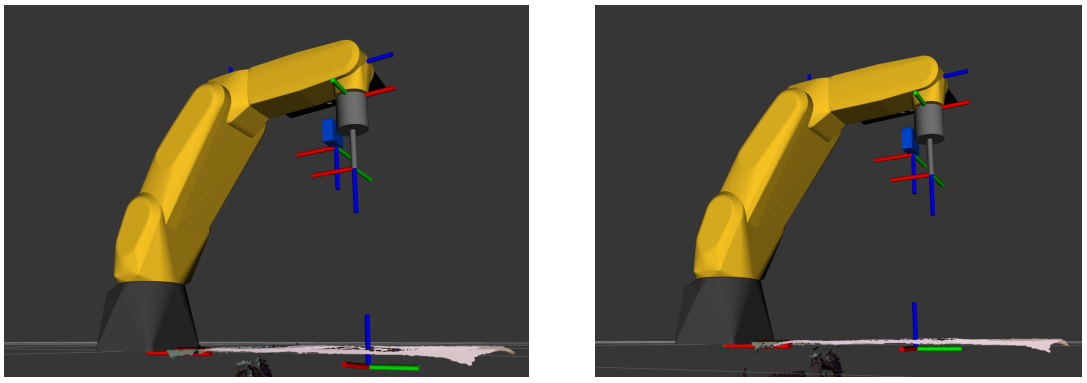


Figura 5.5: Representação do processo de calibração intrínseca.

por `rgb4_A00366922636045A.yaml` e `depth4_A00366922636045A.yaml`. Para que não use os parâmetros padrão, estes ficheiros devem de ser fornecidos como argumentos no `openni_camera.launch`.

Na figura 5.6 pode-se visualizar as diferenças nas imagens obtidas, antes e após o processo de calibração. Estas imagens fazem parte do programa desenvolvido para a calibração extrínseca (que será explicada na secção seguinte), que visa a deteção de um marcador do tipo aruco na bancada de trabalho e consequentemente a representação do seu referencial. Assim, antes da calibração pode-se constatar na figura 5.6a, que o referencial correspondente ao aruco se encontra com um desfasamento em relação à nuvem de pontos e apresenta uma orientação incorreta. Após a calibração, ou seja na figura 5.6b, visualiza-se que a orientação do aruco está melhorada e o desfasamento em Z diminui significativamente.

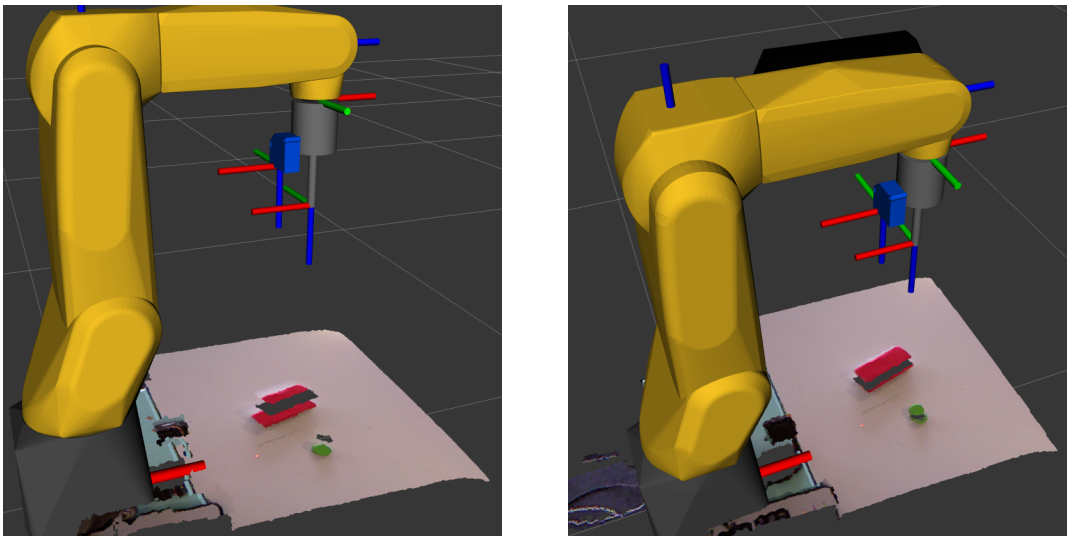


(a) Pré-calibração. (b) Pós-calibração.

Figura 5.6: Representação das melhorias após o processo de calibração intrínseca, comparativamente aos parâmetros usados no trabalho anterior.

Comparativamente aos parâmetros padrão fornecidos pelo fabricante, não é verificado

um desfasamento acentuado em Z, na deteção de um aruco à superfície da mesa. Contudo, como se pode verificar na figura 5.7a, na presença de objetos com alguma altura, existe um erro na atribuição de planos entre a nuvem de pontos e a respetiva cor em rgb. O plano com cor vermelha está na superfície da mesa quando deveria de estar na superfície mais elevada.



(a) Pré-calibração. (b) Pós-calibração.

Figura 5.7: Representação das melhorias após o processo de calibração intrínseca, comparativamente aos parâmetros padrão.

5.3.2 Calibração dos Parâmetros Extrínsecos

Os parâmetros extrínsecos descrevem a posição e orientação da câmara em relação a um sistema de referência, isto é, permitem conhecer a posição de um objeto de referência num cenário real.

Nesta situação, como a câmara está acoplada ao 4º elo, é estritamente necessário calcular a transformação (estática) entre o referencial associado ao elo e o referencial ótico da câmara.

Para a realização desta calibração, usou-se o pacote `visp_hand2eye_calibration` [64], que tem como função estimar a posição da câmara em relação a um referencial do robô, neste caso, `robot_link_4`. Para o seu uso, é necessário fornecer ao nó responsável pela calibração as transformações `world_effector` e `camera_object`. A primeira corresponde à transformação do referencial da base do robô (`robot_base_link`) para o referencial do 4º elo (`robot_link_4`). A segunda corresponde à transformação da câmara para o referencial do objeto/marcador. Desta forma, para calcular a transformação `camera_object` fez-se uso do pacote `aruco_detect` [65] que deteta os marcadores arucos na cena e publica as suas posições relativas à câmara no tópico `/fiducials_transforms`. É de salientar que, inicialmente, o aruco utilizado para a calibração apresentava um tamanho de 83 mm, no entanto o referencial representado no *rviz* associado ao marcador tinha pequenas vibrações associadas a incertezas de localização. Assim, posteriormente,

optou-se por usar um aruco de tamanho médio, 168 mm, de forma aumentar a precisão do detetor de marcadores.

Pode-se visualizar na figura 5.8 a interface durante o processo de calibração extrínseca.

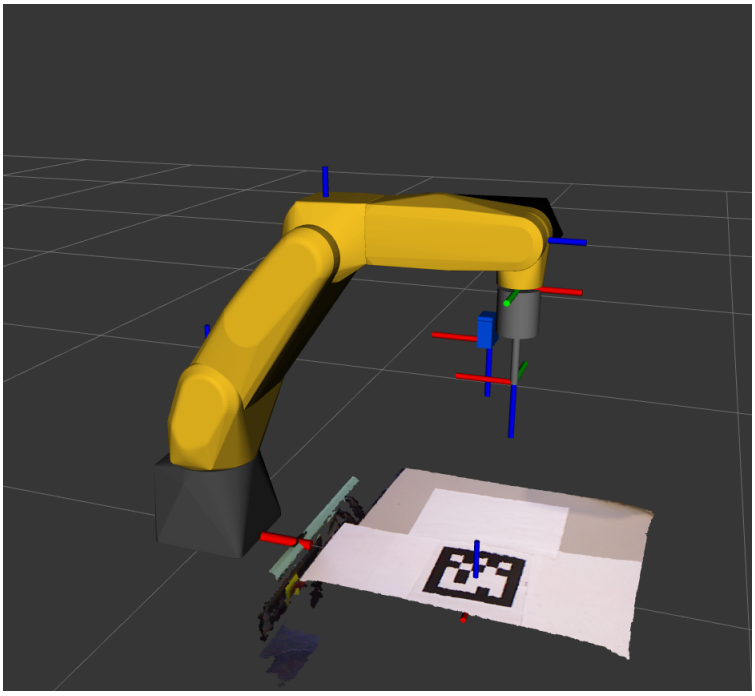


Figura 5.8: Representação do processo de calibração extrínseca.

Para o processo de calibração criou-se o programa `get_extr_camera_calibration.py` que é responsável pela seguinte sequência de processos:

1. Subscreve o tópico `/fiducial_transforms` e publica essa transformação (do `camera_rgb_optical_frame` para `calibration_object`).
2. Lê a transformação entre o `robot_base_link` e o `robot_link_4` e publica-a no tópico `world_effector`.
3. Lê a transformação entre o `camera_link` e o `calibration_object` e publica-a no tópico `camera_object`.
4. Espera pelo *input* do utilizador: retorna ao processo nº1 caso o utilizador pressione a tecla '*enter*'; avança para processo seguinte (nº5) caso o utilizador pressione a tecla 'y'.
5. Pede o serviço `compute_effector_camera` providenciado pelo `visp_hand2eye_calibration`, que retorna a transformação desejada.
6. Apresenta a transformação do 4º elo para a câmara.

Antes da execução do programa referido deverá-se-á lançar o `camera_extr_calibration.launch`, responsável por inicializar e configurar todos os nós necessários ao processo.

Posteriormente, o resultado apresentado por `get_extr_camera_calibration.py` deverá ser incorporado no ficheiro URDF, uma vez que corresponde a uma transformação estática do manipulador para a câmara.

Pode-se visualizar na listagem 5.1, a transformação inserida no ficheiro `binpicking_macro.xacro`.

Listagem 5.1: Ficheiro URDF: `binpicking_macro.xacro`.

```
<?xml version="1.0" ?>
<robot
    xmlns:xacro="http://ros.org/wiki/xacro">
    <!-- include the manipulator xacro -->
    <xacro:include filename="$(find␣bin_picking)/urdf/
        binpicking_robot_macro.xacro"/>
    <!-- include the kinetic xacro -->
    <xacro:include filename="$(find␣hector_xacro_tools)/urdf/
        inertia_tensors.urdf.xacro"/>
    <xacro:include filename="$(find␣hector_sensors_description)/urdf/
        kinect_camera.urdf.xacro" />

    <!-- macro defining the whole robot -->
    <xacro:macro name="bin_picking" params="prefix">
        <xacro:bin_picking_manipulator prefix="${prefix}" />
        <xacro:kinect_camera name="camera" parent="${prefix}robot_link_4">
            <origin
                xyz="0.254881746287␣0.128927213047␣-0.0237188254049"
                rpy="2.03114796593␣1.54366070438␣-2.67392737672" />
        </xacro:kinect_camera>
    </xacro:macro>
</robot>
```

5.4 Laser 1D-Manipulador

A fim de calibrar o sensor laser 1D e incorporar o seu sistema de coordenadas no sistema global, é necessário determinar a transformação entre o referencial da ponta do *end effector* e o emissor do laser.

Inicialmente, como se pode ver à esquerda da figura 5.9, foi colocada a quinta junta do robô a 90° (e todas as outras a 0°) com intuito de estabelecer perpendicularidade entre o *end effector* e a mesa. Com esta configuração de juntas, sabendo que a cota (coordenada Z) da ponta do *gripper* relativamente à base do manipulador era de 451.02 mm, foi assinalada a interseção do feixe laser com a mesa. De modo a identificar o ângulo do feixe lazer relativamente à ponta do robô, moveu-se o manipulador em Z, com o *teach pendant* até à cota Z =37.43 mm, tendo-se verificado um desvio de 4 ± 0.5 mm.

Para um movimento de 413.59 mm, o feixe laser distanciou-se do ponto inicial 4 mm, o que representa uma inclinação do laser em 0.554°. Sendo o desvio inferior a 1°, este valor foi considerado desprezável para as transformações entre referenciais.

Com intuito de calcular as translações em X e Y, foi colocado na ponta do manipulador uma caneta de forma a conseguir medir as distâncias necessárias. Com o manipulador, começou-se por realizar movimentos lineares em X e Y, de forma a visualizar as orientações desses eixos. Posteriormente, foi realizada uma medição em cada um dos eixos, X e Y, da distância do laser à ponta do *gripper* (como se pode observar à direita da figura

5.9) obtendo-se 48.5 mm e 16 mm de desvio, respetivamente. Uma vez que o sensor laser não mede distâncias inferiores a 100 mm, para a obtenção da translação em Z, obteve-se a cota da ponta do manipulador quando encostada à mesa (Z_1) e movimentou-se o manipulador em Z até ultrapassar o limite mínimo de deteção do sensor. Nesta posição, obteve-se a cota da ponta do manipulador (Z_2) e calculou-se a translação do laser em Z (63.6 mm) com a seguinte expressão: $translacao = leitura\_laser - (Z_2 - Z_1)$).

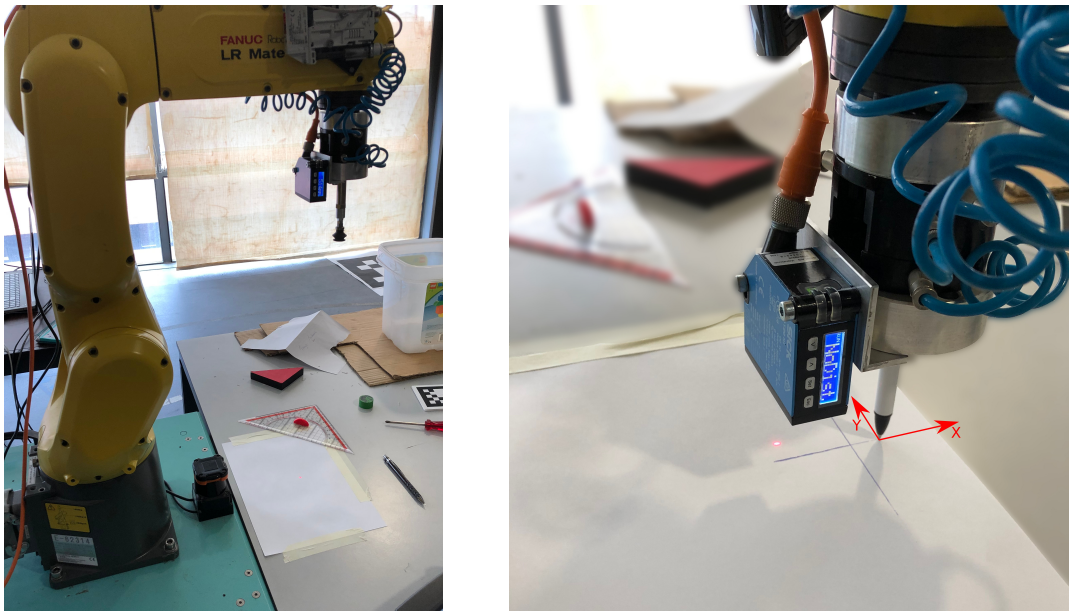


Figura 5.9: Representação do processo de calibração do laser 1D.

Esta transformação é representada no ficheiro URDF do modelo a carregar, designado por `binpicking_sensor_macro.xacro`, do pacote `bin_picking`.

5.5 Sistema Total Calibrado e Integrado

Com a integração física dos componentes e após todas as calibrações realizadas, fica concluída toda a integração virtual. O sistema está, então, preparado para ser "trabalhado" virtualmente , onde todos os componentes presentes são localizados. Desta forma, é possível visualizar a árvore de transformações (`tf tree`) com os referenciais de todo o sistema, na figura 5.10 ou no anexo B (em maior dimensão).

Para visualizar o modelo total, basta executar o ficheiro `display_total_urdf.launch` presente no pacote criado, `robonuc_integration`. Após a sua execução, o modelo integrado será apresentado, tal como na figura 5.11.

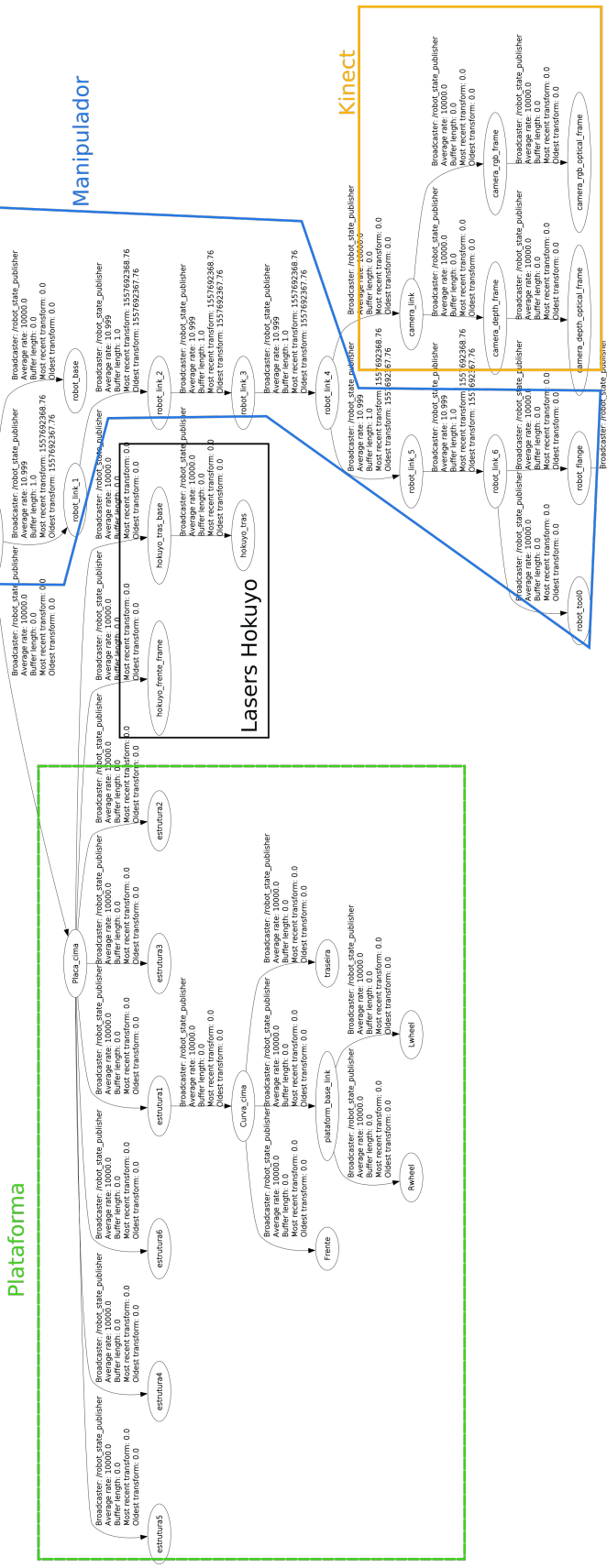


Figura 5.10: Árvore de transformações do modelo total.

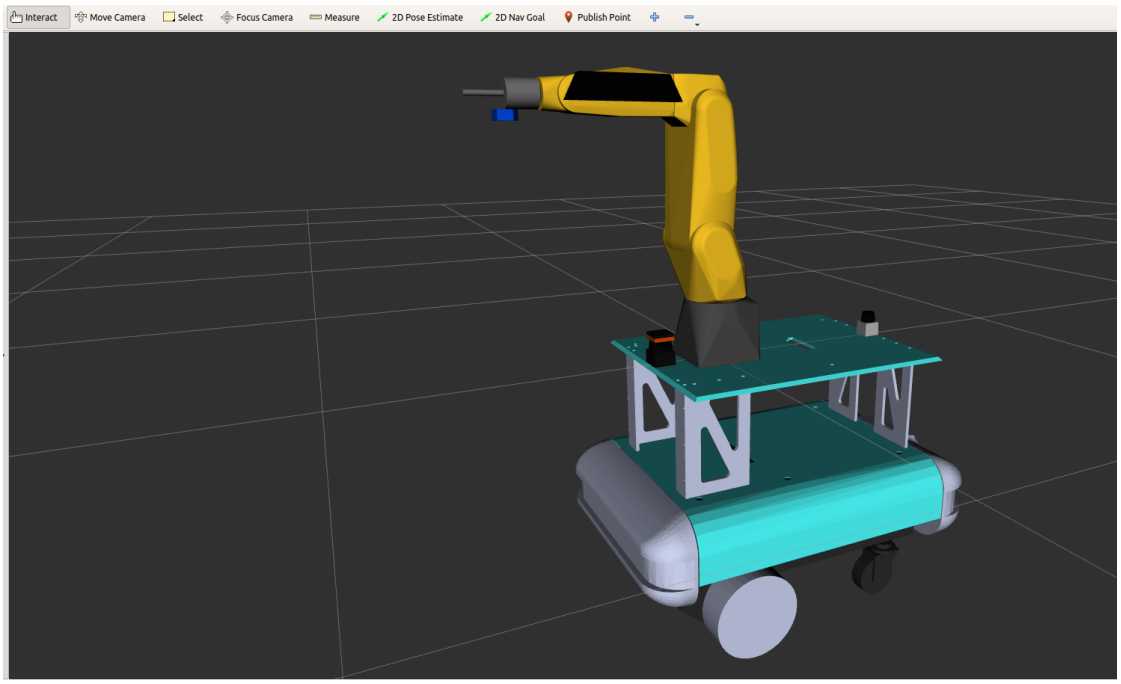


Figura 5.11: URDF do sistema total integrado.

Capítulo 6

Arquitetura Integrada

A manipulação móvel, como já foi referido anteriormente, é uma área de extrema complexidade uma vez que visa a junção de diversos sistemas e diferentes áreas. Para integrar todos os sub-sistemas (plataforma e braço robótico) com a finalidade de executar tarefas de *bin-picking*, foi desenvolvida uma arquitetura integrada baseada em `ROS`. Neste capítulo é descrita a solução proposta para a integração do sistema total.

6.1 Filosofia dos estados

Para a integração de todos os sistemas, com o objetivo de realizar o *bin-picking* com precisão, foram criados diversos estados para o sistema ROBONUC. Assim, o manipulador móvel pode apresentar 5 estados:

- Estado 0: Off (Sistema estático).
- Estado 1: Navegação 2D no ambiente inserido.
- Estado 2: Aproximação à bancada (posto de trabalho).
- Estado 3: Orientação da plataforma face à bancada.
- Estado 4: *Bin-Picking.*

O estado 0 representa o modo Off, ou seja, quando o sistema se encontra estático à espera da inicialização do sistema total. O estado 1 equivale ao estado de navegação 2D, no qual pode alternar entre modo automático ou manual. O estado 2 corresponde à aproximação da bancada de trabalho monitorizado através da informação do laser e que só é ativado quando o utilizador tem a perceção de uma bancada de trabalho (para realização de *bin-picking*), na direção e sentido da plataforma. Após o término do estado 2, o sistema comuta para o estado 3 onde é realizada a orientação em relação à bancada de trabalho, e posteriormente, ativa o estado 4 onde é desencadeado o processo de *bin-picking.* No final do estado 4 o sistema retorna ao estado 1. O diagrama de estados representado na figura 6.1 permite uma melhor compreensão de todo o processo descrito.

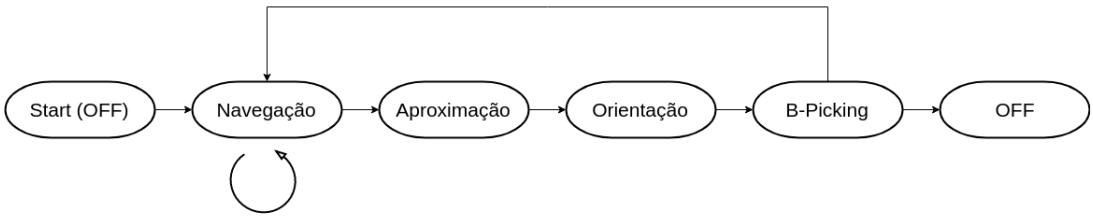


Figura 6.1: Diagrama de estados.

6.2 Controlo entre estados

Para a incorporação dos estados de execução é necessária a criação de um nó "árbitro" que controle todo este processo. O nó desenvolvido é designado por `integrated_referee-.cpp` e é controlado imperativamente pelo comando *xBox*. A nomenclatura de referências usada para os seus botões encontra-se representada na figura 6.2.

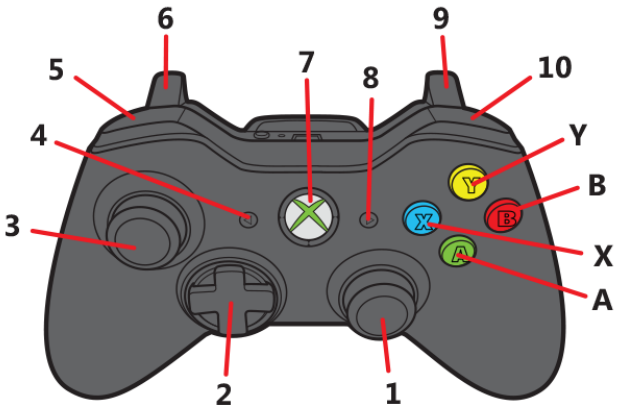
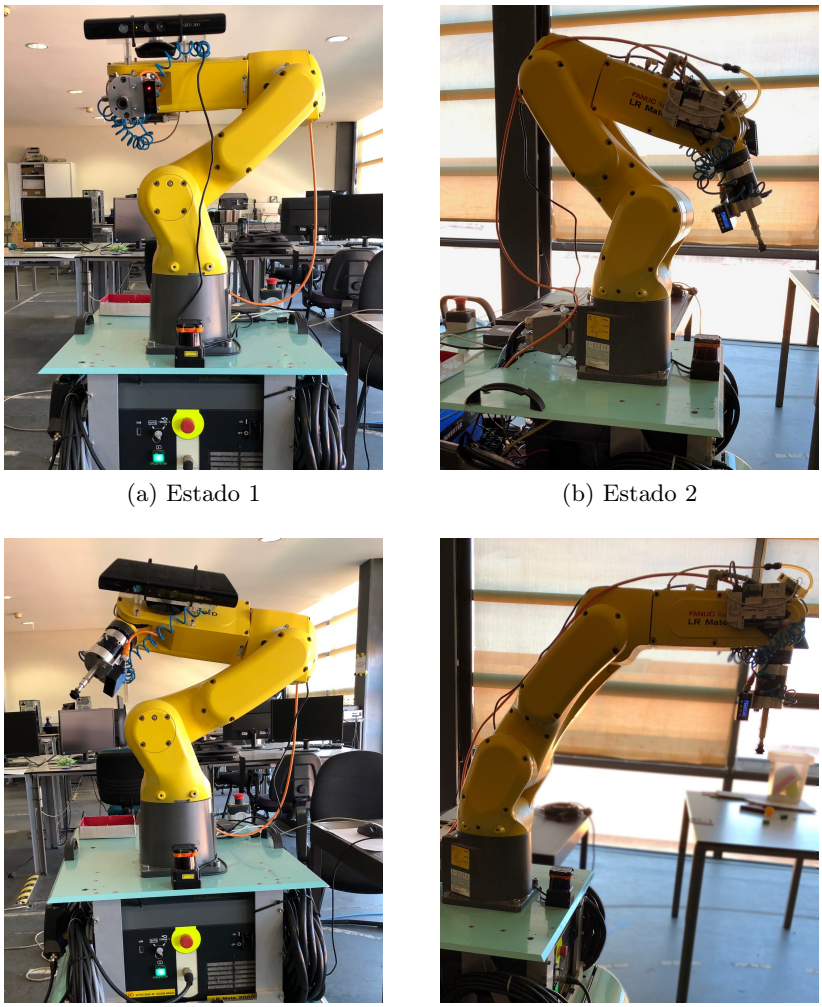


Figura 6.2: Identificação dos botões do comando *XBOX 360* [22]. (Descrição na tabela 6.1.)

De modo a ter o controlo do sistema, cada estado tem associado uma configuração do manipulador. Assim, foi criado um servidor de ações designado por `robot_status_-server.py` e a ação correspondente, `RobotStatusAction`. Conforme o *goal* enviado por parte do cliente (`integrated_referee`), o servidor de ações move o manipulador para as posições pré-definidas. De forma a que o *goal* seja aceite e processado, este deve de ser um número inteiro contido no conjunto de valores {1,2,3,4}, ou seja, um número corresponde a uma configuração do manipulador (como se pode ver na figura 6.3).

Para movimentar o manipulador, no decorrer do trabalho, fez-se uso do pacote `Moveit` e das ferramentas disponibilizadas pelo mesmo. Para o efeito, através do assistente de configuração do `Moveit`, foi necessária a criação de um novo pacote, designado por `moveit_ROBONUC_pack`. Posteriormente, para a configuração das comunicações com o controlador FANUC fez-se as alterações necessárias de acordo com o tutorial disponibilizado em [66]. Este pacote contém todas as configurações necessárias ao ROBONUC, uma vez que o uso das configurações, tendo em conta apenas o manipulador, poderia gerar colisões com os outros componentes da plataforma.



(a) Estado 1 (b) Estado 2

(c) Estado 3 (d) Estado 4

Figura 6.3: Representação das posições do manipulador para os diferentes estados.

O ROS providencia dois modos de ”pedir” tarefas: as ações e os serviços. Os serviços, ao contrário das ações, são maioritariamente usados para realizar tarefas de curta duração. Contudo, a implementação de ações permite um controlo da ação através de mensagens do tipo *feedback* e permite um cancelamento da mesma. Visto que o intuito é controlar as tarefas (de duração consideravelmente longa) a realizar, ou seja, a aproximação à bancada, a orientação da plataforma e o processo de *bin-picking*, foram implementadas ações para cada uma delas. Assim, o `integrated_referee` é um cliente de ações que controla temporalmente a sua execução. Após o arranque, o nó espera pela inicialização de todos os servidores de ações necessários ao processo, isto é, caso um dos servidores não inicie, o sistema fica a aguardar.

Tendo em conta que a segurança é um ponto crítico em espaços de co-existência entre humanos e máquinas, o controlo da máquina por parte do operador deve ser preservado durante todo o processo. Para tal, o árbitro `integrated_referee` verifica reiteradamente se o botão *Dead man's switch* (botão 6 da figura 6.2) está pressionado e, caso não

esteja, interrompe imediatamente todas as ações em execução. Para o efeito, salienta-se a necessidade do uso de *threads* para controlar o processo de tarefas e verificar o estado dos botões do *gamepad*, simultaneamente.

As tarefas controladas pelo `integrated_referee` podem ser percecionadas de forma mais clara com o fluxograma presente na figura 6.4.

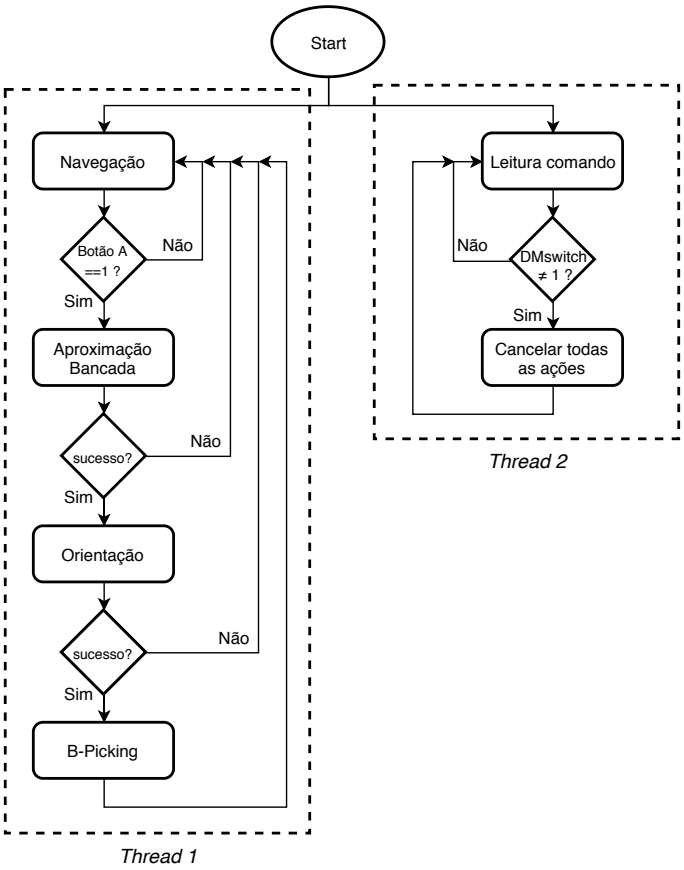


Figura 6.4: Fluxograma de funcionamento do `integrated_referee`.

6.3 Estado 1: Navegação

Conforme referido no capítulo 4, existem dois modos de navegação possíveis: o modo semi-manual e o modo automático. A comuta entre modos é controlada pelo nó árbitro, `r_hybrid`, que verifica o estado dos botões do comando, de forma a preservar a segurança no meio envolvente. Desta forma, após o utilizador escolher um ponto no mapa para navegação autónoma, o modo automático apenas é executado quando o botão 6, designado por *Dead man's switch*, é pressionado. Caso o botão 9 esteja também pressionado, o controlo 2D do manipulador móvel é assegurado pelo modo semi-manual, ou seja pelos *joysticks* 1 e 3 (figura 6.2).

No entanto, como o nó `r_hybrid` foi concebido apenas para controlo bidirecional, prevalecia sempre um dos modos: semi-manual ou automático. Assim, caso o robô se encontrasse parado, e nenhum ponto no mapa fosse selecionado para cálculo de trajetórias

do modo automático, este nó enviava constantemente velocidades nulas para os motores. Como é de prever, este constante tráfego de mensagens gera conflito aquando a execução de outros estados do robô.

Posto isto, foi necessário desenvolver um mecanismo de controlo de estados entre os nós árbitros de forma a dar a conhecer o estado definido pelo `integrated_referee` ao nó `r_hybrid`. Assim sendo, a arquitetura desenvolvida apresenta dois árbitros, dos quais o `integrated_referee` prevalece, e controla a execução do `r_hybrid`.

Este controlo é realizado através da publicação do estado, decidido pelo árbitro principal, no tópico `/RobotStatus`, de forma a que o `r_hybrid` só execute o que lhe compete quando o estado lido no mesmo tópico seja o correspondente ao estado 1 (navegação).

O pedido de uma ação por parte do `integrated_referee`, é despoletado de forma segura, impossibilitando a execução de uma ação em simultâneo com uma tele-operação da plataforma. A arquitetura inerente ao estado de navegação é apresentada na figura 6.5 e salienta-se que o nó `r_client_node` é responsável pelo envio das velocidades para os motores e pela receção da informação dos *encoders*. Esta informação é publicada no tópico `/pid_data` para o cálculo da hodometria realizado pelo `odom_data`.

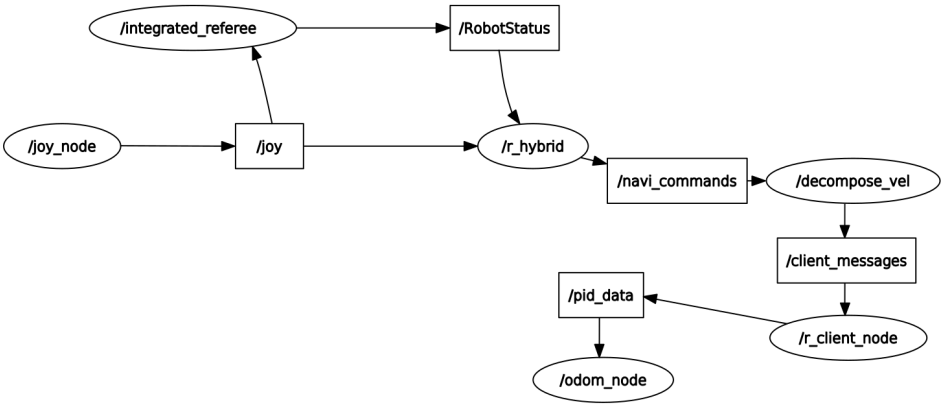


Figura 6.5: Representação simplista da arquitetura inerente ao estado 1.

6.4 Estado 2: Aproximação

De modo a realizar o *bin-picking* de forma semi-automática, foi concebida uma forma "automática" do sistema localizar uma bancada de trabalho.

A deteção da bancada é realizada através da distância lida pelo sensor laser (1D), presente na ponta do manipulador. Assim, o utilizador pode tele-operar ou usar o modo de navegação automática para posicionar a plataforma num local onde a bancada de trabalho seja alcançável, numa linha aparentemente reta e sem obstruções .

Para esse fim, foi necessário o desenvolvimento do nó `sensorRS232` que recebe as leituras provenientes do laser 1D, e publica os seus valores no tópico `/output_laser_sensor_2` a uma frequência de 20 Hz. O nó `check_feasibility`, subscreve o tópico onde são publicados os valores do laser e verifica se a distância obtida é inferior a um dado valor pré-definido. Com a configuração do manipulador durante a "procura" da bancada de trabalho, as distâncias medidas pelo laser serão, seguramente, superiores a 700 mm, uma

vez que corresponderá à distância ao solo. Assim, definiu-se que a distância limite que acionava a deteção da bancada seria de 500 mm. Este valor é facilmente ajustável, uma vez que é definido através de um parâmetro `ROS`, ou seja, aquando o lançamento do nó `check_feasibility`, no *launch file*.

Após a verificação, este nó publica no tópico `/feasibility` dois tipos de mensagens, "Platform should move." caso a plataforma se deva movimentar porque a mesa não foi detetada, ou "Platform should stop.", caso a plataforma deva parar porque houve deteção de um obstáculo.

Para o controlo do processo de aproximação, foi concebida a ação `Robot_PlatformLaserAproximation.action` e o respetivo servidor `GetPlatformLaserAproximation.cpp`. O processo só é iniciado após a receção de um `goal` por parte do servidor, e só é aceite caso não haja execução de outro. Caso exista um pedido de cancelamento da ação por parte de um cliente, o servidor está apto a processar esse pedido e a cancelá-lo. O resultado (`result`) foi definido como uma variável booleana que, caso a ação seja bem sucedida, é retornada como verdadeira, e vice-versa.

O servidor `GetPlatformLaserAproximation` publica velocidades lineares positivas no tópico `/navi_commands` até a deteção da mesa. Note-se que, quando a mesa é detetada, não basta parar de publicar velocidades, é necessário enviar pelo menos uma mensagem com velocidades nulas.

Posteriormente, esse tópico é subscrito pelo nó `decompose_vel` que publica as mensagens a enviar para o CPU2 por parte o nó cliente `r_client_node`.

A arquitetura inerente a esta ação e o seu fluxograma estão representados nas figuras 6.6 e 6.7, respetivamente.

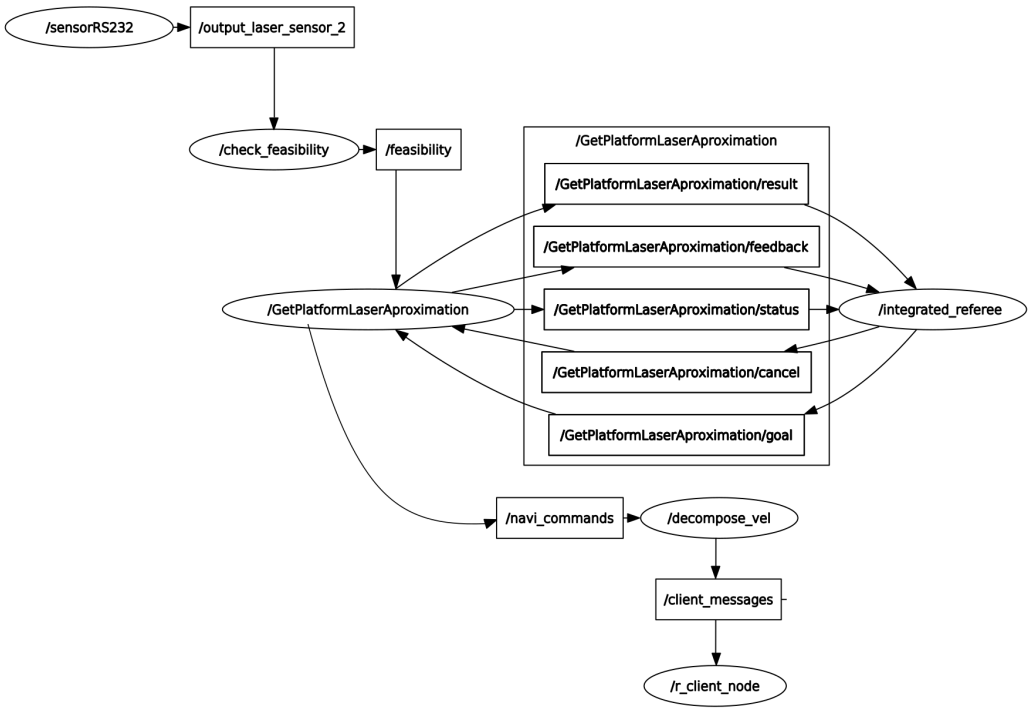


Figura 6.6: Representação da arquitetura inerente ao estado 2.

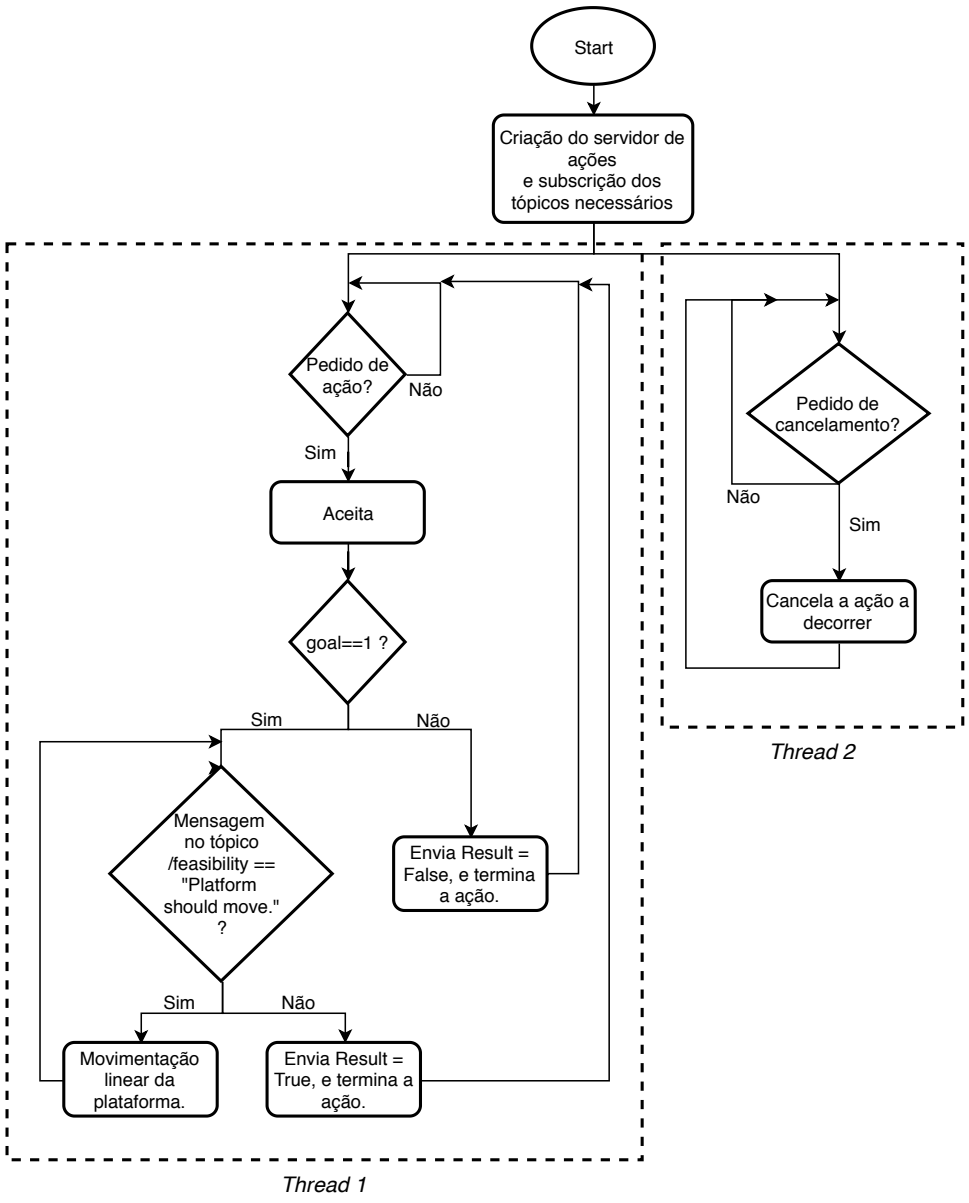


Figura 6.7: Fluxograma inerente ao servidor de ações correspondente ao estado 2.

6.5 Estado 3: Orientação

Como referido anteriormente, o manipulador móvel usado para o projeto contém 2 rodas giratórias para conceder estabilidade à plataforma.

Durante a aproximação à bancada de trabalho, as rodas giratórias podem conferir uma direção ligeiramente diferente da direção inicial onde a ação foi despoletada. É possível observar um exemplo da posição da plataforma relativa à mesa, após a aproximação, na figura 6.8.

Tendo em conta que a plataforma pode chegar enviesada ao local de *bin-picking*, foi criada uma ação para orientar o manipulador móvel de forma a que a placa posterior da plataforma fique paralela à mesa de trabalho (figura 6.9).

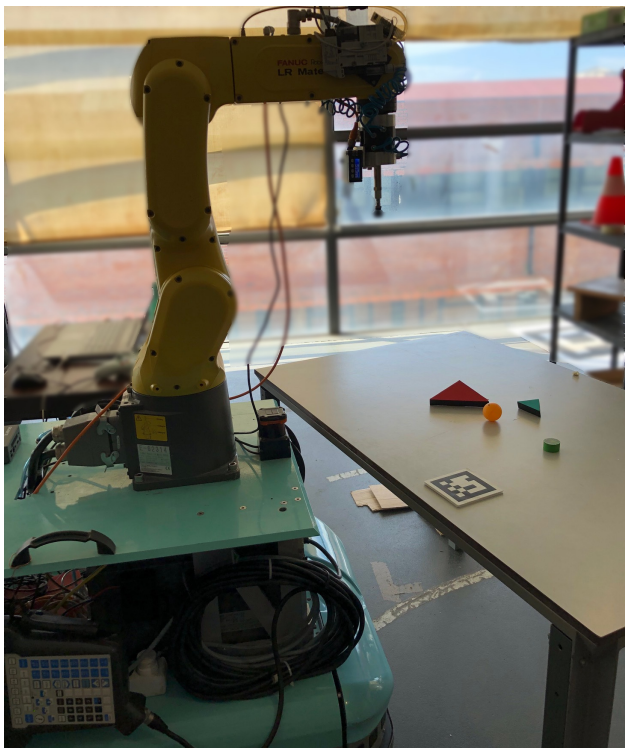


Figura 6.8: Exemplo da posição da plataforma após a aproximação da bancada.

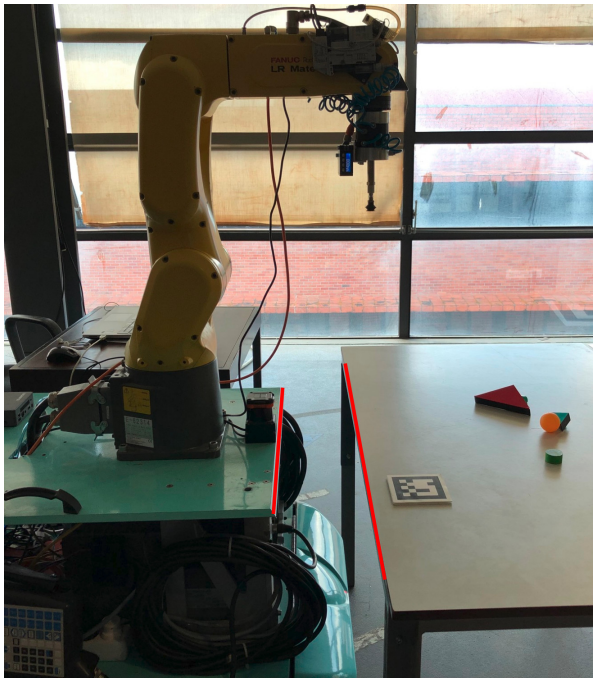


Figura 6.9: Representação da orientação pretendida da plataforma móvel.

Esta necessidade de orientação deve-se, essencialmente, a dois fatores. O primeiro, consiste na impossibilidade da pré-definição de uma posição do manipulador para análise

da bancada, de modo a iniciar o *bin-picking*. A segunda deve-se à impossibilidade de implementar um processo de *bin-picking* mais robusto, no qual poderá existir a necessidade de movimentação da plataforma de forma a aumentar o alcance do manipulador.

Para a sua implementação, foi usado um marcador *fiducials* de pequena dimensão, que define a orientação da mesa. Desta forma, pretende-se determinar a relação entre o referencial base da plataforma móvel e um elemento externo (aruco), posicionado na mesa de trabalho.

O pacote `Fiducials` [67] fornece ferramentas que permitem determinar a posição e orientação do robô, utilizando marcadores *fiducials*, como por exemplo, os arucos. Como já foi referido, a posição é definida pelo vetor de translação ($t_{[3\times1]}$) e uma matriz de rotação ($R_{[3\times3]}$) ou por um quaternião. Através do pacote `aruco_detect`, a posição do marcador relativamente à câmara, ${}^{Camera}T_{aruco}$, pode ser estimada a partir de uma câmara RGB e é publicada no tópico `fiducial_transforms`.

Concebeu-se a ação `Robot_PlatformOrientation.action` e o respetivo servidor `Get-PlatformOrientation.cpp` de forma a realizar a tarefa de orientação.

O servidor subscreve constantemente a posição, no referencial `camera_rgb_optical-_frame`, do aruco detetado e publica a transformação do referencial da câmara para um novo referencial criado (`fiducial_object`). É possível visualizar o referencial criado e a deteção do aruco na figura 6.10.

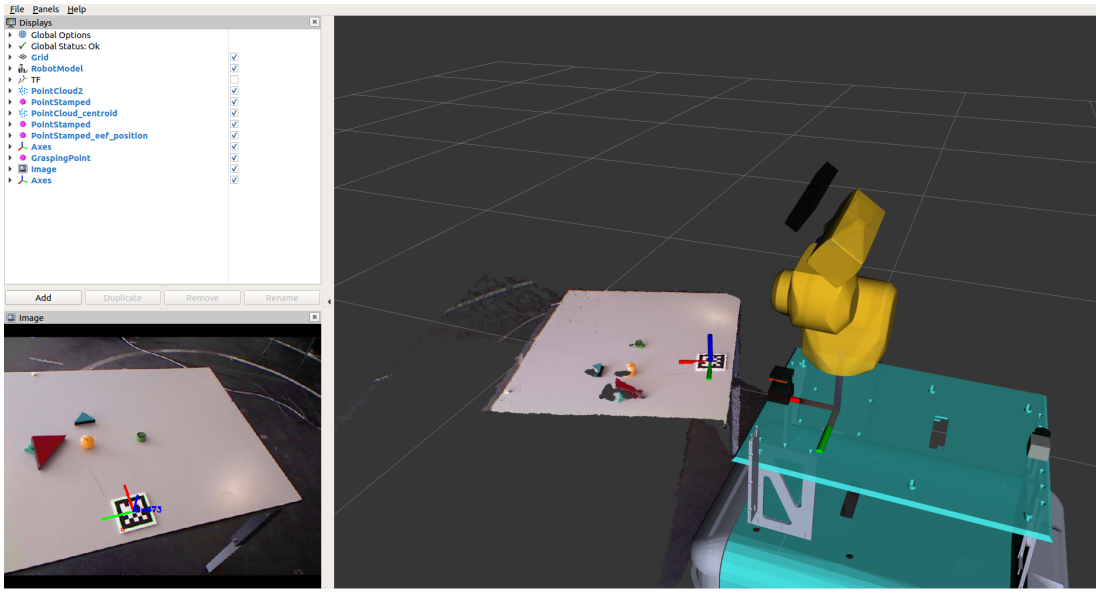


Figura 6.10: Representação do processo de orientação da plataforma móvel.

Assim, quando a ação é solicitada, faz-se uso da classe `tf` para conhecer a transformação do `robot_base_link` para o `fiducial_object`. Caso o ângulo horizontal não esteja compreendido entre $]-0.05, +0.05[$ rad, a plataforma é rodada com base na velocidade angular. Esta velocidade é publicada no tópico `/navi_commands` para posteriormente ser processada e enviada para os motores por parte do CPU2.

O processo de orientação só é iniciado após a receção de um `goal` por parte do servidor e só é aceite caso não exista outro a decorrer. Caso exista um pedido de cancelamento da ação por parte de um cliente, o servidor está apto a processar esse pedido e a cancelá-lo.

O resultado (`result`) foi definido como uma variável booleana, que caso a ação seja bem sucedida, é retornada como verdadeira, ou em caso contrário, como falsa.

A arquitetura inerente a esta ação e o seu fluxograma estão representados nas figuras 6.11 e 6.12, respetivamente.

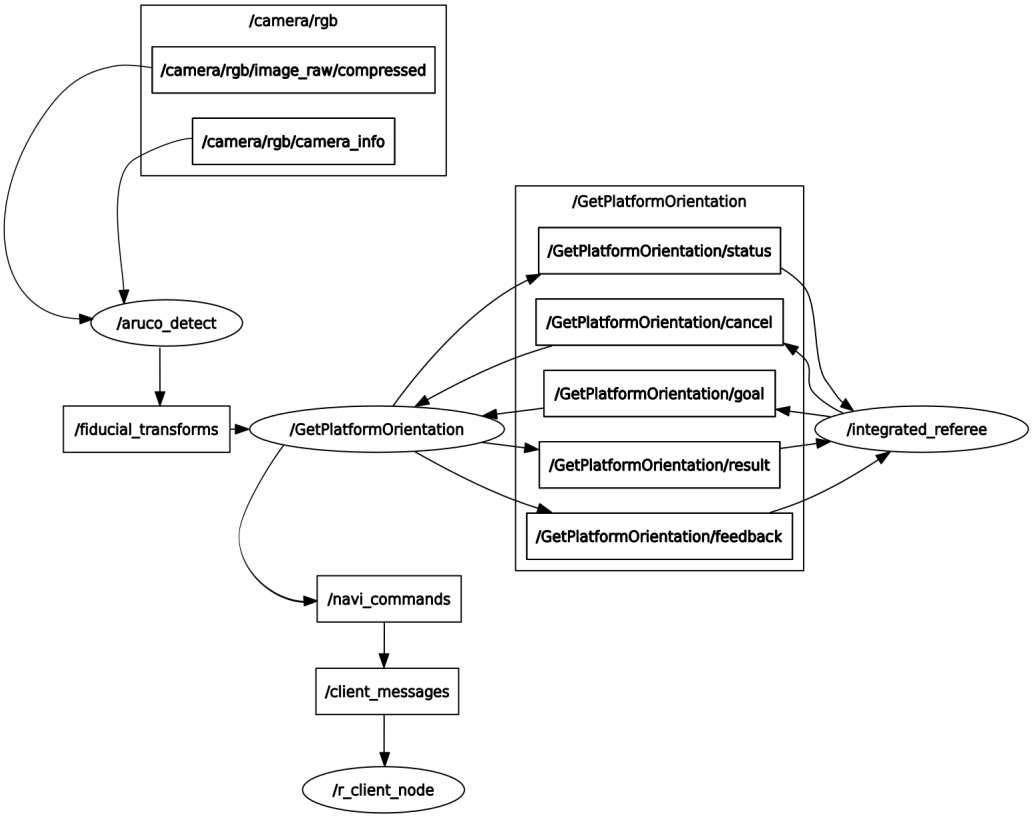


Figura 6.11: Representação da arquitetura inerente ao estado 3.

6.6 Estado 4: Bin-Picking

O processo de *bin-picking* desenvolvido anterioremente (em [13]) não estava completamente fluído e apresentava bastantes limitações para sua utilização direta. Durante a execução do programa são lançados e terminados nós para executar uma tarefa específica, isto é, os nós responsáveis pela localização dos centróides dos objetos, pelo cálculo dos pontos de aproximação e pela aquisição das leituras do sensor laser, são iniciados e terminados após publicarem os valores necessários. Estes nós foram concebidos para publicar os valores apenas uma vez, para um único objeto, ou seja, após essa tarefa os nós necessitam de ser reiniciados. Apesar desta metodologia não ser a mais adequada, o programa estava funcional e a tarefa de *bin-picking* (de um objeto) era executada com êxito. É de salientar que, durante o processo o programa desenvolvido espera sucessivos *inputs* do teclado por parte do utilizador, de forma a aumentar a segurança, uma vez que visa confirmar todos os passos do processo.

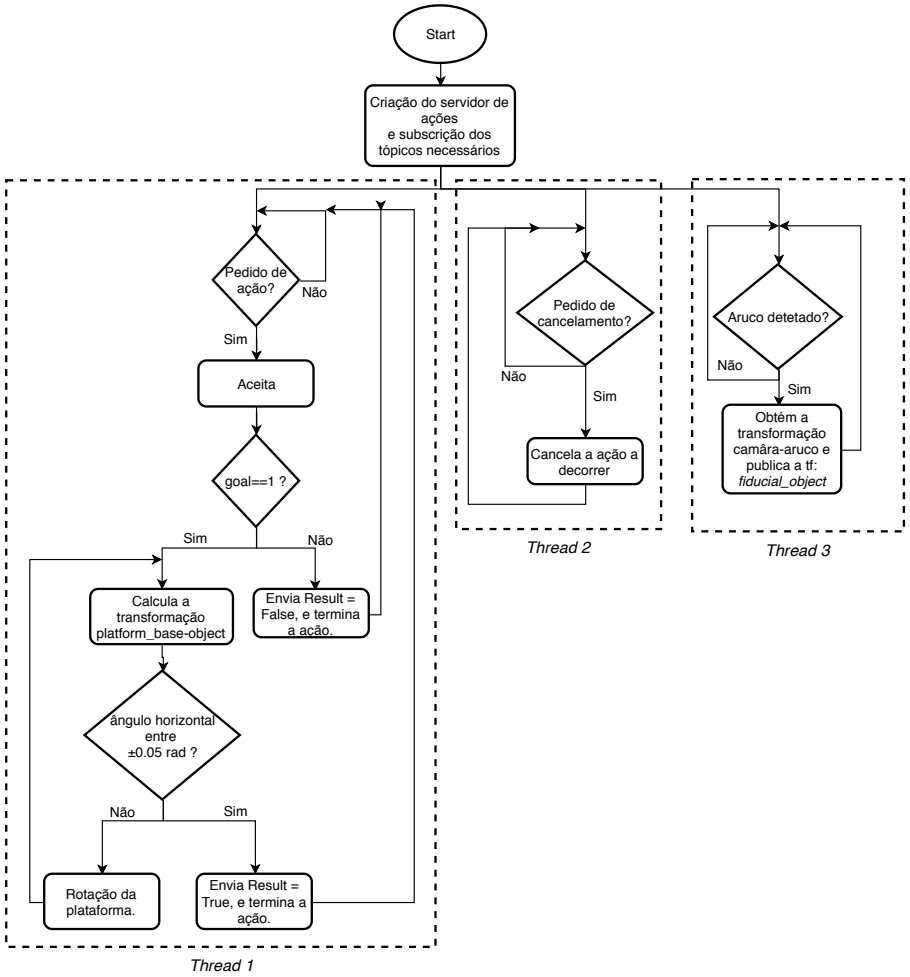


Figura 6.12: Fluxograma inerente ao servidor de ações, correspondente ao estado 3.

Para melhor perceção, é apresentado um fluxograma representativo do programa existente na figura 6.13.

Para o estado 4, concebeu-se a ação `Robot_binpicking.action` e o seu servidor `GetBinPicking.py`. Contudo, a implementação do processo de *bin-picking* associada à *callback* para realização da ação foi impossível de implementar, uma vez que o `ROS` apenas permite iniciar e terminar nós na *main thread*.

De forma a implementar a ação e, consequentemente, aumentar a robustez do processo de *bin-picking*, foi necessário alterar todos os nós inerentes à tarefa. Para isso, foram criados novos ficheiros de forma a não interferir com o trabalho de [13].

A solução necessitava de manter todos os nós ativos após o término da sua tarefa e a possibilidade de repetir tarefas sempre que necessário, sem reiniciar o nó. Dado que as tarefas a realizar são de curta duração, e não existe a necessidade de cancelamento das mesmas, foram criados serviços para realização das tarefas.

Criou-se o nó de serviço, `objDetection_v2.cpp` e o serviço `get_targets_pose` que, após a receção do pedido, executa o cálculo de um centróide e da respetiva normal, e publica os valores nos tópicos `/cloud_centroid` e `/cloud_centroid_normal`, respetivamente. Posteriormente, este nó fica à espera de um pedido de serviço para repetir todo

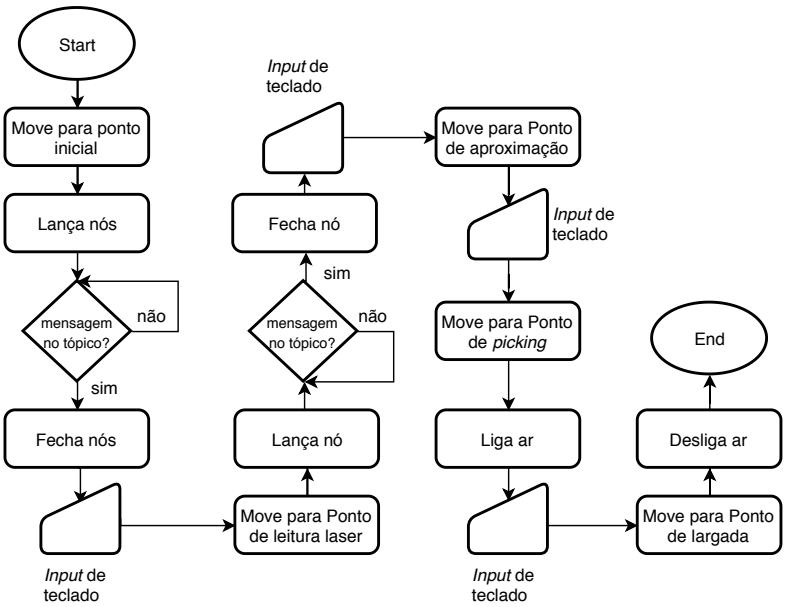


Figura 6.13: Fluxograma da tarefa de *bin-picking* concebida em [13].

o processo e assim sucessivamente.

O nó `pointTFtransfer_v2.cpp` é responsável pela subscrição dos tópicos anteriores: sempre que recebe uma mensagem nos mesmos tópicos, calcula os pontos de aproximação e publica-os no tópico `/targets_pose`.

Deste modo, o cliente de serviços, após a receção das respostas aos pedidos, move o manipulador para o primeiro ponto de aproximação onde solicita um novo serviço, `get_laser_average`, para obter a leitura do sensor laser. Sempre que o nó de serviço, `sensorRS232_v2_service.cpp`, recebe um pedido, realiza dez leituras do laser, calcula a sua média e retorna o valor no tópico `/output_laser_sensor_average`.

Com a distância precisa, do manipulador ao objeto, o cliente de serviços, realiza o cálculo do novo ponto de *grasping*, move o manipulador até esse ponto e ativa a sucção através da publicação do respetivo valor no tópico `io_client_messages`. Esse tópico é subscrito pelo nó, `vs_IO_client`, responsável pela ativação externa do *gripper*. O manipulador é movido até ao ponto de largada do objeto, e o *gripper* é desativado.

A execução da ação termina com êxito e retorna o resultado *"true"* ao cliente. Caso a ação não seja terminada com êxito, retorna o resultado *"false"*.

É de salientar que durante a execução da ação, o utilizador só necessita de confirmar a movimentação do robô para o ponto de leitura do laser, uma vez que esse ponto condiciona todo o resto do processo. Caso o ponto esteja representado corretamente, confia-se na fiabilidade e segurança do resto do sistema. Para isso, basta pressionar o botão B (da figura 6.2) após a solicitação de confirmação do programa.

A arquitetura simplificada inerente a esta ação é apresentada na figura 6.14. Os nós e tópicos relativos ao `moveit` foram substituídos pelo bloco verde, uma vez que complicavam a representação da arquitetura.

Apresenta-se também na figura 6.15, um fluxograma representativo do processo de *bin-picking* a partir do momento que o servidor aceita o *goal*.

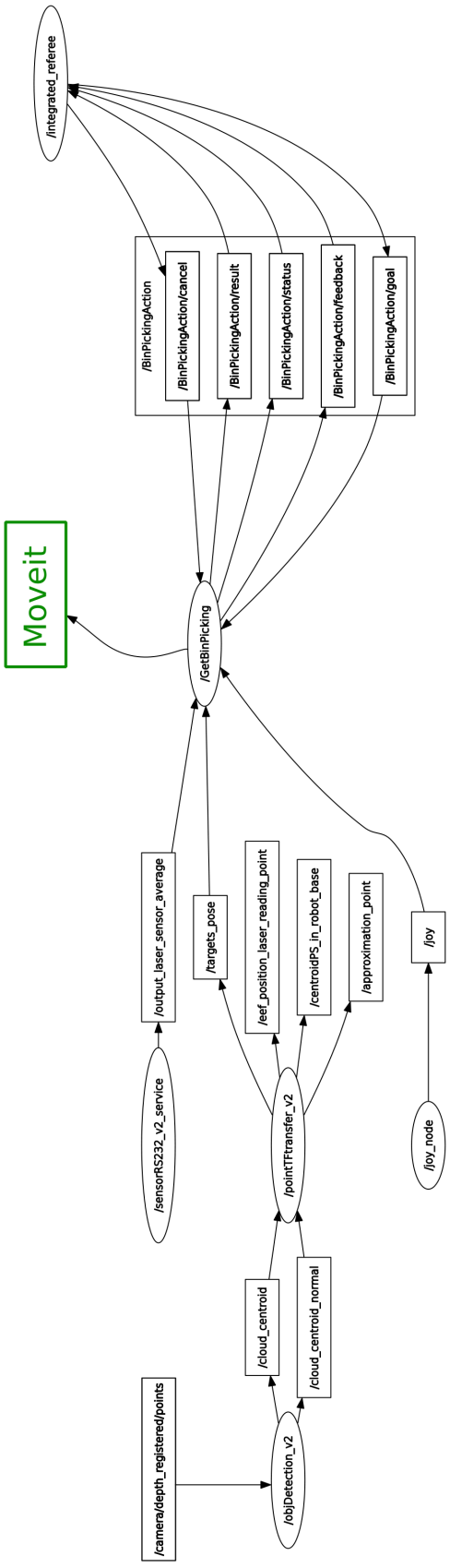


Figura 6.14: Representação da arquitetura inerente ao estado 4.

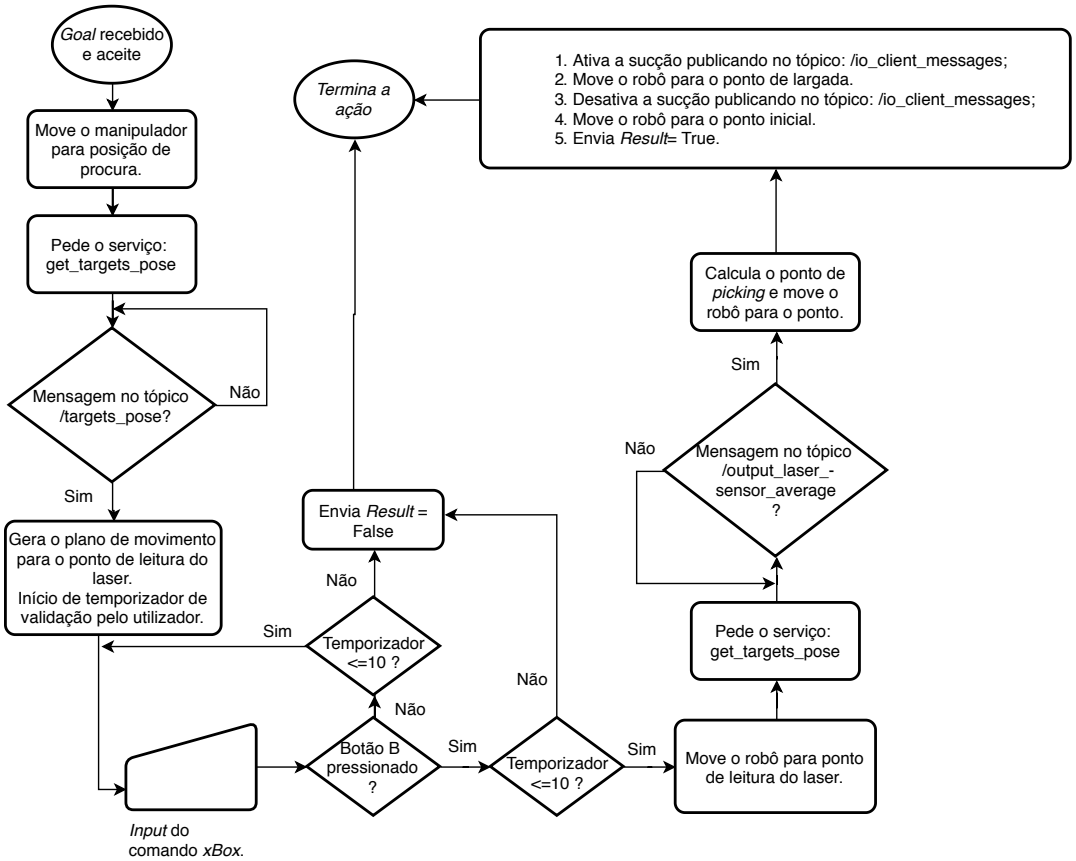


Figura 6.15: Fluxograma inerente ao servidor de ações correspondente ao estado 4.

6.7 Controlo Remoto

De forma a preservar a segurança, para efetuar a navegação e a supervisão do sistema em tarefas autónomas, o utilizador necessita de uma interface para o controlo remoto. Assim, é disponibilizado um comando *XBox* ao utilizador e uma interface gráfica, para operar o manipulador móvel por *wi-fi*.

6.7.1 Interface

O `Rviz` é uma ferramenta de visualização 3D, que permite observar vários tipos de dados e informação em `ROS`, não só *offline*, mas também em tempo real. O seu uso torna-se útil, não só para o *debug* do projeto em fase de desenvolvimento, mas também para a visualização do estado do robô sem necessidade de o observar fisicamente.

Deste modo, a interface desenvolvida para controlo remoto foi baseada nesta ferramenta, e pode ser visualizada na figura 6.16.

Do lado esquerdo apresenta-se o mapa reconstruído do ambiente com base nos lasers LRF, onde se tem a visualização bidimensional dos obstáculos à cota dos lasers. É também possível visualizar um *streaming* da câmara RGB, que dá uma melhor perceção ao utilizador de outros potencias alvos de colisão durante a tele-operação. De forma a iniciar a navegação autónoma, basta o utilizador selecionar a opção *2D Nav Goal*, e de

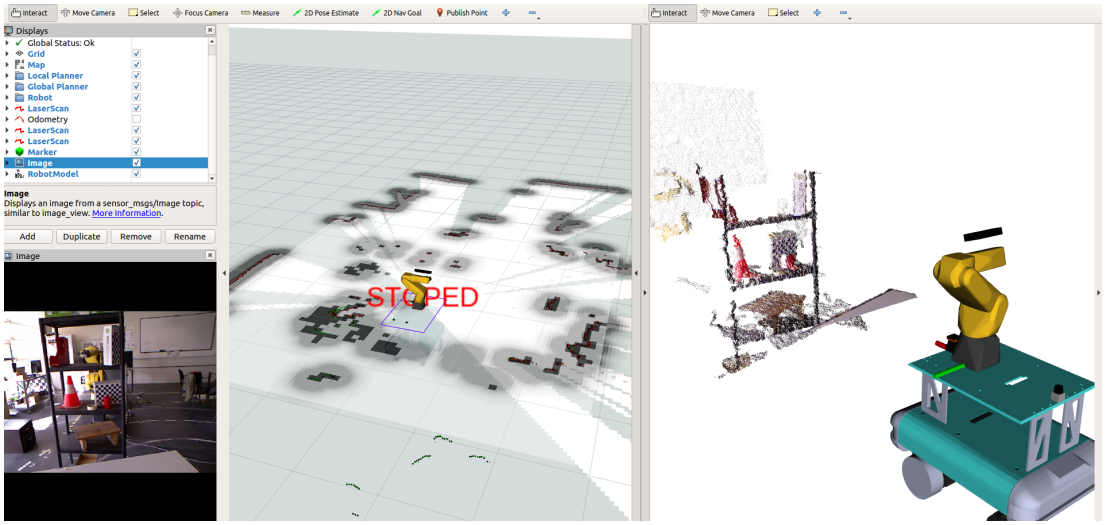
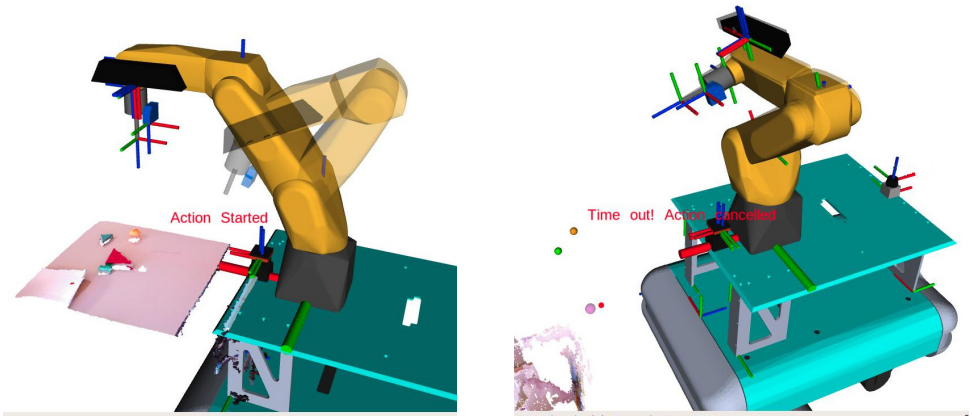


Figura 6.16: Interface para controlo remoto.

seguida, pressionar o local de destino pretendido e escolher a direção da pltaaforma nesse mesmo ponto.

De forma a supervisionar o processo e auxiliar o utilizador, especialmente, na tarefa de *bin-picking*, é apresentada do lado direito, uma vista mais pormenorizada do ambiente com base na informação proveniente da câmara IR. São representados os pontos de aproximação, o planeamento dos movimento do robô antes da execução, bem como algumas perguntas e informações do processo em forma de texto (figura 6.17).



(a) (b)

Figura 6.17: Representação de exemplos de informação apresentada na interface gráfica, durante o processo de *bin-picking*.

6.7.2 Controlador Interativo

Como já foi referido, para o controlo remoto do robô, o utilizador tem como auxílio o comando *XBox*. As funções definidas para cada um dos seus botões encontra-se descrita na tabela 6.1.

Tabela 6.1: Descrição da função dos botões do comando *XBox* da figura 6.2.

Botão	Função
1	Controlo da velocidade angular. Movimento para a esquerda e direita no eixo horizontal.
2	Não utilizado.
3	Controlo da velocidade linear. Movimento para cima e para baixo no eixo vertical.
4	Não utilizado.
5	Diminuição da velocidade máxima.
6	Botão de segurança (*Dead man's switch*)
7	Liga o comando. Pressionar durante 1s.
8	Não utilizado.
9	Botão de segurança para o controlo semi-manual.
10	Aumento da velocidade máxima.
A	Início do processo semi-autónomo de *bin-picking*.
B	Confirmação dos pontos de aproximação durante o *bin-picking*.
X	Não utilizado.
Y	Não utilizado.

6.7.3 ROS Distribuído

O `ROS` é projetado de forma a conseguir lidar com arquiteturas distribuídas, isto é, os nós não fazem pressupostos sobre o local onde a rede é executada, permitindo que a computação seja realocada de modo a responder aos recursos disponíveis [68]. Desta forma, se a velocidade de comunicação for assegurada, é possível executar vários programas em diferentes máquinas sem afetar o desempenho das ferramentas criadas.

Existem duas formas de correr vários programas em máquinas diferentes. Na primeira, existe apenas uma unidade de controlo *Master*, onde é executado o `roscore`, em que todos os processos executados em máquinas diferentes comunicam com o *Master*. Na segunda, existe vários *Master's* com trocas de informação, utilizando pacotes concebidos para o efeito, como por exemplo, o `wifi_comm`.

Para solucionar a limitação de alcance do comando *xBox*, usado para a tele-operação, optou-se pela utilização do ROS distribuído em duas máquinas. Assim, o utilizador necessita de um computador com `ROS`, onde é executado o nó responsável à leitura dos comandos, a fim de comunicar com a unidade de principal de processamento do robô.

Através do *router* existente concebeu-se uma rede *wi-fi* própria para o robô designada de `Robonuc_5G`. Assim o utilizador apenas tem que se conectar à rede, instalar o `ROS` e o pacote *joy*, e aceder ao computador do robô por *Secure Shell (ssh)*.

Para aceder ao computador do robô e utilizar o ROS distribuído devem ser inseridos no terminal os seguintes comandos:

```
ssh robuter@192.168.0.145
export ROS_IP=192.168.0.145
export ROS_MASTER_URI=http://192.168.0.145:11311
roslaunch robonuc_integration robonuc_integration.launch
```

A definição do ROS_Master_URI define o endereço da máquina onde irá correr o processo *MASTER*, ou seja, irá permitir que outros equipamentos identifiquem e comuniquem com os nós do manipulador móvel. O ROS_IP define o ip da maquina onde se pretende executar os nós posteriores. Assim, através do robonuc_integration.launch são iniciados (na maquina do robô) todos os nós necessários ao controlo do *hardware*, à exceção do nó joy e dos nós inerentes à inicialização da interface.

Para a execução dos restantes nós na máquina do utilizador, num novo terminal, deverão ser introduzidos os seguintes comandos:

```
export ROS_IP=192:168.0.xxx (ip do utilizador)
export ROS_Master_URI=http://192.168.0.145:11311
roslaunch robonuc_integration tele_op.launch
```

Assim, o tele_op.launch é responsável por lançar a interface gráfica o nó responsável pela leitura do comando *XBox*.

Para constantes ligações frequentes sugere-se a criação de uma *key* através do *ssh-keygen* e a alteração do ficheiro *.ssh/config* para:

```
Host robuter
        HostName robuter-pc.local
        User robuter
```

Desta forma, para iniciar a comunicação *ssh* basta colocar no terminal, *$ ssh robuter*, descartando a posterior etapa de introdução da *password*.

Capítulo 7

Cinemática Integrada

Para que os dois sistemas (plataforma móvel e manipulador) alcancem uma "ação cooperativa", isto é, de forma a que exista uma compensação mútua para uma determinada tarefa, este capítulo concentra-se na formulação da cinemática integrada do sistema. Assim, neste capítulo é estudada a literatura na abordagem deste problema, são propostos dois modelos de cinemática integrada e são analisadas ferramentas para a sua aplicabilidade no hardware real.

7.1 Problemática e Trabalhos relacionados

Contrariamente aos manipuladores fixos, a extensão do espaço de trabalho por parte dos manipuladores móveis tem vindo a aumentar a necessidade da sua aplicação, uma vez que apresentam um enorme potencial para uma grande variedade de tarefas. Contudo, este tipo de sistemas integrados (com ou sem restrições holonómicas) apresenta questões que aumentam com a complexidade do sistema, das quais se destacam [69]:

- Quais os modelos exatos da cinemática e dinâmica?
- Como planear eficazmente a trajetória de movimentos?
- Como descrever o controlo do movimento/força e posição/força quando o manipulador móvel necessita de interagir com o ambiente?
- Como coordenar múltiplos manipuladores móveis de forma a cooperarem para atingir eficazmente tarefas desejadas, e como aumentar a eficiência de múltiplos robôs?

Ao longo dos anos, o foco da investigação tem recaído sobre a modelação, controlo e coordenação de manipuladores móveis, abrangendo, dentro de cada tema, vários domínios de pesquisa e diferentes abordagens. Enquanto uns têm sido intensivamente estudados, outros são recentes e pouco documentados. Questões relacionadas com os temas incluem a modelação cinemática e dinâmica, controlo de sistemas não-holonómicos, planeamento de trajetórias considerando movimento e manipulação, controlo de movimento, posição e força, bem como a coordenação e interação de múltiplos manipuladores móveis. No entanto, a literatura existente sobre questões fundamentais é limitada [69].

Devido às restrições cinemáticas impostas pela base móvel, o manipulador móvel é um sistema não-holonómico. Além disso, a complexidade da estrutura física, a dinâmica

acoplada entre a base móvel e o manipulador, e a mobilidade da base móvel através de rodas são algumas das características que aumentam substancialmente a dificuldade do projeto e o controlo do sistema [70] [2].

Este tipo de sistemas (não-holonómico) complica consideravelmente o planeamento de movimentos, uma vez que não permite um *feedback* contínuo ao longo do movimento. Desta forma, o planeamento de movimento necessita de ser em malha aberta, temporalmente variável ou descontínuo. Usualmente, o planeamento de movimento é dividido em duas partes: planeamento do caminho e planeamento de trajetórias. Enquanto que o primeiro consiste na definição das contínuas posições de configuração (desde o ponto inicial ao ponto "objetivo"), o segundo consiste na parametrização temporal do caminho gerado. Assim, uma estratégia para planeamento de movimento é inevitavelmente, em ciclo aberto. Metodologias alternativas incluem *feedback* não suave (*non smooth feedback*) e esquemas de tempo variáveis, embora se concentrem geralmente no rastreamento de um caminho holonómico existente [71].

Outro dos problemas consiste em definir o movimento de um manipulador móvel entre duas configurações arbitrarias e usar as suas redundâncias para atingir outros objetivos, como o desvio de obstáculos e/ou evitar singularidades.

Normalmente, os manipuladores robóticos e as plataformas móveis têm vindo a ser estudados separadamente. Assumindo que os dois subsistemas têm dinâmicas diferentes, o movimento do sistema pode ser decomposto no controlo do movimento da plataforma e na evolução do manipulador [2]. Este tipo de alternância gera resultados satisfatórios para tarefas específicas que podem naturalmente ser decompostas em sub-tarefas. Contudo, é interessante considerar a evolução simultânea dos dois sub-sistemas e coordená-los de forma a, por exemplo, aumentar a sua eficiência na realização de tarefas, aumentar a sua manipulabilidade ou executar outras tarefas mais complexas.

Tendo em conta as propriedades do sistema global, a redundância criada pelo aumento do número de graus de liberdade é uma das mais importantes. Em [72], os graus de mobilidade são abordados do mesmo modo dos do manipulador e para a resolução da redundância é pré-definida uma das variáveis de junta. Assim, no exemplo apresentado em [72], a mobilidade segundo o eixo X é então substituída por uma junta prismática e é fixada uma das juntas do braço róbotico de forma a suprimir a redundância existente. Esta propriedade (redundância), característica dos manipuladores móveis, permite usar os graus de liberdade adicionais para a realização de tarefas secundárias como, por exemplo, evitar colisões ou aumentar a manipulabilidade. Em [73], o foco é a resolução das redundâncias cinemáticas com a otimização de posição e configuração dos sistemas durante as tarefas, isto é, a escolha dos graus de redundância é baseada em otimizações de forma a evitar obstáculos, aumentar a manipulabilidade, diminuir a(os) forças/momentos nos atuadores ou todos estes fatores conjugados.

Em [74], o planeamento de trajetórias de manipuladores móveis, para execução de múltiplas tarefas, é retratado como um problema de otimização em que as variareis da plataforma são separadas das variáveis do manipulador na função custo. Nesta referência, o objetivo era minimizar a energia e/ou o tempo de realização de uma sequência de tarefas.

Devido à complexidade deste tipo de sistemas, o controlo é das áreas mais investigadas e, são exemplos os trabalhos: [75], [76], [77]. Mais recentemente, o trabalho desenvolvido em [78] comprova a estabilidade do método proposto, que faz uso de algoritmos de controlo baseados num simples Jacobiano transposto e na teoria de estabilidade de

Lyapunov.

Na dissertação [79] é investigada a modelação, o controlo, e a coordenação dos manipuladores móveis. Um dos problemas do controlo destes sistemas deve-se ao tempo de resposta superior das plataformas móveis comparativamente com os manipuladores. Para sua solução foi desenvolvido um algoritmo de modo a controlar os dois subsistemas: a manipulação e mobilidade dos manipuladores móveis.

Em [71], é investigado o problema de planeamento de movimento para este tipo de sistemas e propoem uma solução baseada na lei de *Full state discontinuous feedback*, que garante a convergência da posição final, e evita as singularidades e obstáculos. Ao invés do uso de algoritmos clássicos de cálculos de trajetórias, usam uma abordagem baseada num campo de potencial, que requer um modelo topológico da geometria do robô e do ambiente de trabalho.

Contudo, devido à dificuldade da modelação dinâmica do sistema integrado, existem abordagens utilizando redes neuronais para efetuar o controlo de manipuladores móveis. Em [80] são apresentadas e discutidas algumas destas abordagens e em [70] é apresentada uma metodologia de controlo das posições de juntas, baseado em redes neuronais, no qual são propostos dois controladores: um para o controlo da plataforma e outro para o manipulador. A dinâmica do manipulador móvel é pré-definida como totalmente desconhecida e é identificada pela rede neuronal usando uma dinâmica de erro baseada num conjunto de sub-funções *Lyapunov* cuidadosamente escolhidas através de uma analise do espaço de junta.

De modo a estabelecer um controlo adequado é essencial uma modelação dinâmica precisa. Assim, em [81] é apresentado um estudo intensivo das propriedades dinâmicas de sistemas robóticos para coordenação dos sistemas.

Na década de 2000 começou a surgir o conceito de configuração endógena, exclusivamente direcionado para manipuladores móveis e fundamentado na teoria de controlo. Este conceito surge como resposta ao problema do planeamento de movimento e controlo da plataforma simultaneamente com o manipulador, ou seja, de modo a alcançar uma determinada posição e orientação do *end-effector* num certo instante de tempo. O trabalho [82] esclarece esta problemática e introduz os conceitos básicos de configuração endógena e do Jacobiano analítico. Fazem uso destes conceitos os trabalhos [83], [84] e [85].

Surgem também, abordagens para o controlo de múltiplos manipuladores móveis de forma a cooperarem em certas tarefas (exemplo em figura 7.1). A modelação do espaço de trabalho e a elaboração de algoritmos de controlo obtido através da combinação dinâmica de cada sistema individual envolvido podem ser consultados em [86], [87] e [88].

Desta forma, é notório o grande número de trabalhos referentes a este tema. Embora existam estratégias que dão resposta a alguns dos problemas enumerados, esta área continua em permanente investigação. Esta realidade deve-se ao facto de não existir uma solução definitiva e consensual que dê resposta a todos os problemas, nomeadamente em sistemas de controlo de elevada precisão.

É possível concluir que as condições impostas pela combinação dos dois sistemas dificultam a sua formalização integrada, uma vez que as suas restrições complicam severamente as formulações matemáticas da dinâmica, controlo, cinemática, etc. Desta forma, o controlo e a análise dinâmica destes sistemas complexos requerem o uso de técnicas mais sofisticadas.

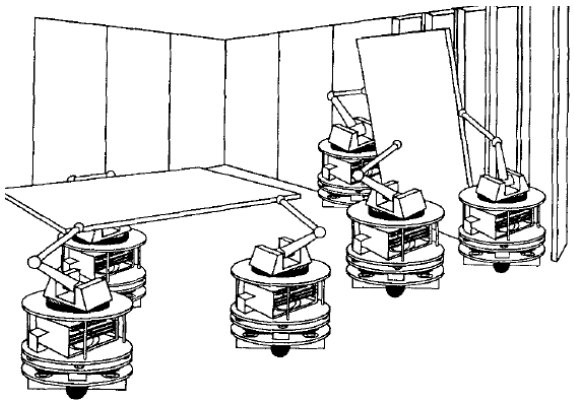


Figura 7.1: Representação de cooperação de manipuladores móveis na realização de tarefas [86].

7.2 Modelos de cinemática propostos

Como já foi exposto na secção anterior, as abordagens típicas para a coordenação de movimentos de sistemas redundantes envolvem o uso de matrizes pseudo-inversas para resolver sistemas degenerados de equações lineares, ou fazem uso de certos critérios para restringir o sistema. Esses algoritmos são essencialmente impulsionados por considerações cinemáticas ignorando a interação dinâmica entre o *end-effetor* e os movimentos internos do manipulador.

Com o intuito de perceber o problema retratado na literatura encontrada, optou-se por estabelecer o modelo integrado da cinemática direta do conjunto. Este modelo é posteriormente testado utilizando a cinemática diferencial para planeamento da trajetória.

7.2.1 Modelação cinemática

A cinemática de um robô consiste no estudo do conjunto de relações entre posições, velocidades e acelerações. Desta forma, para descobrir essas relações, é necessário determinar a posição da garra do manipulador a partir das posições de juntas. Com esse intuito, neste trabalho optou-se por estudar e implementar duas abordagens diferentes, uma considerando 3 graus de liberdade para a plataforma e outra considerando apenas 2 graus de liberdade, designadas por Abordagem 1 e Abordagem 2, respetivamente.

Abordagem 1

A primeira abordagem utilizada une a abordagem tradicional da cinemática direta dos manipuladores robóticos com a cinemática de um robô móvel, e é referida e usada em vários trabalhos encontrados, tais como [89], [79], [90] e [76].

A posição (X_m, Y_m, Z_m) e orientação (A_m, B_m, C_m) do *end effector* do manipulador relativamente à sua base, $\vec{P}_{ee\_m} = [X_m, Y_m, Z_m, A_m, B_m, C_m]^T$, pode ser calculada através da metodologia tradicional de *Denavit-Hartenberg* (DH). Enquanto que o centro dinâmico da plataforma móvel é representado diretamente pelo vetor $\vec{P}_p = [X_p, Y_p, h_p, \theta_p]$, em que X_p, Y_p h_p representam a posição e θ_p a orientação relativamente ao referencial global. Note-se que a altura da plataforma (h_p) é constante e foi considerada nula para simplificação.

Sabendo que o manipulador robótico está sobreposto ao referencial da plataforma e com auxilio de trigonometria básica é possível representar o vetor das coordenadas do *end effector*, $\vec{P_{ee}} = [X_{ee}, Y_{ee}, Z_{ee}, A_{ee}, B_{ee}, C_{ee}]^T$, pelas equações 7.1. É de salientar o uso das designações A, B, C para os ângulos de Euler (*Roll*, *Pitch* e *Yaw*, respetivamente).

$$\begin{cases} X_{ee} = X_p + X_m \times \cos\theta_p - Y_m \times \sin\theta_p \\ Y_{ee} = Y_p + X_m \times \sin\theta_p + Y_m \times \cos\theta_p \\ Z_{ee} = Z_m + h_p \\ A_{ee} = \theta_p + A_m \\ B_{ee} = B_m \\ C_{ee} = C_m \end{cases} \tag{7.1}$$

Para melhor perceção das variáveis usadas apresenta-se a ilustração na figura 7.2.

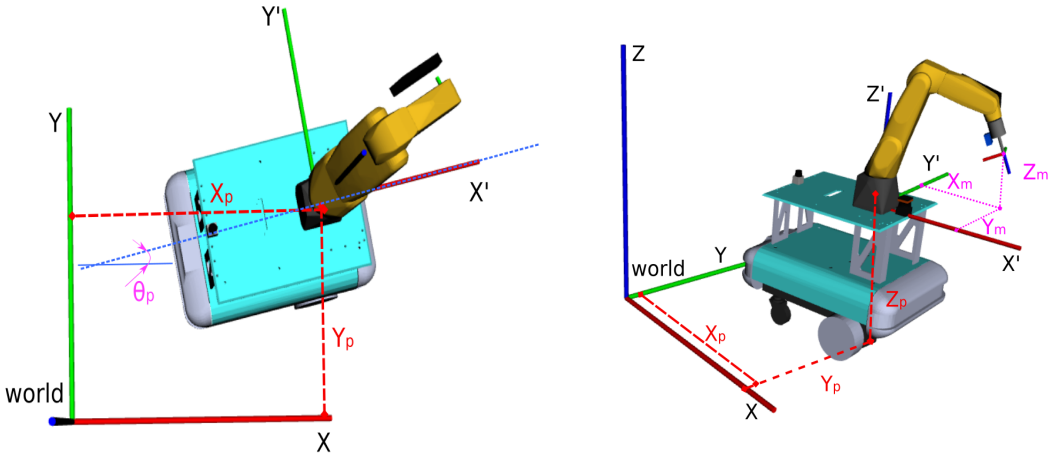


Figura 7.2: Representação ilustrativa das variáveis usadas na abordagem 1.

De modo a determinar a posição $\vec{P}_{ee\_m}$ com a metodologia D-H, foram definidas e numeradas as origens de cada elo baseadas em [91] (figura 7.3) e foi concebida a respetiva tabela D-H (tabela 7.1).

Tabela 7.1: Tabela dos parâmetros D-H usados na abordagem 1.

Elo	θ_i	a_i	d_i	α_i
1	θ_1	a_1	0	$\frac{\pi}{2}$
2	$\theta_2 + \frac{\pi}{2}$	a_2	0	0
3	θ_3	a_3	0	$\frac{\pi}{2}$
4	θ_4	0	d_4	$-\frac{\pi}{2}$
5	θ_5	0	0	$\frac{\pi}{2}$
6	θ_6	0	d_6	0

Assim, através da metodologia adotada é possível calcular a matriz de transformação associada a cada elo (com a equação 7.2) e consequentemente a transformação

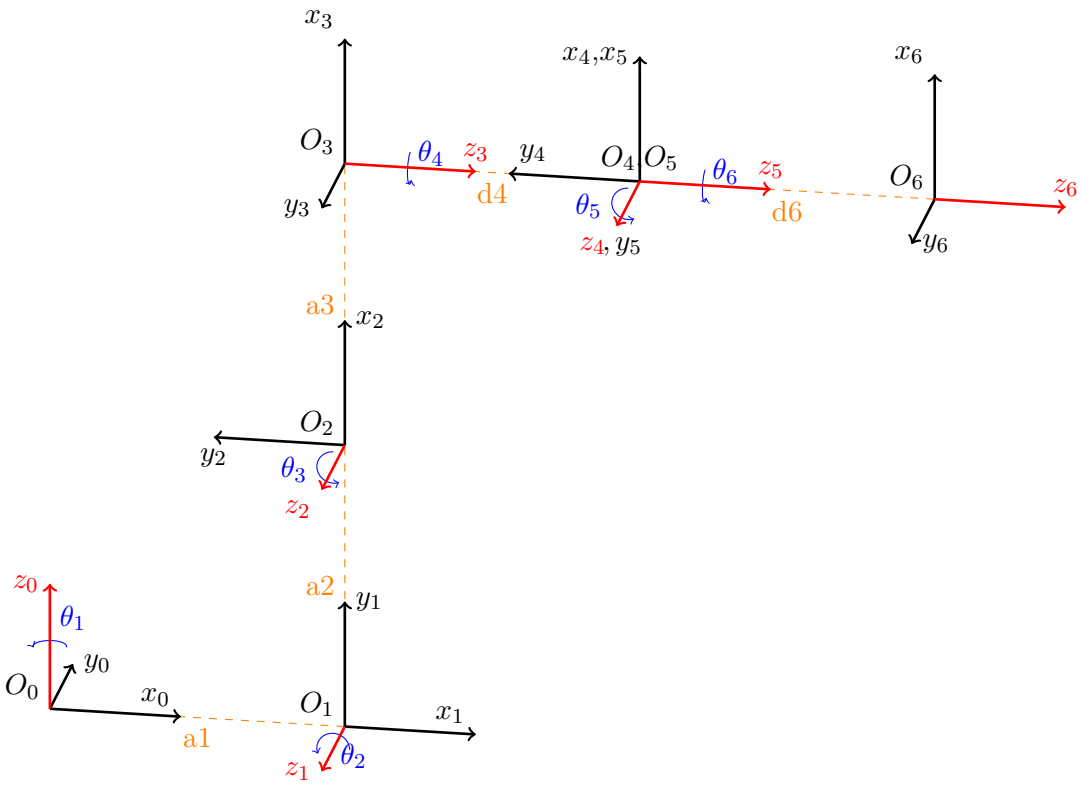


Figura 7.3: Sistemas de coordenadas definidos para o manipulador robótico na abordagem 1.

${}^{base}T_{end\_effector}$ pela sucessiva pós-multiplicação das transformações elementares (equação 7.3).

$$ {}^{i-1}T_i(\theta_i, d_i, a_i, \alpha_i) = Rot_z(\theta_i) \times Trans_z(d_i) \times Trans_x(a_i) \times Rot_x(\alpha_i) \tag{7.2} $$

$$ {}^{0}T_n = \prod_{i=1}^{n} {}^{i-1}T_i \tag{7.3} $$

Sabendo que a matriz de transformação ${}^{base}T_{end\_effector}$ é dada por:

$$ {}^{0}T_6 = \left[\begin{array}{ccc|c} & ROT & & \begin{matrix}X_m\\Y_m\\Z_m\end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] $$

e a matriz ROT por:

$$ ROT = \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix} $$

os ângulos $Roll(A_m)$, $Pitch(B_m)$ e $Yaw(Z_m)$ são calculados através da matriz $ROT_{[3\times3]}$ com o conjunto de equações 7.4.

$$\begin{cases} A_m = \arctan \frac{n_y}{n_x} \\ B_m = \arctan \frac{-n_z}{n_x \times \cos A_m + n_y \times \sin A_m} \\ C_m = \arctan \frac{s_z}{a_z} \end{cases} \tag{7.4}$$

As equações anteriores (equações 7.4) surgem da comparação termo a termo das matrizes resultantes da seguinte equação:

$$Rot_z^{-1}(A_m) \times Trans^{-1}(X_m, Y_m, Z_m) \times {}^0T_6 = Rot_y(B_m) \times Rot_x(C_m) \tag{7.5}$$

Abordagem 2

Para uma segunda abordagem, aproximou-se a plataforma móvel a uma cadeia cinemática de 2 juntas, em que a primeira é do tipo rotacional e a segunda do tipo prismática. Desta forma, o sistema pode ser estudado como um manipulador de 8 juntas, em que a posição da plataforma (X_p,Y_p,θ_p) pode ser representada pelo ângulo da primeira junta (θ_p) e pelo deslocamento de junta 2 (d_p).

Aplicando a metodologia *D-H* foram definidas e enumeradas as origens dos 8 elos (figura 7.4), e concebeu-se a tabela *D-H* 7.2.

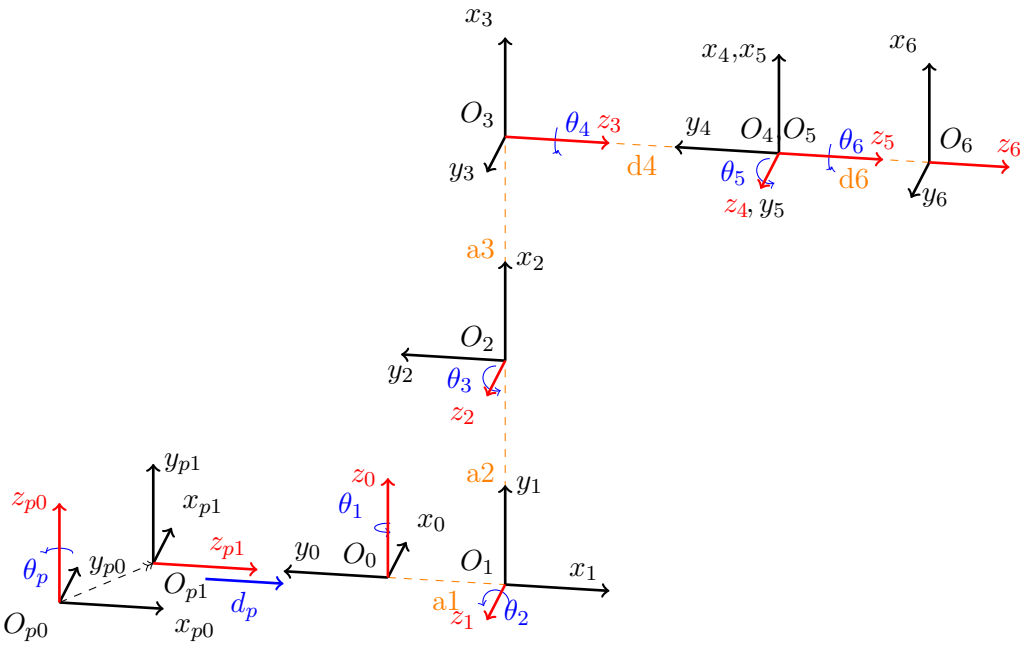


Figura 7.4: Sistemas de coordenadas definidos para o sistema total na abordagem 2.

Fazendo uso das equações descritas para a abordagem 1 (equações 7.2, 7.3 e 7.4) , foi possível o cálculo da posição da garra do manipulador (X_m, Y_m, Z_m) e orientação (A_m,B_m,C_m), em função das 8 juntas consideradas.

Tabela 7.2: Tabela dos parâmetros D-H usados na abordagem 2.

Elo	θ_i	a_i	d_i	α_i
1	$\theta_p + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
2	0	0	d_p	$-\frac{\pi}{2}$
3	$\theta_1 - \frac{\pi}{2}$	a_1	0	$\frac{\pi}{2}$
4	$\theta_2 + \frac{\pi}{2}$	a_2	0	0
5	θ_3	a_3	0	$\frac{\pi}{2}$
6	θ_4	0	d_4	$-\frac{\pi}{2}$
7	θ_5	0	0	$\frac{\pi}{2}$
8	θ_6	0	d_6	0

Como o referencial global é coincidente com o referencial local da base do manipulador, a posição do *end effector* no referencial global, $\vec{P_{ee}} = [X_{ee}, Y_{ee}, Z_{ee}, A_{ee}, B_{ee}, C_{ee}]^T$ é dado pelas equações 7.6.

$$\begin{cases} X_{ee} = X_m \\ Y_{ee} = Y_m \\ Z_{ee} = h_p + Z_m \\ A_{ee} = A_m \\ B_{ee} = B_m \\ C_{ee} = C_m \end{cases} \tag{7.6}$$

7.2.2 Cinemática diferencial

Tal como já foi exposto anteriormente, para a cinemática direta existe uma relação entre o espaço de juntas e o espaço cartesiano que pode ser definida como $\vec{r} = \vec{F}(\vec{q})$. Desta forma, introduzindo o conceito de movimento diferencial, para que a posição da garra do manipulador descreva um dado incremento de deslocamento no espaço, é necessário conhecer os incrementos nas diversas juntas de um manipulador ou vice-versa. Esta relação entre os dois espaços é dada pelo Jacobiano (J) da função vetorial $\vec{F}$ do manipulador.

Assim, a relação entre as variações de posição com o tempo, ou seja as velocidades, podem também ser definida como:

$$\frac{d\vec{r}}{dt} = J \times \frac{d\vec{q}}{dt}$$

Consequentemente, o inverso:

$$\frac{d\vec{q}}{dt} = J_I \times \frac{d\vec{r}}{dt} \tag{7.7}$$

também é verdadeiro, sendo que J_I representa o Jacobiano inverso.

Diferenciando as expressão de $\vec{P_{ee}}$ em ordem a cada variável a controlar, resultaram dois Jacobianos (J_1 para abordagem 1 e J_2 para a abordagem 2):

$$J_1 = \begin{bmatrix} \frac{\partial X_{ee}}{\partial X_p} & \frac{\partial X_{ee}}{\partial Y_p} & \frac{\partial X_{ee}}{\partial \theta_p} & \cdots & \frac{\partial X_{ee}}{\partial \theta_6} \\ \frac{\partial Y_{ee}}{\partial X_p} & \frac{\partial Y_{ee}}{\partial Y_p} & \frac{\partial Y_{ee}}{\partial \theta_p} & \cdots & \frac{\partial Y_{ee}}{\partial \theta_6} \\ \vdots & \vdots & \ddots & \vdots & \\ \frac{\partial C_{ee}}{\partial X_p} & \frac{\partial C_{ee}}{\partial Y_p} & \frac{\partial C_{ee}}{\partial \theta_p} & \cdots & \frac{\partial C_{ee}}{\partial \theta_6} \end{bmatrix} J_2 = \begin{bmatrix} \frac{\partial X_{ee}}{\partial \theta_p} & \frac{\partial X_{ee}}{\partial d_p} & \cdots & \frac{\partial X_{ee}}{\partial \theta_6} \\ \frac{\partial Y_{ee}}{\partial \theta_p} & \frac{\partial Y_{ee}}{\partial d_p} & \cdots & \frac{\partial Y_{ee}}{\partial \theta_6} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C_{ee}}{\partial \theta_p} & \frac{\partial C_{ee}}{\partial d_p} & \cdots & \frac{\partial C_{ee}}{\partial \theta_6} \end{bmatrix}$$

7.2.3 Manipulabilidade

De acordo com [89], a manipulabilidade de um manipulador robótico (ω_a) pode ser quantificada pela equação 7.8, em que q_a representa a configuração de juntas para um determinado ponto no espaço e J_a o Jacobiano numérico para essa mesma configuração.

$$\omega_a = \sqrt{\det\left(J_a(q_a) \times J_a^T(q_a)\right)} \tag{7.8}$$

Tendo em consideração os ângulos de orientação das rodas da plataforma (castores e rodas diferenciais), surgem outras abordagens mais complexas para definição da manipulabilidade de manipuladores móveis, como pode ser visto em [89] e [92]. No entanto, como nesta dissertação não é possível obter esses valores, faz-se uso da expressão tradicional para o calculo/análise da manipulabilidade.

7.2.4 Implementação prática

Para estudar a viabilidade dos modelos cinemáticos propostos em cada abordagem simulou-se alguns movimentos utilizando o *matlab*.

Para o efeito, foram utilizadas as medidas presentes no modelo URDF do robô *FANUC LR Mate 200iD* para os comprimentos dos elos (as mesmas utilizadas no capítulo anterior). A tabela 7.3, apresenta as medidas e a sua respetiva associação às variáveis utilizadas.

Tabela 7.3: Tabela referente aos valores usados para comprimentos dos elos.

Variável	**Valor [m]**
a_1	0.050
a_2	0.330
a_3	0.035
d_4	0.335
d_6	0.080

Definindo a posição *hardware* do sistema como todas as variáveis a 0, a representação do sistema pode ser visualizada na imagem 7.5 e na imagem 7.6, para a abordagem 1 e 2 respetivamente. Para uma melhor perceção, as cores associadas a cada referencial seguiram a seguinte lógica: branco, vermelho, verde, azul, amarelo, magenta, vermelho, verde, azul.

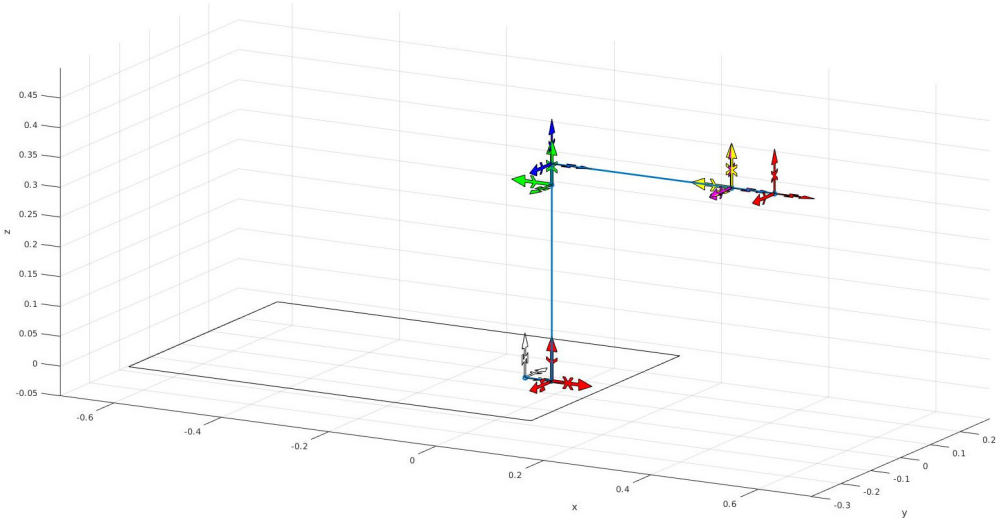


Figura 7.5: Representação do sistema na posição *hardware* para a abordagem 1.

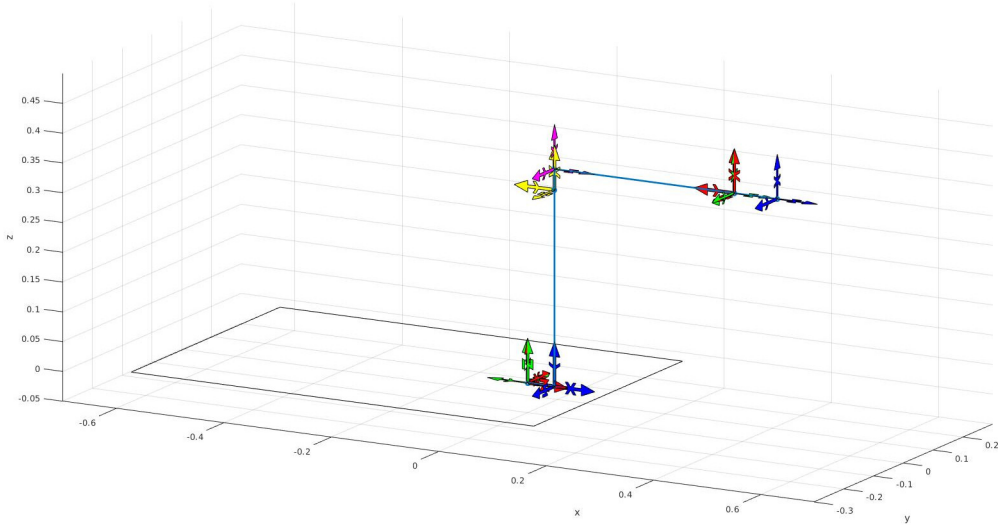


Figura 7.6: Representação do sistema na posição *hardware* para a abordagem 2.

Para a simulação, a metodologia usada foi semelhante nas duas abordagens:

- Definição da trajetória a executar dividida em curtos intervalos ($d\vec{r}$);
- Cálculo dos valores incrementais das juntas correspondente a cada incremento no espaço cartesiano, com base na equação 7.7.
- Cálculo da posição do *end effector* e das origens de cada elo para cada instante, através da cinemática direta.
- Representação dos pontos de passagem do *end effector*, do referencial base da plataforma móvel, e da posição do robô ao longo da trajetória.
- Cálculo da manipulabilidade para cada posição do robô ao longo do caminho.

Este tipo de estratégia (cinemática diferencial), para o planeamento de trajetórias, apresenta erros cumulativos. Assim, para minimizar esses erros, optou-se por usar um número elevado de pontos de passagem para as trajetórias testadas (400 pontos).

É de salientar que, caso as últimas cinco juntas (de ambas as abordagens) se encontrem definidas como 0, estamos na presença de singularidade, ou seja, o Jacobiano inverso é não definido. Deste modo, as trajetórias a executar não podem iniciar ou passar por uma configuração do robô com os valores anteriormente referidos.

Trajetória 1

Para o ponto inicial da trajetória, foi definida a posição de juntas como $[0, 0, 0, 0, 0, 0, \frac{\pi}{4}, 0, 0]$ para abordagem 1, e $[0, 0, 0, 0, 0, 0, \frac{\pi}{4}, 0, 0]$ para a abordagem 2. Esta configuração de juntas corresponde a uma posição inicial do *end effector*:

$$Pi_{ee} = \begin{bmatrix} 0.4650 & 0 & 0.3650 & -1.5708 & -0.7854 & -1.5708 \end{bmatrix}$$

Assim, a trajetória 1 consistiu na movimentação linear ($\delta x = 2$, $\delta y = 2$, $\delta z = 0.3$) do manipulador desde o ponto inicial (Pi_{ee}) até o ponto final (Pf_{ee}), definido como:

$$Pf_{ee} = \begin{bmatrix} 2.4650 & 2 & 0.6650 & -1.5708 & -0.7854 & -1.5708 \end{bmatrix}$$

Após a simulação, obtiveram-se as trajetórias do manipulador móvel, representadas nas figuras 7.7 e 7.8, para a abordagem 1 e 2 respetivamente. São representadas as posições do *end effector* a vermelho e as posições da base da plataforma a azul.

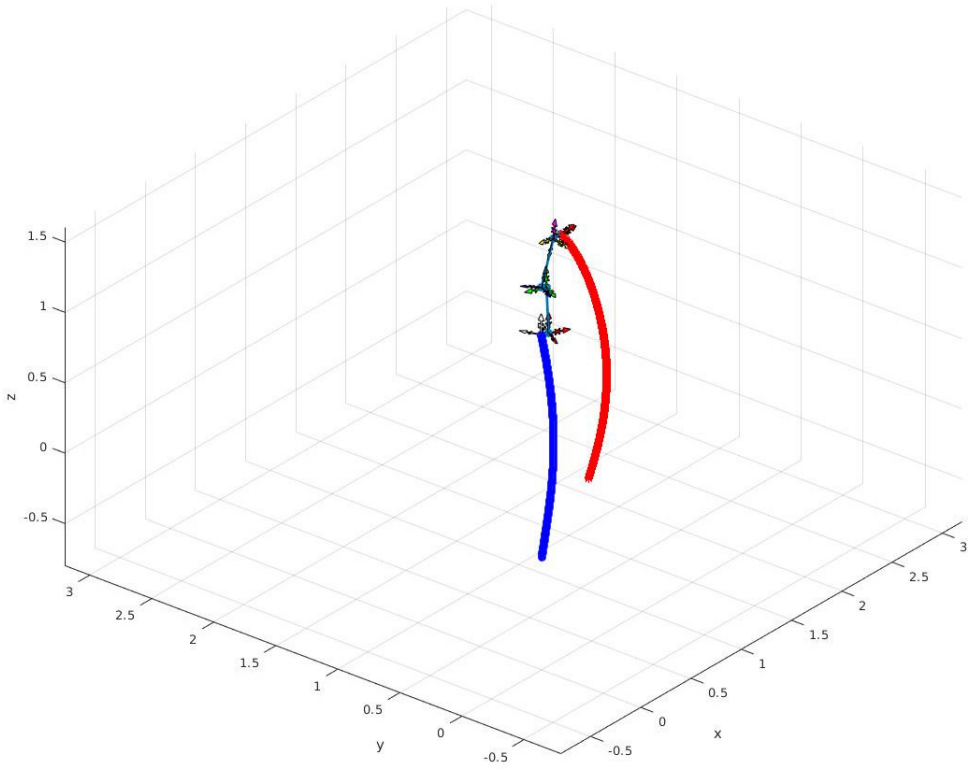


Figura 7.7: Representação da trajetória 1 definida pela abordagem 1.

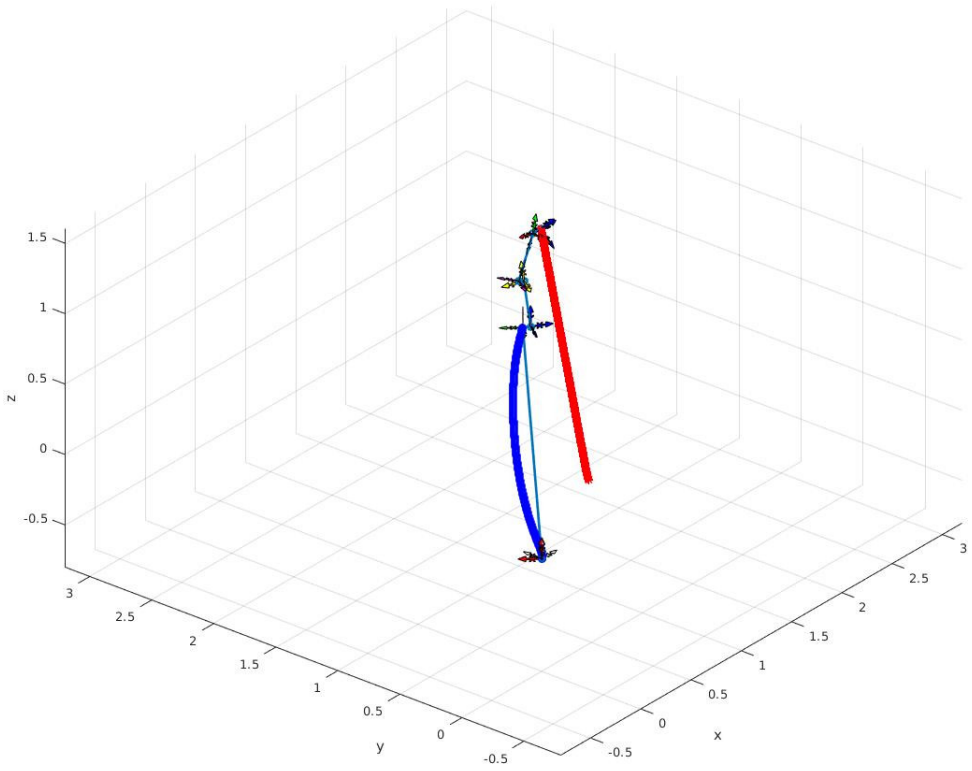


Figura 7.8: Representação da trajetória 1 definida pela abordagem 2.

É de salientar que a configuração final do manipulador móvel se mantém com as abordagens utilizadas. As figuras 7.9 e 7.10 representam as configurações finais para cada abordagem.

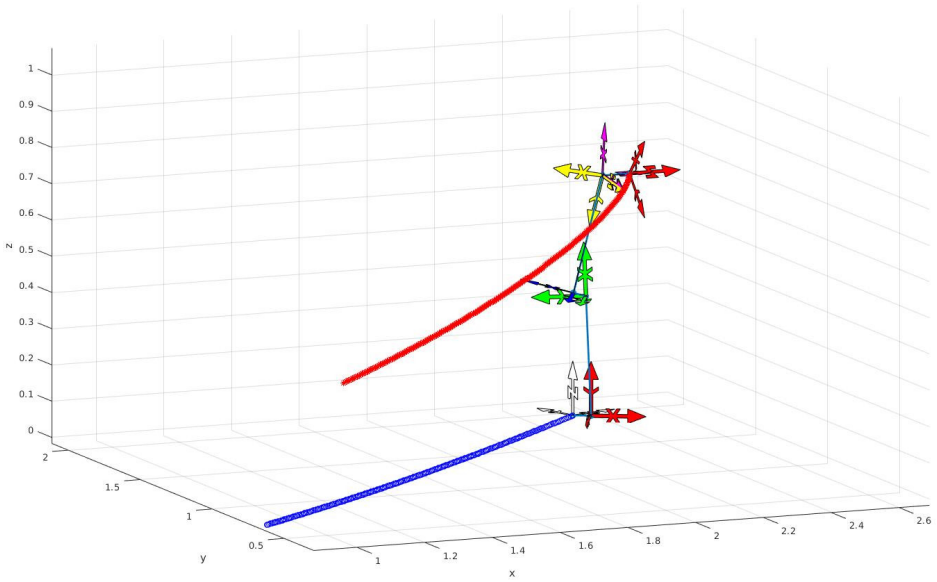


Figura 7.9: Representação da posição final do sistema, para a trajetória 1 definida pela abordagem 1.

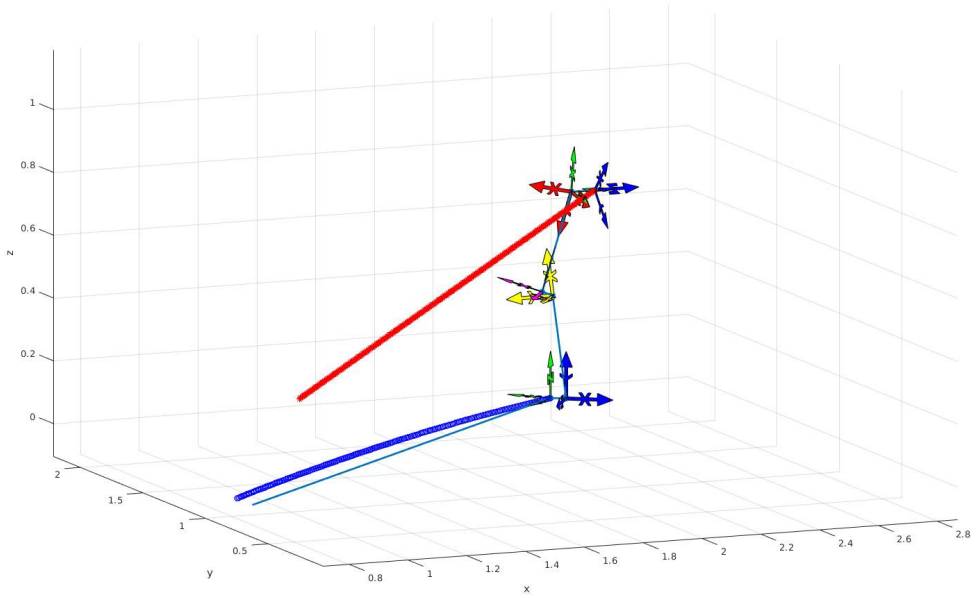
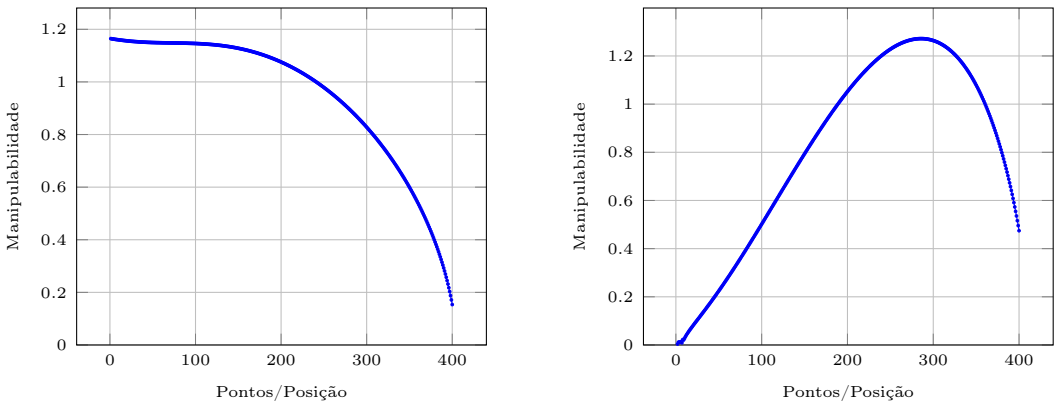


Figura 7.10: Representação da posição final do sistema, para a trajetória 1 definida pela abordagem 2.

Ao longo da trajetória é possível visualizar a manipulabilidade do sistema total, considerando a equação 7.8. Observa-se a evolução da manipulabilidade ao longo da trajetória na imagem 7.11



(a) Manipulabilidade do sistema total ao longo da trajetória 1 para a abordagem 1.

(b) Manipulabilidade do sistema total ao longo da trajetória 1 para a abordagem 2.

Figura 7.11: Representação dos valores de manipulabilidade do sistema total ao longo da trajetória 1, para as diferentes abordagens.

Trajetória 2

Para a segunda trajetória parametrizou-se uma curva no espaço, de modo a simular uma tarefa de pintura numa parede de 5 m . Esta parametrização foi definida com base na equação 7.9.

$$\begin{cases} 0 \leq t \leq 5 \\ x(t) = x_0; \\ y(t) = y_0 + t; \\ z(t) = z_0 + 0.08 \times \cos(16t) \end{cases} \tag{7.9}$$

Derivando as expressões em ordem a t e assumindo que não há variação da orientação do *end effector* é possível calcular os incrementos de cada componente ($d\vec{r}$) em cada iteração.

O *end effector* iniciou a trajetória a partir do ponto inicial definido como:

$$Pi_{ee} = \begin{bmatrix} 0.4650 & 0 & 0.3650 & -1.5708 & -0.7854 & -1.5708 \end{bmatrix}$$

Após a simulação foram obtidas as trajetórias do manipulador móvel, tal como se pode observar nas figuras 7.12 e 7.13, para a abordagem 1 e 2 respetivamente.

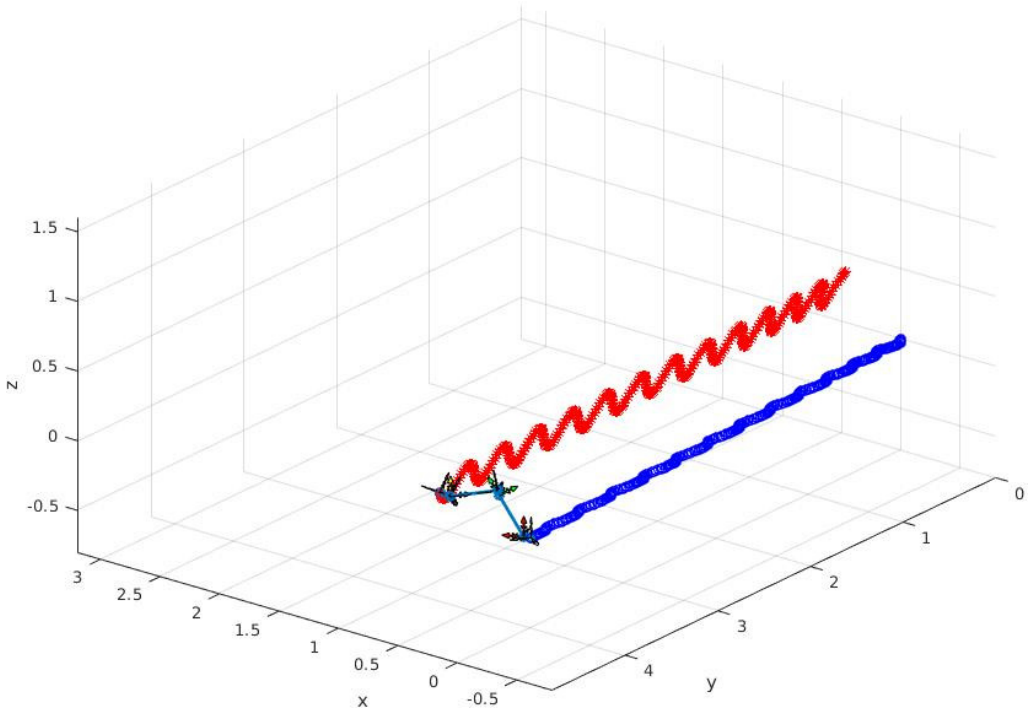


Figura 7.12: Representação da trajetória 2 definida pela abordagem 1.

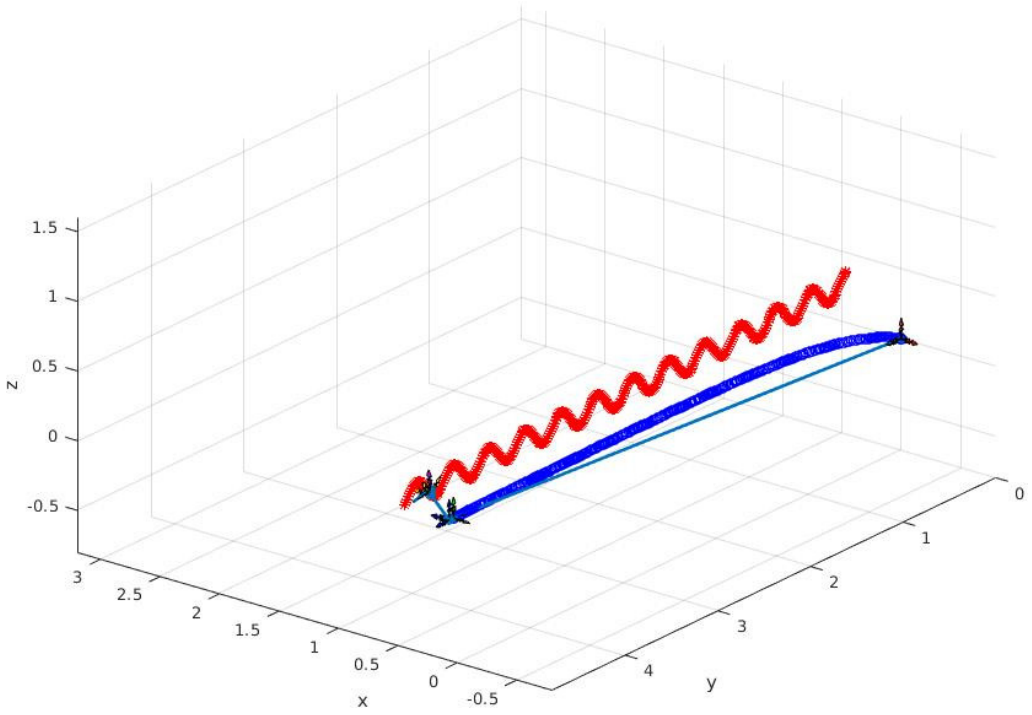


Figura 7.13: Representação da trajetória 2 definida pela abordagem 2.

Contrariamente à trajetória 1, é de salientar que configuração final do manipulador móvel não se mantêm com as abordagens utilizadas. As figuras 7.14 e 7.15 representam as configurações finais para cada abordagem.

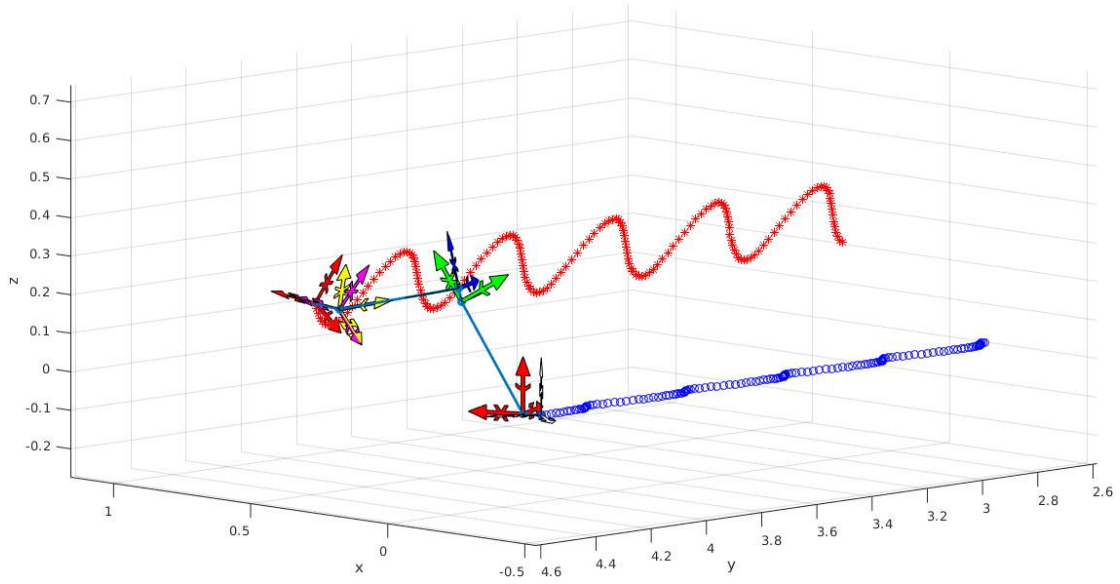


Figura 7.14: Representação da posição final do sistema, para a trajetória 2 definida pela abordagem 1.

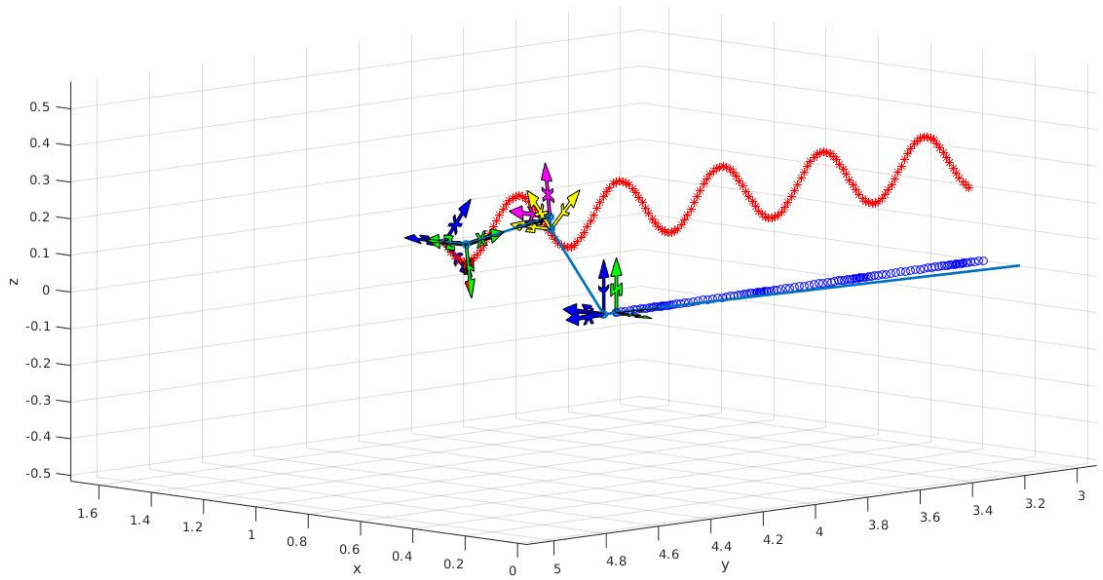
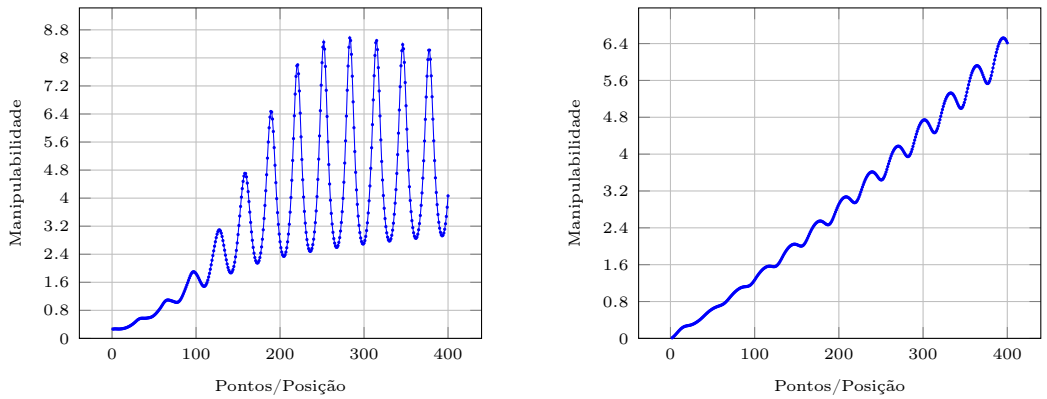


Figura 7.15: Representação da posição final do sistema, para a trajetória 2 definida pela abordagem 2.

Ao longo da trajetória é calculada a manipulabilidade do sistema total, considerando a equação 7.8. Na figura 7.16 está representada a evolução da manipulabilidade ao longo da trajetória 2.



(a) Manipulabilidade do sistema total ao longo da trajetória 2 para a abordagem 1.

(b) Manipulabilidade do sistema total ao longo da trajetória 2 para a abordagem 2.

Figura 7.16: Representação dos valores de manipulabilidade do sistema total ao longo da trajetória 2, para as diferentes abordagens.

Análise

Analisando as trajetórias propostas, conclui-se que a propagação de erros devido à cinemática diferencial é mais notória nas trajetórias impostas pela abordagem 1, uma vez que o caminho do *end effector* apresenta uma ligeira inclinação em ambas as trajetórias

consideradas. Tal efeito pode ser minimizado pelo o aumento do número de pontos, ou seja, pela diminuição dos incrementos usados.

Após a análise dos gráficos de manipulabilidade obtidos, é de salientar que a generalização da manipulabilidade de braços robóticos tradicionais, para manipuladores móveis, não é aplicável. Ulteriormente à análise dos Jacobianos, é possível concluir que este efeito é originado pelo aumento da distância à origem, uma vez que as juntas associadas à cinemática da plataforma móvel não apresentam limites.

Deste modo, optou-se por estudar apenas a manipulabilidade do braço robótico ao longo das trajetórias nas diferentes abordagens. Contudo, a obtenção do Jacobiano utilizada anteriormente (obtida pela diferenciação das expressões trigonométricas inversas) apresentava problemas de definição nas suas componentes angulares, o que implicou a sua reformulação para este caso. Com base em [93], definiu-se o Jacobiano do braço robótico (J) em duas matrizes (equação 7.10), J_v e J_w, que representam a contribuição da velocidade de rotação das juntas para a velocidade do *end effector*, respetivamente.

$$J = \begin{bmatrix} J_v \\ J_w \end{bmatrix} \tag{7.10}$$

Enquanto que J_v é calculado normalmente através da derivação das coordenadas x, y, z em ordem às variáveis de juntas, a matriz J_w é calculada através de:

$$J_w = \begin{bmatrix} {}^0z_0 & {}^0z_1 & {}^0z_2 & {}^0z_3 & {}^0z_4 & {}^0z_5 \end{bmatrix} \tag{7.11}$$

onde

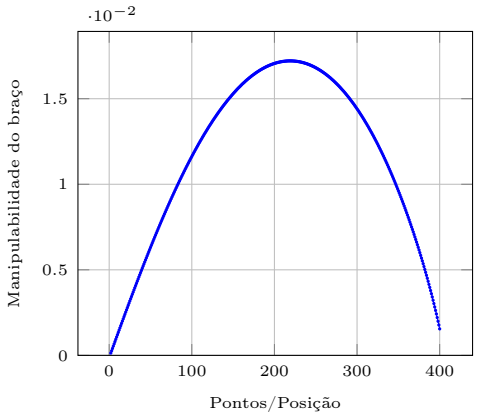
$$\begin{aligned} {}^0z_0 &= k \\ {}^0z_1 &= {}^0R_1 k \\ {}^0z_2 &= {}^0R_2 k = {}^0R_1 {}^1R_2 k \\ {}^0z_3 &= {}^0R_3 k = {}^0R_1 {}^1R_2 {}^2R_3 k \\ {}^0z_4 &= {}^0R_4 k = {}^0R_1 {}^1R_2 {}^2R_3 {}^3R_4 k \\ {}^0z_5 &= {}^0R_5 k = {}^0R_1 {}^1R_2 {}^2R_3 {}^3R_4 {}^4R_5 k \end{aligned} \tag{7.12}$$

com $k = [0, 0, 1]^T$ e ${}^0R_i$ é dado pela componente rotacional de cada matriz de transformação entre elos.

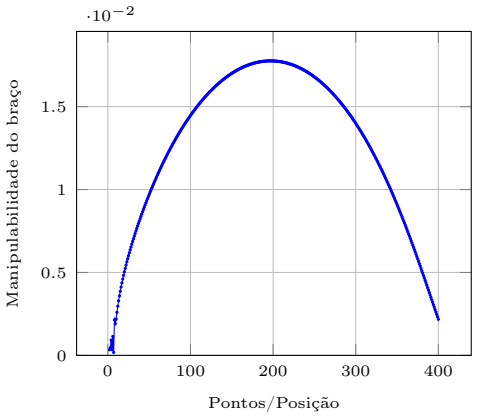
Assim, foram obtidas as evoluções da manipulabilidade do braço robótico ao longo das trajetórias 1 e 2 para as duas abordagens (figuras 7.17 e 7.18 , respetivamente).

Nas figuras 7.17a e 7.17b é notória a diminuição da manipulabilidade a partir de um certo ponto médio. Esta diminuição pode estar associada à aproximação do limite de altura (coordenada em z) que o manipulador pode atingir, ou seja, a cota do ponto final da trajetória é próxima da cota máxima que o manipulador pode alcançar.

Analisado as figuras 7.18a e 7.18b, existe um aumento da manipulabilidade em ambas as abordagens na fase inicial da trajetória 2. Este efeito é mais notório na abordagem 2 (figura 7.18b) e pode ser explicado pela aproximação da plataforma ao plano de trabalho (figura 7.13), isto é, onde a curva está parametrizada.

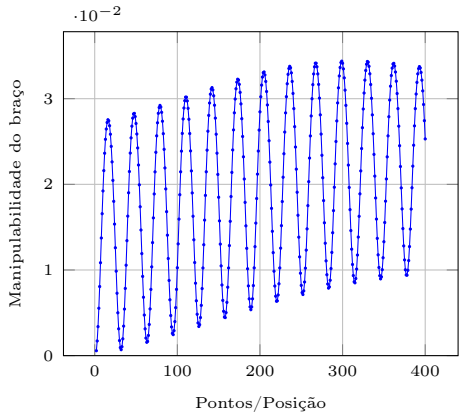


(a) Manipulabilidade do braço robótico ao longo da trajetória 1 para a abordagem 1.

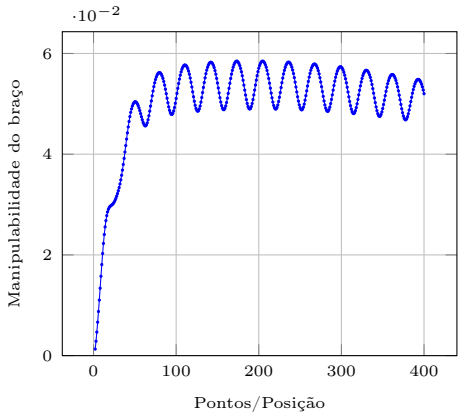


(b) Manipulabilidade do braço robótico ao longo da trajetória 1 para a abordagem 2.

Figura 7.17: Representação dos valores de manipulabilidade do braço robótico ao longo da trajetória 1, para as diferentes abordagens.



(a) Manipulabilidade do braço robótico ao longo da trajetória 2 para a abordagem 1.



(b) Manipulabilidade do braço robótico ao longo da trajetória 2 para a abordagem 2.

Figura 7.18: Representação dos valores de manipulabilidade do braço robótico ao longo da trajetória 2, para as diferentes abordagens.

7.3 Aplicabilidade da cinemática integrada no ROBONUC

Para averiguar a possibilidade de aplicação da cinemática integrada em ambiente `ROS` no sistema real ROBONUC, foi efetuado um estudo do funcionamento das ferramentas atualmente usadas no projeto, bem como eventuais ferramentas úteis para aplicação futura.

Como já foi referido anteriormente, o `moveit` é uma ferramenta que permite o controlo, o planeamento de movimentos, evita colisões e possibilita a incorporação de extensões para resolução da cinemática inversa.

Atualmente, neste projeto, para a resolução da cinemática inversa do manipulador é usado um *plug-in* standard do `moveit`, o `KDLKinematics`. Contudo, este plug-in, tal

como o alternativo (IKfast), não geram resultados para robôs com mais de 7 graus de liberdade [94].

De acordo com [95], o método usado pelo KDLKinematics apresenta mais alguns problemas, dos quais se destacam as falhas frequentes de convergência em juntas limitadas e a falha de ação quando o algoritmo fica "preso" num mínimo local.

Assim, como alternativa surge a biblioteca TRAC-IK que faz uso de dois métodos em simultâneo, um método usando o Jacobiano pseudo-inverso e um método de cinemática inversa de otimização não-linear. O primeiro (KDL-RR) é um melhoramento do método de Newton usado no KDL, que deteta e diminui os mínimos locais, enquanto que o segundo (SQP-SS) é uma otimização PQS (programação quadrática sequêncial) que utiliza métodos quasi-Newton de forma a lidar melhor com os limites das juntas. Ambos os métodos são lançados em simultâneo e quando algum converge para uma solução, o algoritmo é interrompido e a solução é retornada [95] [96].

Esta biblioteca pode ser incorporada no moveit através do seu *plug-in* ou pode ser usada individualmente. Para uso individual apenas é necessário fornecer o URDF do sistema e a configuração de juntas do ponto inicial. Para a sua aplicação sugere-se os exemplos presentes em [97].

De modo a perceber o controlo realizado pelo moveit no robô real, estudou-se a arquitetura (do moveit) existente (figura 4.12). A comunicação entre o ROS e um robô é realizada através de uma ação predefinida designada por FollowJointTrajectory action. No entanto, para o manipulador usado (onde é necessário o uso do meta-pacote ROS industrial), o pacote industrial_robot_client fornece uma ação para esse fim, designada de joint_trajectory_action. Assim, o nó principal do moveit (move_group) faz uso dessa ação para comunicar com o respetivo nó servidor de ações joint_trajectory_action. De forma a atuar o manipulador, os nós executados no controlador FANUC subscrevem o tópico /joint_path_command, publicado pelo servidor (joint_trajectory_action). Neste tópico é publicada uma estrutura de mensagens do tipo JointTrajectoryPoint.msg, na qual são enviadas informações de posição, velocidades, acelerações, esforços e tempo. Em suma, num sentido unidirecional, a interface (em C++ ou Python) comunica com o nó principal do moveit (move_group), que consequentemente usa a ação joint_trajectory_action para enviar a trajetória ou ponto de destino para o controlador, através do tópico /joint_path_command. A figura 7.19 representa o processo descrito.

Assim, atualmente não existe nenhuma ferramenta direcionada para planeamento e controlo de movimentos integrados de um manipulador móvel. Tal como foi concebido na abordagem 2, para uso de planeadores já existentes como o moveit, a solução pode consistir na aproximação dos graus de mobilidade da plataforma em ângulos de junta de um manipulador e posteriormente converter as soluções para cada sistema. Como os atuadores de cada subsistema têm origens diferentes, o seu controlo simultâneo necessita de uma arquitetura de controlo global.

Alternativamente aos planeadores existentes, podem ser concebidos algoritmos de raiz para o sistema integrado. Algumas bibliotecas, como EXtensible Optimization Toolset framework (EXOTica) [98] ou Flexible Collision Library (FCL) [99] podem ser utilizadas. Enquanto que a primeira fornece algoritmos genéricos para otimização direcionadas de sistemas robóticos, a segunda fornece ferramentas para evitar colisões.

Em suma, para uma futura continuidade do trabalho, conclui-se que após o desenvolvimento de uma arquitetura de controlo simultâneo dos dois sistemas, existe a possibili-

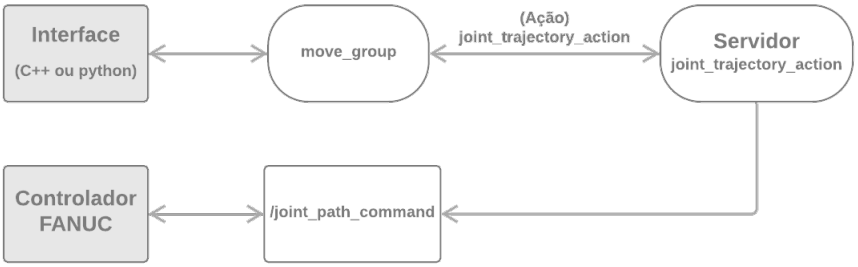


Figura 7.19: Representação da arquitetura atual para controlo do manipulador FANUC por parte do `moveit`.

dade da implementação de uma cinemática integrada para o sistema real, considerando as simplificações e ferramentas enumeradas.

Uma solução possível e interessante de ser implementada, seria a simplificação utilizada na abordagem 2 utilizando a biblioteca `TRAC-IK`, como extensão do `moveit`, para planeamento de movimento. As velocidades das duas primeiras juntas (rotacional e prismática), calculadas pela biblioteca, teriam que ser convertidas para as velocidades individuais de cada roda. Esta conversão pode ser realizada com base na equação 7.13 (deduzidas em [20]), possibilitando assim o controlo de cada uma das rodas, na qual v_r e v_l representam a velocidade da roda direita e esquerda, respetivamente. No entanto, o sistema poderá ficar comprometido devido às limitações do hardware existente, como por exemplo, a presença das rodas castor. Neste caso seria necessário adotar estratégias de forma a monitorizar e controlar a sua orientação.

$$\begin{cases} v_r = \frac{2V_{robot}+\omega_{robot}L}{2R} \\ v_l = \frac{2V_{robot}-\omega_{robot}L}{2R} \end{cases} \tag{7.13}$$

Capítulo 8

Testes e Resultados

De forma a validar e demonstrar a solução desenvolvida foram realizados testes em dois ambientes (ambiente 1 e ambiente 2), sujeitos a diferentes condições. Além destes, de forma a avaliar o seu desempenho, foram realizados testes ao controlo remoto, foi analisada a reação dos estados propostos em diferentes velocidades e analisado o comportamento do sistema para trajetórias retilíneas. Neste capítulo são apresentados os testes efetuados, os resultados obtidos e a sua respetiva discussão.

Por último, foi realizada uma demonstração de um trajeto sinusoidal de forma a demonstrar o potencial de aplicação da cinemática integrada.

8.1 Controlo Remoto

Após a realização de alguns testes, conclui-se que a implementação do controlo remoto, com recurso ao ROS distribuído, foi realizada com sucesso. No entanto, a taxa de informação necessária à interface gráfica para a tele-operação é superior à capacidade do *router* existente na plataforma, existindo assim, um tempo de atraso muito elevado entre o que a interface fornece e o que realmente acontece.

Como estes problemas podem comprometer a segurança do sistema, os testes descritos nas próximas secções foram realizados com o computador de bordo e um monitor em cima da plataforma, de modo a tele-operar o manipulador móvel em "tempo real".

Para executar todos os nós necessários e inicializar a interface no computador de bordo deve-se executar, no terminal, a seguinte linha:

```
roslaunch robonuc_integration robonuc_integration.launch
rviz:=true
```

Caso o argumento `rviz` não seja inserido, este será lançado como falso e a interface gráfica não é lançada.

8.2 Ambiente 1

Para a realização de alguns testes, foi escolhido o laboratório de robótica como o "ambiente 1", onde, a plataforma foi tele-operada até a uma zona de aproximação e posteriormente foram iniciados os processos de aproximação, orientação e de *bin-picking*.

Uma vez que este ambiente (ambiente 1) apresenta condições desfavoráveis ao bom funcionamento do modo automático de navegação, não foi possível a sua utilização. Este

ambiente gera facilmente problemas de localização e consequentemente problemas de navegação autónoma.

Foram realizados 11 testes no "ambiente 1". A tabela 8.1 traduz os resultados obtidos, onde cada "✓" representa a execução positiva de cada estado, e cada "✗" representa a falha de execução do estado.

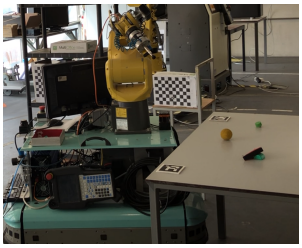
Teste	Estado 1	Estado 2	Estado 3	Estado 4
1	✓	✓	✓	✓
2	✓	✓	✓	✓
3	✓	✓	✓	✗
4	✓	✓	✓	✗
5	✓	✓	✓	✗
6	✓	✓	✓	✓
7	✓	✗	✗	✗
8	✓	✓	✓	✗
9	✓	✓	✓	✓
10	✓	✓	✓	✓
11	✓	✓	✓	✓

Tabela 8.1: Tabela Representativa do sucesso de cada estado em cada experiência efetuada no ambiente 1.

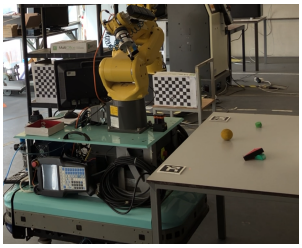
Esta tabela mostra que num total de 11 testes, 4 falharam o *bin-picking* e 1 falhou o estado de aproximação (estado 2).

Nos testes efetuados, 6 foram executados com completo sucesso. A figura 8.1 representa um exemplo da sequência de estados de uma destas tarefas, na qual pode ser visualizada a seguinte sequência: início da aproximação à bancada (fig. 8.1a); início do processo de orientação (fig. 8.1b); finalização do estado de orientação e início do *bin-picking* (fig. 8.1c); *picking* do objeto (fig. 8.1d).

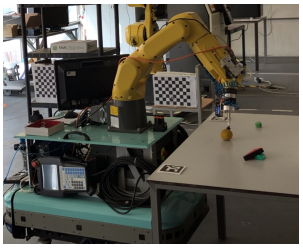
(a)



(b)



(c)



(d)

Figura 8.1: Representação ilustrativa da sequência de estados de uma tarefa completa.

Nos testes 3, 4 e 5 o manipulador não conseguiu executar a aproximação linear para o *picking* do objeto alvo. Este incumprimento do estado não foi causado pela falha do estado 4, mas sim, devido a limitações de alcance do braço para a trajetória de aproximação imposta. A figura 8.2 representa um dos testes cujo *bin-picking* falhou por falta de alcance.

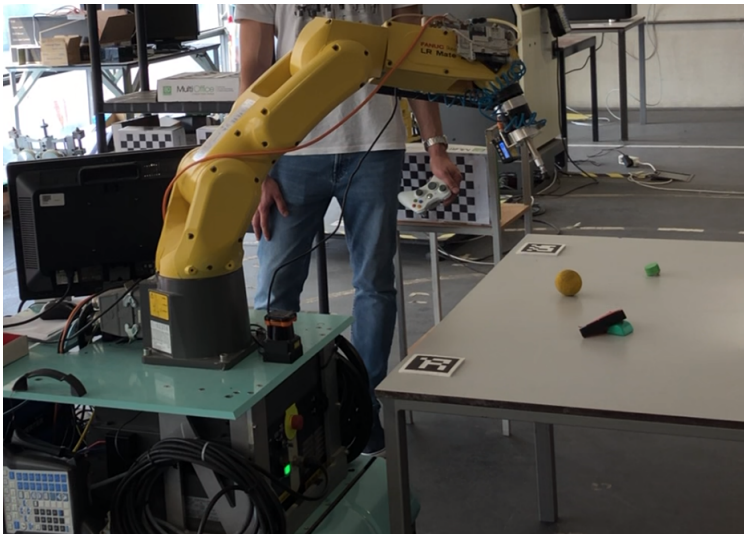


Figura 8.2: Representação ilustrativa da falta de alcance num determinado teste, do ambiente 1.

No teste 7, o efeito das rodas castor na trajetória de aproximação causou a colisão da plataforma com a bancada de trabalho. Tal aconteceu devido ao excesso de orientação da plataforma face à bancada de trabalho, ou seja, o laser não detetou nenhum obstáculo. Assim, o utilizador foi obrigado a cancelar a ação após a perceção da colisão.

A orientação final aquando a colisão pode ser observada na figura 8.3.

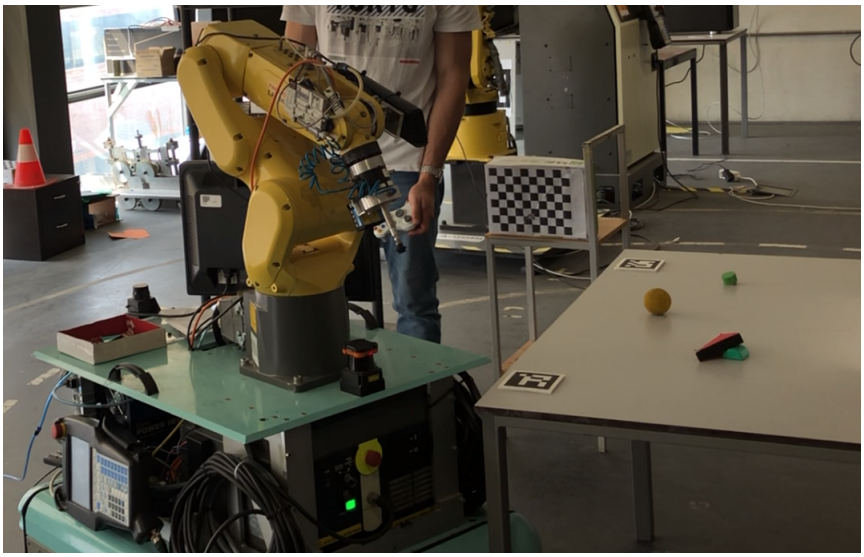


Figura 8.3: Representação da colisão no teste 7 no ambiente 1.

8.3 Ambiente 2

De modo a demonstrar e testar a integração da navegação autónoma no sistema, o segundo ambiente foi escolhido por apresentar uma geometria mais facilitada, permitindo uma redução dos erros de localização por parte do `HectorSLAM`. A figura 8.4 representa o ambiente anteriormente referido.

Figura 8.4: Representação do ambiente 2.

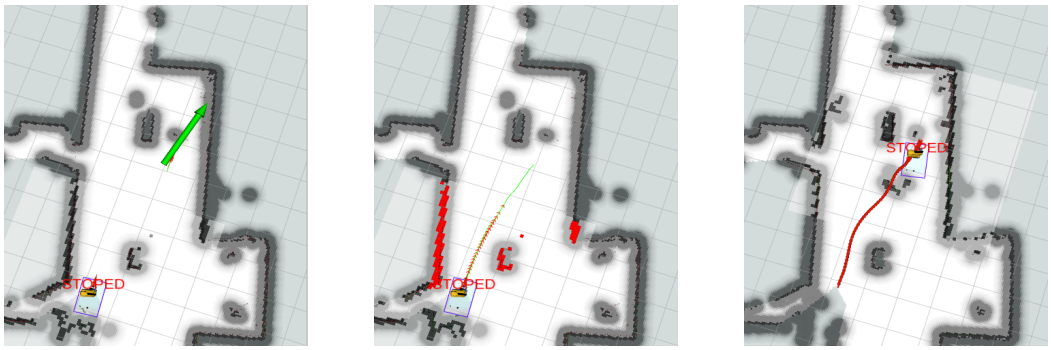
Tal como se pode observar na figura 8.5, para evitar os problemas de alcance percecionados no "ambiente 1", no "ambiente 2" os objetos foram posicionados mais próximos da extremidade de aproximação do manipulador móvel.

Figura 8.5: Representação da posição dos objetos na bancada presente no ambiente 2.

O procedimento efetuado é semelhante em todas as tentativas. A primeira etapa, caso não exista mapa reconstruido, consiste no reconhecimento do ambiente para a construção do mapa. A segunda consiste no posicionamento do manipulador móvel na posição de partida. Por fim, a terceira consiste na escolha do ponto e orientação final no mapa (figura 8.6a).

A figura 8.6b representa a trajetória a executar, calculada pelos nós responsáveis, e a figura 8.6c representa o trajeto do robô baseado na hodometria. É de salientar que esta hodometria é proveniente das leituras dos *enconders* das rodas com correções por parte do `HectorSLAM`.

Para este ambiente optou-se por realizar mais testes do que no "ambiente 1", obtendo assim um total de 26 testes. A tabela 8.2 representa os resultados obtidos.



(a) Escolha do ponto e orientação de destino.

(b) Trajetória a executar.

(c) Trajeto proveniente da hodometria.

Figura 8.6: Representação do modo de navegação automático no ambiente 2.

Teste	Estado 1	Estado 2	Estado 3	Estado 4
1	✓	✓	✓	✓
2	✗	✗	✗	✗
3	✓	✓	✓	✓
4	✓	✓	✓	✓
5	✗	✗	✗	✗
6	✓	✓	✓	✓
7	✓	✓	✓	✓
8	✗	✗	✗	✗
9	✗	✗	✗	✗
10	✓	✓	✓	✓
11	✓	✓	✓	✓
12	✓	✓	✓	✓*
13	✓	✓	✓	✓
14	✗	✗	✗	✗
15	✗	✗	✗	✗
16	✓	✓	✓	✓*
17	✓	✓	✓	✓
18	✓	✓	✓	✓*
19	✗	✗	✗	✗
20	✓	✓	✓	✓*
21	✗	✗	✗	✗
22	✓	✓	✓	✓
23	✗	✗	✗	✗
24	✓	✓	✓	✓
25	✓	✓	✓	✓
26	✗	✗	✗	✗

Tabela 8.2: Tabela que descreve o sucesso de cada estado em cada experiência efetuada no ambiente 2. (* sucesso do estado à segunda tentativa)

Em suma, num total de 26 testes, o modo autónomo falhou 10 vezes e a missão foi cumprida 16 vezes, sendo que, em 4 destas foram necessárias duas tentativas para o correto *picking* do objeto. Em uma destas 4, o *picking* falhou por falta de alcance, conseguindo reaproximar-se e atingir o objetivo à segunda tentativa. É de salientar, sempre que, no estado de navegação, o modo autónomo não alcançou o local desejado, todos os outros estados não puderam ser executados.

As falhas provenientes do modo automático devem-se a falhas de mapeamento e de localização. Na figura 8.7 estão representados alguns destes casos, onde é notória a falha na reconstrução do ambiente, e consequentemente da localização.

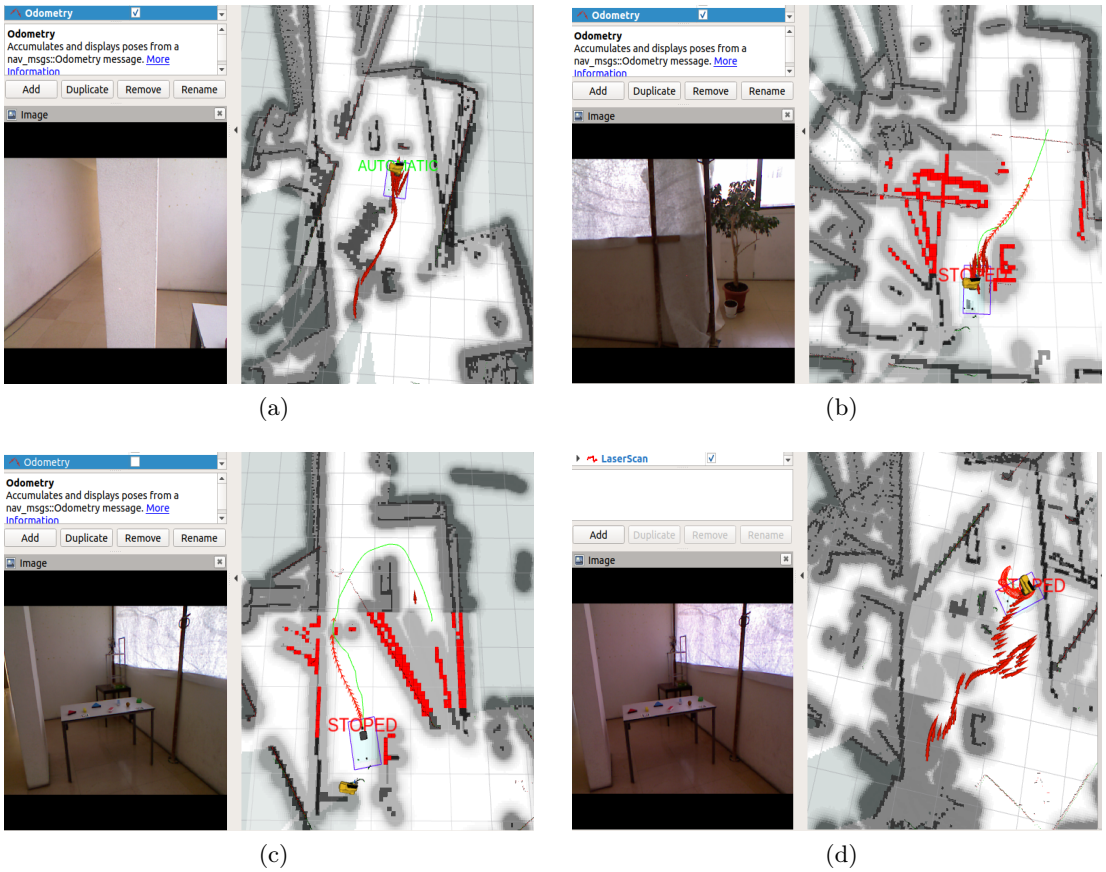


Figura 8.7: Representação das falhas do modo de navegação automático no ambiente 2.

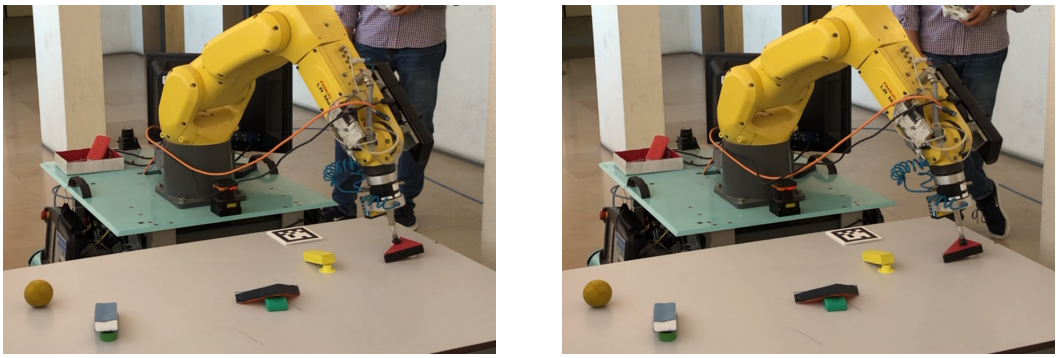
Desta forma, para a contínua utilização do sistema sem o reiniciar, é necessário eliminar o mapa atual e reiniciar o mapeamento. Esta ação pode ser realizada inserindo no terminal o comando:

```
rostopic pub /syscommand std_msgs/String "reset"
```

Houve situações onde o processo de *bin-picking* só conseguiu apanhar os objetos à segunda tentativa. Os testes 12, 16, 18 e 20 foram exemplos destas situações.

As interferências da luz infra-vermelho proveniente da luz solar, ou da falta de consistência de segmentação por parte das funções `PCL`, podem originar as falhas do estado 4. A figura 8.8a representa a primeira tentativa de *picking*, onde metade da ventosa

falha o objeto, enquanto que a figura 8.8b representa a segunda tentativa de *picking*, onde este foi realizado exatamente no centro do objeto alvo. É de evidenciar que não houve qualquer alteração no posicionamento da plataforma ou dos objetos para os dois momentos representados na figura 8.8.



(a) Primeira tentativa. (b) Segunda Tentativa.

Figura 8.8: Representação do processo de *bin-picking* à segunda tentativa.

No entanto, a falha verificada no teste 20 não foi originada pelos mesmos motivos, mas sim por falta de alcance por parte do braço. Durante o estado 3 (orientação), devido à influência das rodas castor, a plataforma deslocou-se ligeiramente para a sua retaguarda. Esta distância foi compensada com a re-execução dos estados 2 e 3, aproximando a plataforma à bancada de trabalho, completando assim a tarefa de *bin-picking* com êxito.

Note-se que o *software* `OpenNI`, por vezes, não carrega os parâmetros intrínsecos presentes nos ficheiros obtidos pela calibração intrínseca. Este problema gera, obviamente, falhas críticas no estado de *bin-picking*, uma vez que usa os parâmetros intrínsecos de *default*, resultando em nuvens de pontos como as representadas na figura 8.9. De forma a prever falhas originadas por este problema, sempre que o sistema foi iniciado para a realização dos testes, foi verificado quais os parâmetros intrínsecos utilizados pelo *driver*.

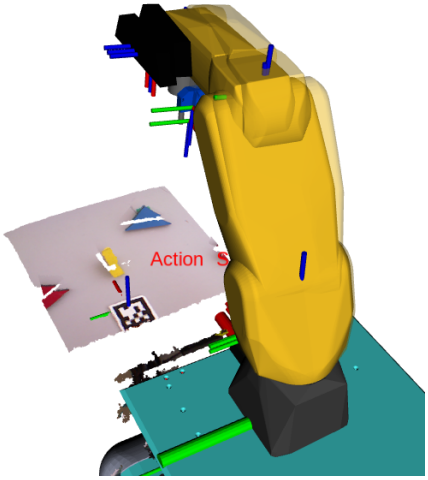


Figura 8.9: Representação da nuvem de pontos obtida com os parâmetros intrínsecos de *default*.

8.4 Análise dos estados propostos em diferentes velocidades

De modo a testar o comportamento da máquina de estados proposta face ao aumento da velocidade, foram efetuados dois testes para cada conjunto de valores de velocidades (designados de T_i). Para alterar estes valores, apenas é necessário alterar os parâmetros referentes a cada velocidade (no ficheiro `checker_params.yaml` pertencente ao pacote `robonuc_integration`).

Os valores de velocidade testados podem ser observados na tabela 8.3. Salienta-se que a velocidade do braço é representada em percentagem ao seu valor máximo, que é de $4\,\mathrm{m\,s^{-1}}$.

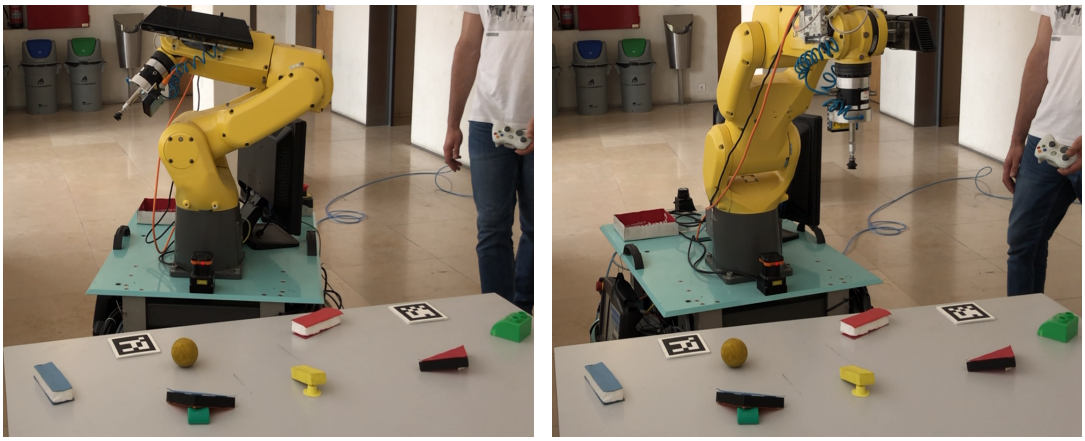
	Default	T_1	T_2	T_3	T_4	T_5	T_6
Vel. estado 2 ($\mathbf{m\,s^{-1}}$)	0.035	0.05	0.08	0.1	0.15	0.15	0.2
Vel. estado 3 ($\mathbf{s^{-1}}$)	0.04	0.05	0.05	0.05	0.05	0.07	0.08
Vel. braço (%)	10	15	30	50	50	50	70

Tabela 8.3: Representação das velocidades testadas para cada grupo T_i.

Para todos os testes efetuados, a velocidade não comprometeu a transição de estados, ou seja, o funcionamento geral da solução proposta. No entanto, a exatidão do sistema nos estados 2 e 3 foi afetada.

No estado 2 (aproximação), em velocidades elevadas (T_4, T_5 e T_6), existe uma maior aproximação da bancada face a velocidades mais reduzidas. Contudo, no momento da travagem "brusca", o manipulador móvel desloca-se alguns centímetros para a sua retaguarda. Este deslocamento posiciona o manipulador a uma distancia à mesa superior à que é verificada com valores de velocidade menores.

Relativamente ao estado 3 (orientação), para o conjunto de valores T_5 e T_6, em que a velocidade angular imposta é superior a $0.05\,\mathrm{s^{-1}}$, a orientação final da plataforma apresenta um erro maior. As figuras 8.10a e 8.10b representam a orientação da plataforma no inicio e após o estado 3, respetivamente.



(a) (b)

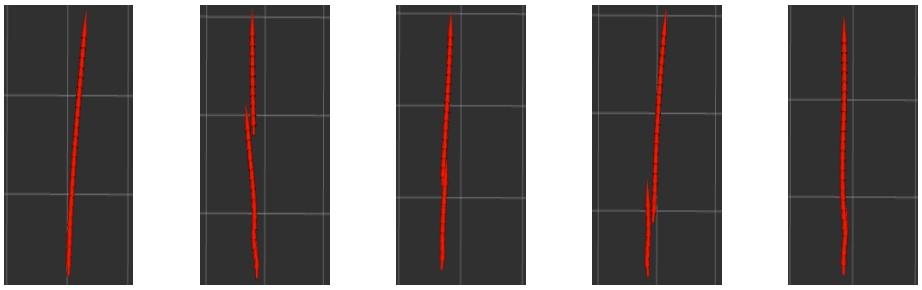
Figura 8.10: Representação da orientação inicial e final da plataforma, com velocidades definidas em T_6.

Este efeito pode ter origem no elevado tempo de processamento da imagem (necessário à orientação), no elevado tempo de resposta do *hardware* e/ou elevada inércia mássica do sistema. Todos estes fatores influenciam o estado de orientação, originando assim uma rotação maior do que a necessária.

Por último, o aumento da velocidade do braço robótico não teve influência em nenhum estado, diminuindo apenas o tempo de ciclo do processo global.

8.5 Análise de uma trajetória "reta"

Como referido anteriormente, no estado 2 é imposta uma velocidade linear positiva até a deteção de um objeto por parte do laser 1-D. Assim, para verificar a influência das rodas castor, foram realizados 5 testes de aproximação retilínea (nas mesmas condições) para uma distância de 2.5 m. A figura 8.11 representa os resultados da hodometria para os testes efetuados.



(a) (b) (c) (d) (e)

Figura 8.11: Representação dos 5 testes de hodometria para uma trajetória retilínea.

Em condições ideais, este trajeto seria retilíneo, no entanto, as rodas castor influenciam o movimento. Tal como se pode observar na figura 8.12, a plataforma foi posicionada no ponto inicial (figura 8.12a) e terminou o percurso numa orientação diferente (figura 8.12b). A hodometria correspondente a este trajeto pode ser visualizada na figura 8.11a.

Em todos os testes efetuados foram observadas ligeiros desvios de orientação originados pelas rodas castor. Contudo, a hodometria representada nas figuras 8.11b, 8.11c, 8.11d apresentam algumas descontinuidades durante o percurso, podendo-se concluir que a hodometria calculada, ou as leituras proveniente dos *enconders*, apresentam algumas falhas.

8.6 Demonstração de trajeto sinusoidal

De forma a demonstrar o potencial de aplicação da cinemática integrada, nesta secção é apresentado o desenvolvimento de uma solução que visa a cooperação entre os dois sistemas (plataforma e braço robótico). Nesse sentido, optou-se por desenvolver ferramentas que permitissem a execução de um movimento sinusoidal ao longo de um plano. Foram desenvolvidos os nós `move_sinusoide.py` e `move_linear_vel_platform.py`, responsáveis pelo o movimento do manipulador e pelo movimento da plataforma, respetivamente.

(a) (b)

Figura 8.12: Representação da posição inicial e final da trajetória realizada num dos testes.

Assim, foi concebida uma função designada de `generate_plan_sinusoide` que aceita como argumentos de entrada o grupo do manipulador (definido pelo *moveit*), a amplitude e o número de períodos da sinusoide, e a orientação do *gripper* durante o movimento. Com estas variáveis, a função calcula todos os pontos de passagem da trajetória e com recurso a ferramentas do `moveit` retorna o plano de movimento.

O nó `move_sinusoide.py` faz uso dessa função para obter o plano e posteriormente executa o movimento no manipulador. Num instante próximo é iniciado o nó `move_linear_vel_platform.py` que, caso o botão *Dead man's switch* esteja pressionado, tem como função enviar velocidades lineares positivas para a plataforma. Caso contrário, as velocidades enviadas serão nulas.

Ambos os nós estão preparados para trabalhar em simultâneo com o nó `r_hybrid` (responsável pela tele-operação e navegação automática). Esta sincronização é possível devido ao tópico anteriormente criado `/RobotStatus`, que apenas permite a navegação quando o estado do robô é publicado como "1". Desta forma, o nó `move_sinusoide.py` inicia o movimento sinusoidal após o utilizador pressionar o botão "X" e apenas o continua caso o botão *Dead man's switch* esteja pressionado durante todo o movimento. Se o botão *Dead man's switch* for largado, o sistema pára, e para recomeçar basta pressionar novamente os mesmos botões.

Deste modo, o sistema está apto para, após ser posicionado, iniciar o movimento proposto as vezes necessárias sem o reiniciar.

A figura 8.13 apresenta o fluxograma relativo ao funcionamento do nó `move_sinusoide.py`.

Com auxilio do laser 1D e de forma a demonstrar o trajeto da ponta do manipulador, foram gravadas as suas sucessivas projeções numa parede ao longo do movimento (figura 8.14). O trajeto resultante é apresentado na figura 8.15. Note-se que a curva representada na parede foi obtida por exposição de longa duração.

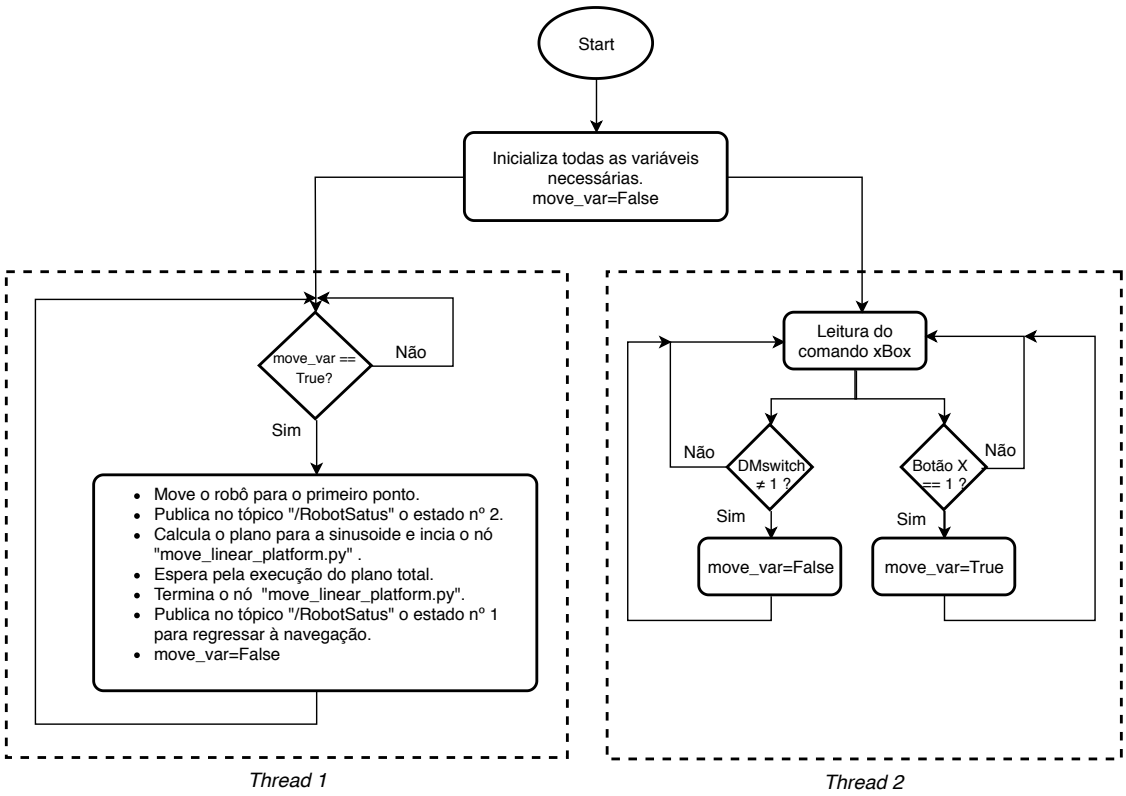


Figura 8.13: Fluxograma inerente ao nó `move_sinusoide.py`.

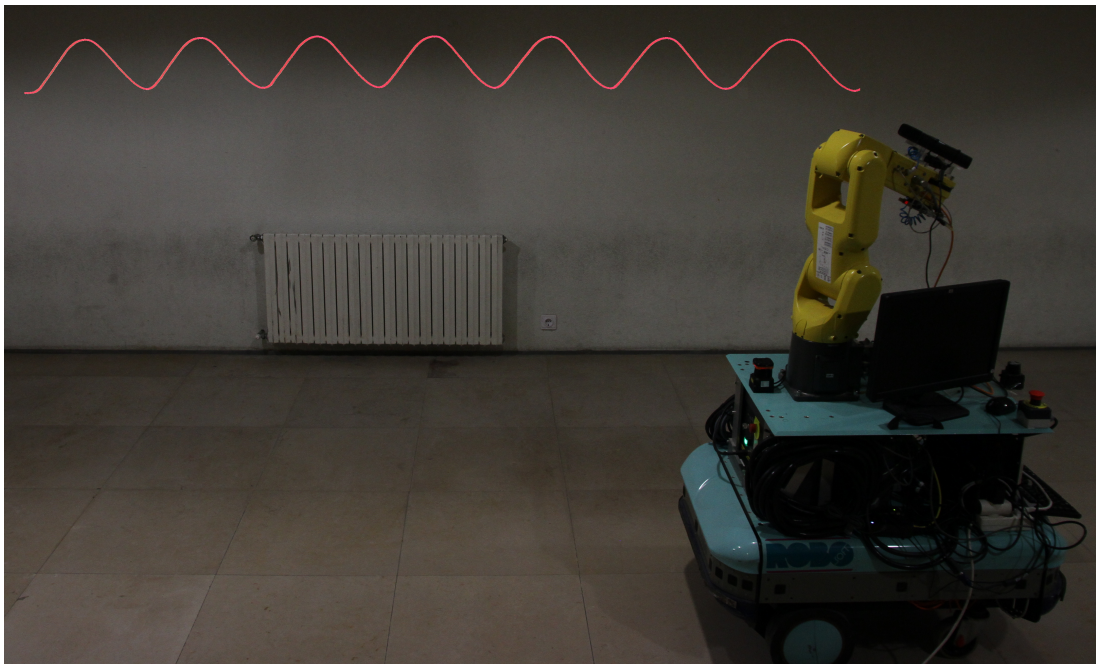


Figura 8.14: Representação da sinusoide resultante no ambiente utilizado.

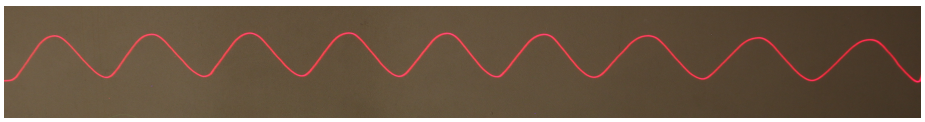


Figura 8.15: Representação da sinusoide resultante do movimento integrado do sistema total.

8.7 Conclusão

Após a análise dos testes efetuados nos diferentes ambientes, constata-se que:

- A tele-operação no modo semi-manual funcionou 100% dos casos, enquanto que o modo automático funcionou em 62% dos casos.
- No ”ambiente 1”, a taxa de sucesso do cumprimento do objetivo (apanhar uma peça) foi de 55%.
- No ”ambiente 2”, nos casos onde o modo automático cumpriu o objetivo, a taxa de sucesso, à primeira tentativa, foi de 75%. No entanto, se forem consideradas duas tentativas, a taxa de sucesso é de 100%.
- A arquitetura proposta, ou seja, o funcionamento da máquina de estados proposta, no ”ambiente 1”, funcionou em 91% dos testes efetuados, enquanto que no ambiente 2 funcionou em 100%.
- O aumento da velocidade não comprometeu o funcionamento da solução proposta.

A integração e transição dos diferentes estados está a funcionar perfeitamente. Sempre que um estado é cancelado o manipulador regressa ao estado 1 (navegação), fornecendo possibilidade de o utilizador reiniciar o processo sempre que necessitar. Salienta-se que em todos os testes efetuados, não houve qualquer tipo de ”congelamento” do sistema nem colisões entre componentes do ROBONUC, o que demonstra a robustez e fiabilidade do sistema implementado.

Outras soluções poderiam ter sido adotadas. Para a fusão de dois estados (aproximação e orientação), pensou-se em estimar a posição e orientação de um marcador ”aruco”, posicionado perpendicularmente à bancada de trabalho e de dimensão considerável, através da kinect. Desta forma, a posição e orientação da bancada seria estimada com base no ”aruco”, e através da árvore de transformações era possível calcular a posição e orientação da bancada relativamente à base da plataforma. Posteriormente, poderia ser usada a ação `MoveBaseAction` fornecida pelos nós de navegação autónoma para movimentação do robô. No entanto, como foi verificado, o modo de navegação automático tem bastantes limitações, especialmente em ambientes pouco estruturados.

Desta forma, pode-se concluir que a máquina de estados proposta apresenta um número adequado de estados para a atingir o objetivo, com elevada taxa de sucesso. A conceção de uma solução com menos estados aparenta ser difícil, enquanto que a utilização de mais estados pode ser excessiva.

Relativamente à demonstração do movimento sinusoidal, conclui-se que a solução proposta cumpre o objetivo e demonstra a aplicabilidade do estudo da cinemática integrada do sistema global.

Capítulo 9

Conclusões e Trabalho Futuro

9.1 Conclusões

Os principais objetivos preconizados desta dissertação consistiram na conceção/alteração de ferramentas de forma a atingir uma integração funcional robusta, na adoção de estratégias de forma a aumentar a sua autonomia energética e na conceção teórica de um modelo de cinemática "global".

De forma a dar resposta aos objetivos mencionados, e compreender pormenorizadamente as ferramentas existentes, para posterior utilização, foi feito um estudo prévio seguido da instalação do *hardware* e *software* de cada subsistema. Esta instalação foi acompanhada com a atualização da versão `ROS`, o que careceu da correção de erros originados pelas diferenças de versões (versão atual e versão original do desenvolvimento).

Posteriormente, foi necessária a integração física do sistema, o que envolveu a montagem dos sistemas, e a instalação do inversor e dos conversores de tensão, devido à necessidade de adotar estratégias de aumento de autonomia energética do robô.

De seguida, procedeu-se à "integração virtual", onde foi necessário calibrar todo o sistema, ou seja, foi necessária a conceção do modelo URDF total com auxilio das calibrações enumeradas (laser 1D-robô; câmara-robô; robô-laser's 2D), bem como as calibrações intrínsecas das câmaras RGB e IR.

Para a integração dos sistemas, com o sistema totalmente integrado fisicamente e virtualmente, procedeu-se à conceção de uma arquitetura integrada, o que posteriormente originou uma máquina de estados. Assim, para alem da criação de novos nós e tópicos, foi necessária a intervenção em ambos os subsistemas desenvolvidos anteriormente. Para aumentar a robustez do sistema implementado foram criadas ações para cada estado do sistema ROBONUC.

Ulteriormente à implementação prática, procedeu-se a um estudo teórico sobre a problemática envolvida em manipuladores móveis e modelou-se cinematicamente o sistema integrado utilizando duas abordagens distintas. Estas abordagens foram implementadas e testadas em `Matlab`.

Posteriormente, foram realizados testes em diferentes ambientes, que permitiram demonstrar a viabilidade e a robustez da arquitetura proposta. Outras soluções poderiam ter sido adotadas, contudo, conclui-se que o número de estados propostos para a máquina de estados é o adequado.

Por ultimo, a fim de mostrar a aplicabilidade de um sistema intra-cooperativo, no qual os subsistemas cooperem entre si, foi realizada uma demonstração do funcionamento

dos dois sistemas em simultâneo.

É possível concluir que os objetivos propostos foram totalmente atingidos, contribuindo assim, para a continuidade do bom desenvolvimento do projeto ROBONUC.

Desta forma é possível enumerar as contribuições do trabalho desenvolvido, da seguinte forma:

- Intervenção física para integração de todo o *hardware*.
- Modelação do ROBONUC para o seu uso em `ROS`.
- Desenvolvimento de uma arquitetura integrada.
- Criação de uma máquina de estados para implementar a arquitetura geral.
- Adaptação no subsistema de navegação.
- Reestruturação do subsistema de *bin-picking*.
- Modelação cinemática do manipulador móvel ROBONUC utilizando duas abordagens.
- Criação de Blog:
  `https://tiagoatavares.github.io/tiagotavares_blog/`.
- Criação de Repositório github:
  `https://github.com/lardemua/Robonuc`.
- *Youtube playlist*:
  `https://youtube.com/playlist?list=PLK0VRJVGJNKGAQHxasco4lEc0a4sJIUX7`.

9.2 Trabalho Futuro

O trabalho desenvolvido resultou na combinação de diferentes tarefas que podem ser melhoradas individualmente, de forma a criar um sistema ainda mais robusto. Por exemplo, para a tarefa de aproximação e orientação à bancada, poderiam ser implementados algoritmos mais sofisticados, de forma a percecionar a bancada e a sua orientação através de reconhecimento de objetos.

De modo a atingir movimentos integrados, ou seja, de modo a que plataforma e o manipulador cooperem entre si, será necessário formular e implementar sistemas de controlo com base na dinâmica do sistema total. Para isso, sugere-se a implementação de *encoders* nas rodas castores para controlar e prever a trajetória da plataforma, ou a sua substituição de forma a conceber uma plataforma omnidirecional.

Uma vez que este tipo de sistemas partilha ambientes com humanos, numa perspetiva de segurança, a substituição do manipulador robótico por um manipulador colaborativo aumentaria significativamente o conforto e ampliaria a gama de tarefas possíveis de implementar.

Devido a alguns problemas não padronizados por parte da plataforma móvel, sugere-se uma reestruturação/verificação dos sistemas de controlo das rodas.

Finalmente, para tornar o sistema totalmente autónomo, sugere-se a instalação de um reservatório de ar comprimido ou a substituição do *gripper* por um elétrico.

Referências

[1] Oliver Brock. *A Historical Perspective*. 2012. URL: http://www.mobilemanipulation.org/index.php/about (acedido em 18/02/2019).

[2] Gilles Foulon, J. -Yves Fourquet e Marc Renaud. «Coordinating mobility and manipulation using nonholonomic mobile manipulators». Em: *Control Engineering Practice* 7.3 (1 de mar. de 1999), pp. 391–399. ISSN: 0967-0661. DOI: 10.1016/S0967-0661(98)00158-0.

[3] Mads Hvilshøj, Simon Bøgh, Oluf Skov Nielsen e Ole Madsen. «Autonomous industrial mobile manipulation (AIMM): past, present and future». Em: *Industrial Robot: An International Journal* 39.2 (2 de mar. de 2012), pp. 120–135. ISSN: 0143-991X. DOI: 10.1108/01439911211201582.

[4] Dov Katz, Emily Horrell, Yuandong Yang, Brendan Burns, Thomas Buckley, Anna Grishkan, Volodymyr Zhylkovskyy, Oliver Brock e Erik Learned-Miller. «The UMass Mobile Manipulator UMan: An Experimental Platform for Autonomous Mobile Manipulation». Em: (2006), p. 8.

[5] Oussama Khatib. «Mobile manipulation: The robotic assistant». Em: *Robotics and Autonomous Systems*. Field and Service Robotics 26.2 (28 de fev. de 1999), pp. 175–183. ISSN: 0921-8890. DOI: 10.1016/S0921-8890(98)00067-0.

[6] Robert Holmberg e Oussama Khatib. «Development and control of a holonomic mobile robot for mobile manipulation tasks». Em: *International Journal of Robotics Research* (2000).

[7] B. Graf, M. Hans e R. D. Schraft. «Mobile robot assistants». Em: *IEEE Robotics Automation Magazine* 11.2 (jun. de 2004), pp. 67–77. ISSN: 1070-9932. DOI: 10.1109/MRA.2004.1310943.

[8] W. Merkt, Y. Yang, T. Stouraitis, C. E. Mower, M. Fallon e S. Vijayakumar. «Robust shared autonomy for mobile manipulation with continuous scene monitoring». Em: *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*. 2017 13th IEEE Conference on Automation Science and Engineering (CASE). Ago. de 2017, pp. 130–137. DOI: 10.1109/COASE.2017.8256092.

[9] Michiel van Osch, Debjyoti Bera, Kees van Hee, Yvonne Koks e Henk Zeegers. «Tele-operated service robots: ROSE». Em: *Automation in Construction* 39 (1 de abr. de 2014), pp. 152–160. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2013.06.009.

[10] Sungman Park, Yeongtae Jung e Joonbum Bae. «An interactive and intuitive control interface for a tele-operated robot (AVATAR) system». Em: *Mechatronics* 55 (1 de nov. de 2018), pp. 54–62. ISSN: 0957-4158. DOI: 10.1016/j.mechatronics.2018.08.011.

[11] *Bin Picking - Blumenbecker*. International industrial service provider. URL: https://www.blumenbecker.com/industrial-automation/industrial-robotics/bin-picking/ (acedido em 28/04/2019).

[12] *Bin Picking - The revolutionary technique for industry 4.0*. Teqram BV. 9 de jun. de 2017. URL: https://teqram.com/robotic-systems/bin-picking/ (acedido em 26/02/2019).

[13] Joana Mota. «Precision Bin-Picking using a 3D Sensor and a 1D Laser Sensor». Dissertação de Mestrado. Universidade de Aveiro, 2018.

[14] Y. Yang, V. Ivan, Z. Li, M. Fallon e S. Vijayakumar. «iDRM: Humanoid motion planning with realtime end-pose selection in complex environments». Em: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids). Nov. de 2016, pp. 271–278. DOI: 10.1109/HUMANOIDS.2016.7803288.

[15] S. Chitta, E. G. Jones, M. Ciocarlie e K. Hsiao. «Mobile Manipulation in Unstructured Environments: Perception, Planning, and Execution». Em: *IEEE Robotics Automation Magazine* 19.2 (jun. de 2012), pp. 58–71. ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2191995.

[16] Nayden Chivarov, Denis Chikurtev, Emanuil Markov, Stefan Chivarov e Peter Kopacek. «Cost Oriented Tele-Controlled Service Robot for Increasing the Quality of Life of Elderly and Disabled - ROBCO 18». Em: *IFAC-PapersOnLine*. 18th IFAC Conference on Technology, Culture and International Stability TECIS 2018 51.30 (1 de jan. de 2018), pp. 192–197. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.11.285.

[17] Max Schwarz, Tobias Rodehutskors, David Droeschel, Marius Beul, Michael Schreiber, Nikita Araslanov, Ivan Ivanov, Christian Lenz, Jan Razlaw, Sebastian Schüller, David Schwarz, Angeliki Topalidou-Kyniazopoulou e Sven Behnke. «NimbRo Rescue: Solving Disaster-response Tasks with the Mobile Manipulation Robot Momaro». Em: *Journal of Field Robotics* 34.2 (1 de mar. de 2017), pp. 400–425. ISSN: 1556-4959. DOI: 10.1002/rob.21677.

[18] Max Schwarz, Marius Beul, David Droeschel, Sebastian Schüller, Arul Selvam Periyasamy, Christian Lenz, Michael Schreiber e Sven Behnke. «Supervised Autonomy for Exploration and Mobile Manipulation in Rough Terrain with a Centaur-Like Robot». Em: *Front. Robot. AI* 3 (2016). ISSN: 2296-9144. DOI: 10.3389/frobt.2016.00057.

[19] M. Nieuwenhuisen, D. Droeschel, D. Holz, J. Stückler, A. Berner, J. Li, R. Klein e S. Behnke. «Mobile bin picking with an anthropomorphic service robot». Em: *2013 IEEE International Conference on Robotics and Automation*. 2013 IEEE International Conference on Robotics and Automation. Mai. de 2013, pp. 2327–2334. DOI: 10.1109/ICRA.2013.6630892.

[20] Bruno Vieira. «Retrofitting of the Robuter's Platform to an AGV With Visual Guidance». Dissertação de Mestrado. Universidade de Aveiro, 2017.

[21] Vítor Silva. «Integração de Manipulador FANUC na Plataforma Robuter para Manipulação Móvel». Dissertação de Mestrado. Universidade de Aveiro, 2017.

[22] Luís Sarmento. «Navegação Assistida e Semi-Autonoma da Plataforma ROBONUC». Dissertação de Mestrado. Universidade de Aveiro, 2018.

[23] Paul Shepanski, Richard Kelly e Timothy Jenkel. «Mobile Manipulation for the KUKA youBot Platform». Em: *Major Qualifying Projects (All Years)* (11 de mar. de 2013).

[24] *KMR iiwa*. KUKA AG. URL: https://www.kuka.com/pt-pt/produtos-servi%C3%A7os/mobilidade/rob%C3%B4s-m%C3%B3veis/kmr-iiwa (acedido em 27/02/2019).

[25] *INESC TEC*. INESC TEC. URL: https://www.inesctec.pt/en (acedido em 19/04/2019).

[26] *Stamina – CRIIS*. URL: http://criis.inesctec.pt/index.php/criis-projects/stamina/ (acedido em 19/04/2019).

[27] *CARLoS – CRIIS*. URL: http://criis.inesctec.pt/index.php/criis-projects/carlos/ (acedido em 19/04/2019).

[28] *ColRobot – CRIIS*. URL: http://criis.inesctec.pt/index.php/criis-projects/colrobot/ (acedido em 19/04/2019).

[29] *Clearpath Robotics: Mobile Robots for Research & Development*. Clearpath Robotics. URL: https://www.clearpathrobotics.com/ (acedido em 19/04/2019).

[30] *GB-BKi7A-7500 (rev. 1.0) | Mini-PC Barebone (BRIX) - GIGABYTE Global*. GIGABYTE. URL: https://www.gigabyte.com/Mini-PcBarebone/GB-BKi7A-7500-rev-10#ov (acedido em 16/03/2019).

[31] *ArduinoLeonardo ETH*. URL: https://store.arduino.cc/arduino-leonardo-eth (acedido em 17/03/2019).

[32] Arduino. *ArduinoMicro*. URL: https://store.arduino.cc/arduino-micro (acedido em 07/04/2019).

[33] *Scanning Rangefinder Distance Data Output/URG-04LX-UG01 Product Details | HOKUYO AUTOMATIC CO., LTD.* URL: https://www.hokuyo-aut.jp/search/single.php?serial=166 (acedido em 07/04/2019).

[34] *UTM-30LX*. ROS Components. URL: http://www.roscomponents.com/en/lidar-laser-scanner/87-utm-30lx.html (acedido em 07/04/2019).

[35] *Xbox 360 Manuals | Xbox 360 Specs | Kinect Manual*. Xbox Support. URL: https://support.xbox.com/en-US/xbox-360/console/manual-specs (acedido em 07/04/2019).

[36] *Router Asus RT-AC51U | Redes*. ASUS Portugal. URL: https://www.asus.com/pt/Networking/RTAC51U/ (acedido em 07/04/2019).

[37] *Robô industrial FANUC LRMate 200iD - Fanuc*. URL: https://www.fanuc.eu/pt/pt/rob%C3%B4s/p%C3%A1gina-filtro-rob%C3%B4s/s%C3%A9rie-lrmate/lrmate-200-id (acedido em 07/04/2019).

[38] *kinect Microsoft Quits Manufacturing Kinect Motion Controller*. eWEEK. URL: https://www.eweek.com/pc-hardware/microsoft-quits-manufacturing-kinect-motion-controller (acedido em 07/04/2019).

[39] *DT20 Hi SICK*. URL: https://www.sick.com/ag/en/distance-sensors/displacement-measurement-sensors/dt20-hi/c/g176377 (acedido em 07/04/2019).

[40] *ArduinoUno Rev3*. URL: https://store.arduino.cc/arduino-uno-rev3 (acedido em 07/04/2019).

[41] *Inversor de potencia DC-AC de instalación fija, 2000W | RS Components*. URL: https://pt.rs-online.com/web/p/inversores-de-potencia-dc-ac-de-instalacion-fija/1793339/ (acedido em 24/05/2019).

[42] *UH700 | USB 3.0 7-Port Hub | TP-Link*. URL: https://www.tp-link.com/en/home-networking/computer-accessory/uh700/?fbclid=IwAR0cZ7-iFC2HoTW5w0%5C-Qv2UnNCXi03ApzxmwGTOELwl3y9fDdelHG1hCXiXE#qrcode (acedido em 24/05/2019).

[43] Francisco Martín, Enrique Soriano e José M. Cañas. «Quantitative analysis of security in distributed robotic frameworks». Em: *Robotics and Autonomous Systems* 100 (1 de fev. de 2018), pp. 95–107. ISSN: 0921-8890. DOI: 10.1016/j.robot.2017.11.002.

[44] *ROS/Introduction - ROS Wiki*. URL: http://wiki.ros.org/ROS/Introduction (acedido em 17/03/2019).

[45] *melodic - ROS Wiki*. URL: http://wiki.ros.org/melodic (acedido em 17/03/2019).

[46] *ROS/Concepts - ROS Wiki*. URL: http://wiki.ros.org/ROS/Concepts (acedido em 17/03/2019).

[47] V. Costa, T. Cunha, M. Oliveira, H. Sobreira e A. Sousa. «Robotics: Using a competition mindset as a tool for learning ROS». Em: *Advances in Intelligent Systems and Computing* 417 (2016), pp. 757–766. DOI: 10.1007/978-3-319-27146-0_58.

[48] *actionlib/DetailedDescription - ROS Wiki*. URL: http://wiki.ros.org/actionlib/DetailedDescription (acedido em 20/04/2019).

[49] J. J. Leonard e H. F. Durrant-Whyte. «Mobile robot localization by tracking geometric beacons». Em: *IEEE Transactions on Robotics and Automation* 7.3 (jun. de 1991), pp. 376–382. ISSN: 1042-296X. DOI: 10.1109/70.88147.

[50] S Samuel Abhishek. «Trajectory Planning of a Mobile Robot». Em: National Conference on Technological Advancements in Mechanical Engineering. Jawaharlal Nehru Technological University, Kakinada, 2016, p. 8.

[51] *hokuyo_node - ROS Wiki*. URL: http://wiki.ros.org/hokuyo_node (acedido em 22/04/2019).

[52] *hector_slam - ROS Wiki*. URL: http://wiki.ros.org/hector_slam (acedido em 22/04/2019).

[53] *teb_local_planner - ROS Wiki*. URL: http://wiki.ros.org/teb_local_planner (acedido em 01/05/2019).

[54] *navigation - ROS Wiki*. URL: http://wiki.ros.org/navigation (acedido em 01/05/2019).

[55] *openni - ROS Wiki*. URL: http://wiki.ros.org/openni (acedido em 02/05/2019).

[56] *Point Cloud Library (PCL)*. URL: http://pointclouds.org/about/ (acedido em 02/05/2019).

[57] *tf - ROS Wiki*. URL: http://wiki.ros.org/tf (acedido em 04/05/2019).

[58] *Description_ Ros_ Industrial*. ROS-Industrial. URL: https://rosindustrial.org/about/description (acedido em 01/05/2019).

[59] *Concepts | MoveIt!* URL: https://moveit.ros.org/documentation/concepts/ (acedido em 20/03/2019).

[60] *sw_ urdf_ exporter - ROS Wiki*. URL: http://wiki.ros.org/sw_urdf_exporter (acedido em 11/05/2019).

[61] Timo Korthals. *Contribute to tik0/amiro_ robot development by creating an account on GitHub*. original-date: 2017-03-14T16:47:27Z. 21 de jan. de 2019. URL: https://github.com/tik0/amiro_robot (acedido em 12/05/2019).

[62] C Cattaneo, G Mainetti e R Sala. «The Importance of Camera Calibration and Distortion Correction to Obtain Measurements with Video Surveillance Systems». Em: *J. Phys.: Conf. Ser.* 658 (18 de nov. de 2015), p. 012009. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/658/1/012009.

[63] *camera_ calibration - ROS Wiki*. URL: http://wiki.ros.org/camera_calibration (acedido em 10/05/2019).

[64] *visp_ hand2eye_ calibration - ROS Wiki*. URL: http://wiki.ros.org/visp_hand2eye_calibration (acedido em 13/05/2019).

[65] *aruco_ detect - ROS Wiki*. URL: http://wiki.ros.org/aruco_detect (acedido em 13/05/2019).

[66] *Industrial/Tutorials/Create_ a_ MoveIt_ Pkg_ for_ an_ Industrial_ Robot - ROS Wiki*. URL: http://wiki.ros.org/Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot (acedido em 23/05/2019).

[67] *fiducials - ROS Wiki*. URL: http://wiki.ros.org/fiducials (acedido em 21/05/2019).

[68] *ROS/Tutorials/MultipleMachines - ROS Wiki*. URL: http://wiki.ros.org/ROS/Tutorials/MultipleMachines (acedido em 20/05/2019).

[69] Zhijun Li e Shuzhi Sam Ge. *Fundamentals in Modeling and Control of Mobile Manipulators*. 0ª ed. CRC Press, 19 de abr. de 2016. ISBN: 978-0-429-10114-4. DOI: 10.1201/b15008.

[70] Sheng Lin e A. A. Goldenberg. «Neural-network control of mobile manipulators». Em: *IEEE Transactions on Neural Networks* 12.5 (set. de 2001), pp. 1121–1133. ISSN: 1045-9227. DOI: 10.1109/72.950141.

[71] H. G. Tanner e K. J. Kyriakopoulos. «Nonholonomic motion planning for mobile manipulators». Em: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065). Vol. 2. Abr. de 2000, 1233–1238 vol.2. DOI: 10.1109/ROBOT.2000.844767.

[72] H. Seraji. «An on-line approach to coordinated mobility and manipulation». Em: IEEE International Conference on Robotics and Automation. Mai. de 1993, 28–35 vol.1. DOI: 10.1109/ROBOT.1993.291957.

[73] F. G. Pin e J.- Culioli. «Multi-criteria position and configuration optimization for redundant platform/manipulator systems». Em: *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications. Jul. de 1990, 103–107 vol.1. DOI: 10.1109/IROS.1990.262375.

[74] W. F. Carriker, P. K. Khosla e B. H. Krogh. «Path planning for mobile manipulators for multiple task execution». Em: *IEEE Transactions on Robotics and Automation* 7.3 (jun. de 1991), pp. 403–408. ISSN: 1042-296X. DOI: 10.1109/70.88151.

[75] T. Sugar e V. Kumar. «Decentralized control of cooperating mobile manipulators». Em: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146). Vol. 4. Mai. de 1998, 2916–2921 vol.4. DOI: 10.1109/ROBOT.1998.680672.

[76] E. Papadopoulos e J. Poulakakis. «Planning and model-based control for mobile manipulators». Em: *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*. Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113). Vol. 3. Out. de 2000, 1810–1815 vol.3. DOI: 10.1109/IROS.2000.895234.

[77] Musa Mailah, Endra Pitowarno e Hishamuddin Jamaluddin. «Robust Motion Control for Mobile Manipulator Using Resolved Acceleration and Proportional-Integral Active Force Control». Em: *International Journal of Advanced Robotic Systems* 2.2 (1 de jun. de 2005), p. 14. ISSN: 1729-8814. DOI: 10.5772/5794.

[78] Mirosław Galicki. «Tracking the Kinematically Optimal Trajectories by Mobile Manipulators». Em: *J Intell Robot Syst* 93.3 (1 de mar. de 2019), pp. 635–648. ISSN: 1573-0409. DOI: 10.1007/s10846-018-0868-7.

[79] Yoshio Yamamoto. «Control and Coordination of Locomotion and Manipulation of a Wheeled Mobile Manipulator». Tese de doutoramento. Philadelphia: University of Pennsylvania, 1994.

[80] Long Jin, Shuai Li, Jiguo Yu e Jinbo He. «Robot manipulator control using neural networks: A survey». Em: *Neurocomputing* 285 (12 de abr. de 2018), pp. 23–34. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.01.002.

[81] Oussama Khatib. «Inertial Properties in Robotic Manipulation: An Object-Level Framework». Em: *The International Journal of Robotics Research* 14.1 (fev. de 1995), pp. 19–36. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/027836499501400103.

[82] Krzysztof Tchoń, Janusz Jakubiak e Robert Muszyński. «REGULAR JACOBIAN MOTION PLANNING ALGORITHMS FOR MOBILE MANIPULATORS». Em: *IFAC Proceedings Volumes*. 15th IFAC World Congress 35.1 (1 de jan. de 2002), pp. 121–126. ISSN: 1474-6670. DOI: 10.3182/20020721-6-ES-1901.00832.

[83] Krzysztof Tchoń e Janusz Jakubiak. «Extended Jacobian inverse kinematics algorithms for mobile manipulators». Em: *Journal of Robotic Systems* 19.9 (2002), pp. 443–454. ISSN: 1097-4563. DOI: 10.1002/rob.10052.

[84] Krzysztof Tchoń e Katarzyna Zadarnowska. «Kinematic dexterity of mobile manipulators: an endogenous configuration space approach». Em: *Robotica* 21.5 (out. de 2003), pp. 521–530. ISSN: 02635747, 14698668. DOI: 10.1017/S0263574703005022.

[85] K. Tchon e J. Jakubiak. «A repeatable inverse kinematics algorithm with linear invariant subspaces for mobile manipulators». Em: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 35.5 (out. de 2005), pp. 1051–1057. ISSN: 1083-4419. DOI: 10.1109/TSMCB.2005.848495.

[86] O. Khatib, K. Yokoi, K. Chang, D. Ruspini, R. Holmberg e A. Casal. «Coordination and decentralized cooperation of multiple mobile manipulators». Em: *Journal of Robotic Systems* 13.11 (1996), pp. 755–764. ISSN: 1097-4563. DOI: 10.1002/(SICI)1097-4563(199611)13:11<755::AID-ROB6>3.0.CO;2-U.

[87] T. Sugar, J. P. Desai, V. Kumar e J. P. Ostrowski. «Coordination of multiple mobile manipulators». Em: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164). Vol. 3. Mai. de 2001, 3022–3027 vol.3. DOI: 10.1109/ROBOT.2001.933081.

[88] H. G. Tanner, S. G. Loizou e K. J. Kyriakopoulos. «Nonholonomic navigation and control of cooperating mobile manipulators». Em: *IEEE Transactions on Robotics and Automation* 19.1 (fev. de 2003), pp. 53–64. ISSN: 1042-296X. DOI: 10.1109/TRA.2002.807549.

[89] B. Bayle, J. -Y. Fourquet e M. Renaud. «Manipulability of Wheeled Mobile Manipulators: Application to Motion Generation». Em: *The International Journal of Robotics Research* 22.7 (1 de jul. de 2003), pp. 565–581. ISSN: 0278-3649. DOI: 10.1177/02783649030227007.

[90] Gilles Foulon, Jean-Yves Fourquet e Marc Renaud. «On Continuous Path Tasks for a 6-DOF Manipulator Mounted on a Nonholonomic Mobile Platform». Em: *Field and Service Robotics*. Ed. por Alexander Zelinsky. London: Springer London, 1998, pp. 344–349. ISBN: 978-1-4471-1275-4 978-1-4471-1273-0. DOI: 10.1007/978-1-4471-1273-0_52.

[91] Daniel Constantin, Marin Lupoae, Cătălin Baciu e Dan-Ilie Buliga. «Forward Kinematic Analysis of an Industrial Robot». Em: (2015), pp. 90–95.

[92] B. Bayle, J.- Fourquet e M. Renaud. «Manipulability analysis for mobile manipulators». Em: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164). Vol. 2. Mai. de 2001, 1251–1256 vol.2. DOI: 10.1109/ROBOT.2001.932782.

[93] Gonçalo Daniel Ribeiro da Silva. «Cinemática Composta de Manipuladores Móveis». Dissertação de Mestrado. Universidade do Porto, 2018.

[94] *IKFast Kinematics Solver — moveit_ tutorials Kinetic documentation*. URL: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html (acedido em 30/05/2019).

[95] P. Beeson e B. Ames. «TRAC-IK: An open-source library for improved solving of generic inverse kinematics». Em: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids). Nov. de 2015, pp. 928–935. DOI: 10.1109/HUMANOIDS.2015.7363472.

[96] *TRAC-IK Kinematics Solver — moveit_tutorials Kinetic documentation*. URL: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/trac_ik/trac_ik_tutorial.html (acedido em 30/05/2019).

[97] *traclabs / trac_ik / trac_ik_examples / src / ik_tests.cpp — Bitbucket*. URL: https://bitbucket.org/traclabs/trac_ik/src/master/trac_ik_examples/src/ik_tests.cpp (acedido em 30/05/2019).

[98] *Extensible Optimization Framework. Contribute to ipab-slmc/exotica development by creating an account on GitHub*. original-date: 2015-10-20T13:24:57Z. 30 de mai. de 2019. URL: https://github.com/ipab-slmc/exotica (acedido em 30/05/2019).

[99] *Flexible Collision Library. Contribute to flexible-collision-library/fcl development by creating an account on GitHub*. original-date: 2012-09-19T16:50:51Z. 30 de mai. de 2019. URL: https://github.com/flexible-collision-library/fcl (acedido em 30/05/2019).

Apêndice A

Execução dos processos desenvolvidos

A.1 *Setup* inicial

Antes de executar o processo desenvolvido é necessário assegurar as configurações iniciais. Assim, para execução do projeto apresentado deve-se:

1. Assegurar a carga das baterias do ROBONUC.
2. Verificar o estado do disjuntor da plataforma (presente junto à roda dianteira do lado direito), responsável pela passagem de corrente da bateria para os restantes componentes.
3. Verificar a ligação dos componentes ao inversor: controlador FANUC, kinect, transformador (LASER 1D e 2D), hub, computador.
4. Ligar: o inversor, a plataforma, o computador e o controlador FANUC.
5. Verificar ligações ao hub: Laseres Hokuyo 2D, arduíno para leituras do laser 1D, ligação a arduíno para controlo dos IO's e recetor *xBox*.
6. Verificar a designação atribuída às entradas usb dos lasers. Por *default* o ACMtty0 corresponde ao laser 2D traseiro, o ACMtty1 ao laser 2D dianteiro e o ACMtty2 ao laser 1D.
7. Verificar ligação da kinect ao mini-pc . Estritamente necessário ser na porta 3.0.
8. Verificar ligações ao *switch* e a ligação *ethernet* para o mini-pc.
9. No TP: correr o programa MONIO em *background logic* (ver figura A.1), correr o ROS server e pressionar o botão de *cycle start Setup*.
10. Lançar o sistema com a linha de código:

```
roslaunch robonuc_integration
    robonuc_integration.launch rviz:=true
```

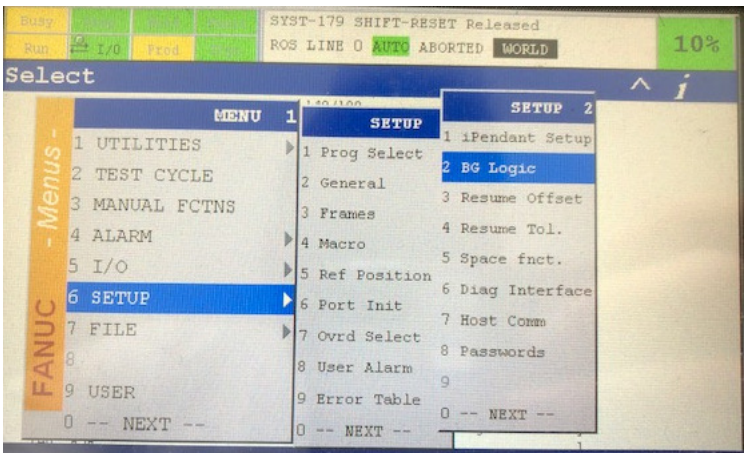


Figura A.1: Representação do menu do *teach pendant* para executar o *Background logic*.

A.2 Calibração extrínseca: Kinect

Os ficheiros necessários à calibração da câmara estão presentes na pasta `camera_calibration` pertencente ao pacote `robonuc_integration` .

Para proceder à calibração da camara Kinect apenas é necessário seguir os seguintes passos:

1. Lançar o `camera_extr_calibration.launch` responsável pelo o lançamento dos nós necessários e da interface.
2. Correr o ficheiro `get_extr_camera_calibration.py`.
3. Mover o robô e pressionar a tecla ”enter” para guardar a posição.
4. Repetir o passo anterior vezes suficientes.
5. Para terminar pressionar a tecla ’y’.
6. Copiar a transformação apresentada para o URDF, `binpicking_macro.xacro`, presente no pacote `bin_picking`. (Ver (listagem 5.1))

A.3 *Launch* Files

A.3.1 integrated_referee.launch

O *launch* file principal `integrated_referee.launch`:

```
<?xml version="1.0"?>
<launch>
    <arg name="rviz" default="false"/>

    <!-- Launch the hybrid_launch  (with joy node for xbox )-->
    <include file="$(find robonuc_integration)/launch/robonuc_platform.
        launch">
        <arg name="rviz_arg" value="$(arg rviz)"/>
    </include>
```

```
<!-- SensorRs232 -->
<include file="$(find␣robonuc_integration)/launch/sensorRS232_v2.launch
    " />

<!-- Launch laser_limit parameter for automatic perception  -->
<rosparam command="load" file="$(find␣robonuc_integration)/param/
    checker_params.yaml"/>
<!-- Node to check check_feasibility based on laser_limit -->
<node name="check_feasibility" pkg="robonuc_integration" type="
    check_feasibility" output="screen"/>

<!-- Launch the referee! Decides to move or to stop platform, and
    execute or not B-picking  -->
<node name="integrated_referee2" pkg="robonuc_integration" type="
    integrated_referee2" output="screen"/>

<!-- moveit lancuh and robot control -->
<include file="$(find␣robonuc_integration)/launch/moveit_fanuc.launch"
    />

<!-- moveit interface -->
<group if="$(arg␣rviz)">

    <include file="$(find␣moveit_ROBONUC_pack)/launch/moveit_rviz.
        launch">
        <arg name="config" value="true"/>
    </include>

</group>

<!-- action serv to control robot status -->
<node pkg="robonuc_action" type="robot_status_server2.py" name="
    robot_status_server2" />

<!-- action serv for robot aproxmation -->
<node name="GetPlatformLaserAproximation" pkg="
    robonuc_aprox_laser_action" type="GetPlatformLaserAproximation"
    output="screen"/>

<!-- action serv for robot orientation -->
<node name="GetPlatformOrientation" pkg="
    robonuc_plat_orientation_action" type="GetPlatformOrientation"
    output="screen"/>

<!-- Kinect Drivers -->
<include file="$(find␣robonuc_integration)/launch/kinect.launch" />

<!-- launch the aruco detection -->
<include file="$(find␣aruco_detect)/launch/aruco_detect.launch">
    <arg name="camera" value="camera/rgb" />
    <arg name="image" value="image_raw" />
    <arg name="fiducial_len" value="0.08" />
    <arg name="dictionary" value="16" />
</include>
```

```
    <!-- action serv for bin_picking -->
    <node name="GetBinPicking" pkg="binpicking_action" type="GetBinPicking.
        py" output="screen"/>

    <!-- Services for B-picking process -->
    <include file="$(find␣robonuc_integration)/launch/Picking_services.
        launch" />

    <!-- Node to control FANUC I/Os -->
    <node name="vs_IO_client" pkg="robonuc" type="vs_IO_client" />


</launch>
```

A.3.2 camera_extr_calibration.launch

Launch file necessário à calibração:

```
<?xml version="1.0"?>
<launch>
    <!-- launch the aruco detection -->
    <include file="$(find␣aruco_detect)/launch/aruco_detect.launch" >
        <arg name="camera" value="camera/rgb" />
        <arg name="image" value="image_raw" />
        <arg name="fiducial_len" value="0.163" />
        <arg name="dictionary" value="16" />
    </include>

    <!-- launch the robot stream interface -->
    <arg name="robot_ip" default="192.168.0.231" doc="IP␣of␣controller" />
    <arg name="J23_factor" default="1" doc="Compensation␣factor␣for␣joint␣
        2-3␣coupling␣(-1,␣0␣or␣1)" />
    <arg name="use_bswap" default="true" doc="If␣true,␣robot␣driver␣will␣
        byte-swap␣all␣incoming␣and␣outgoing␣data" />

    <rosparam command="load" file="$(find␣bin_picking)/config/joint_names.
        yaml" />

    <include file="$(find␣fanuc_driver)/launch/robot_state.launch">
        <arg name="robot_ip" value="$(arg␣robot_ip)" />
        <arg name="J23_factor" value="$(arg␣J23_factor)" />
        <arg name="use_bswap" value="$(arg␣use_bswap)" />
    </include>

    <node name="robot_state_publisher" pkg="robot_state_publisher" type="
        robot_state_publisher" />

    <include file="$(find␣bin_picking)/launch/load_bin_picking.launch" />

    <node name="rviz" pkg="rviz" type="rviz" args="-d␣$(find␣bin_picking)/
        config/robot_state_visualize.rviz" required="true" />

    <!-- launch the calibrator node -->
    <node pkg="visp_hand2eye_calibration" type="
        visp_hand2eye_calibration_calibrator" name="calibrator_node" />

    <!-- launch the kinetic -->
```

```
    <include file="$(find␣robonuc_integration)/launch/kinect.launch"/>

</launch>
```

A.3.3 display_total_urdf.launch

Para visualizar o modelo virtual criado, lançar o `display_total_urdf.launch`:

```
<?xml version="1.0"?>
<launch>

    <!-- Launch URDF -->
    <param name="robot_description" command="$(find␣xacro)/xacro␣--inorder␣
        '$(find␣robonuc_integration)/urdf/robonuc.xacro'" />

    <!-- Robot_state_publisher-->
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="
        state_publisher" />

    <!-- Joint_state_publisher-->
    <param name="use_gui" value="true" />
    <node name="joint_state_publisher" pkg="joint_state_publisher" type="
        joint_state_publisher" />

    <!-- start RViz configuration-->
    <node name="rviz" pkg="rviz" type="rviz" args="-d␣$(find␣
        robonuc_integration)/config/rviz_show_urdf_total.urdf.rviz"
        required="true" />

</launch>
```

A.4 Cinemática integrada

Todos os ficheiros desenvolvidos, relativos à simulação em *matlab*, estão presentes na pasta `cinematica_integrada_matlab`.

Descreve-se, de seguida, a função de cada ficheiro.

- `cinematica_aprox1.m` e `cinematica_aprox2.m`: cálculo e animação das trajétorias propostas, para a abordagem 1 e abordagem 2, respetivamente.
- `GetJacobian_aprox1.m` e `GetJacobian_aprox2.m`: cálculo do jacobiano simbólico (total) para a aboragem 1 e abordagem 2, respetivamente.
- `Getjacobian_only_manipulator.m`: cálculo do jacobiano simbólico (apenas do braço róbotico).
- `Jacobian_manipulator`, `Jacobian_robonuc_aprox1` e `Jacobian_robonuc_aprox1`: funções que aceitam como entrada os valores de juntas, e retornam o jacobiano numérico para o braço robótico, para o sistema total da abordagem 1 e para o sistema total da abordagem 2, respetivamente.
- `JacobianInv_robonuc_aprox1.m` e `JacobianInv_robonuc_aprox1.m`: funções que aceitam como entrada os valores de juntas, e retornam o jacobiano numérico inverso para o sistema total da abordagem 1 e abordagem 2, respetivamente.

- `Manipulabilidade.m`: função que aceita como entrada o número da abordagem a utilizar (1 ou 2) e os respetivos valores de juntas, de forma a calcular a manipulabilidade do ssitema total para a configuração fornecida.
- `Manipulabilidade_manipulador.m`: função que aceita como entrada os valores de juntas do braço robótico, de forma a calcular a manipulabilidade para a configuração fornecida.

É de salientar que a pasta `lib` (pertencente à pasta `cinematica_integrada_matlab`) contém funções desenvolvidas para auxilio da implementação prática e das animações concebidas.

A.5 Demonstração Sinusoide

Os ficheiros necessários à demonstração do movimento sinusoidal estão presentes na pasta `demonstracao_sinusoide`, contida no pacote `robonuc_integration`. Para inicializar todos os nós necessários ao movimento, é necessário executar o ficheiro `robonuc_sinusoide.launch`:

```
<?xml version="1.0"?>
<launch>
    <arg name="rviz" default="true"/>

    <!-- Launch the hybrid_launch  (with joy for xbox )-->
    <include file="$(find robonuc_integration)/launch/robonuc_platform.
        launch">
        <arg name="rviz_arg" value="$(arg rviz)"/>
    </include>

    <!-- moveit lancuh and robot control -->
    <include file="$(find robonuc_integration)/launch/moveit_fanuc.launch"
        />

    <group if="$(arg rviz)">

        <include file="$(find moveit_ROBONUC_pack)/launch/moveit_rviz.
            launch">
            <arg name="config" value="true"/>
        </include>
    </group>

    <!-- move_sinusoide -->
    <node pkg="robonuc_integration" type="move_sinosoide.py" name="
        move_sinosoide.py" />

</launch>
```

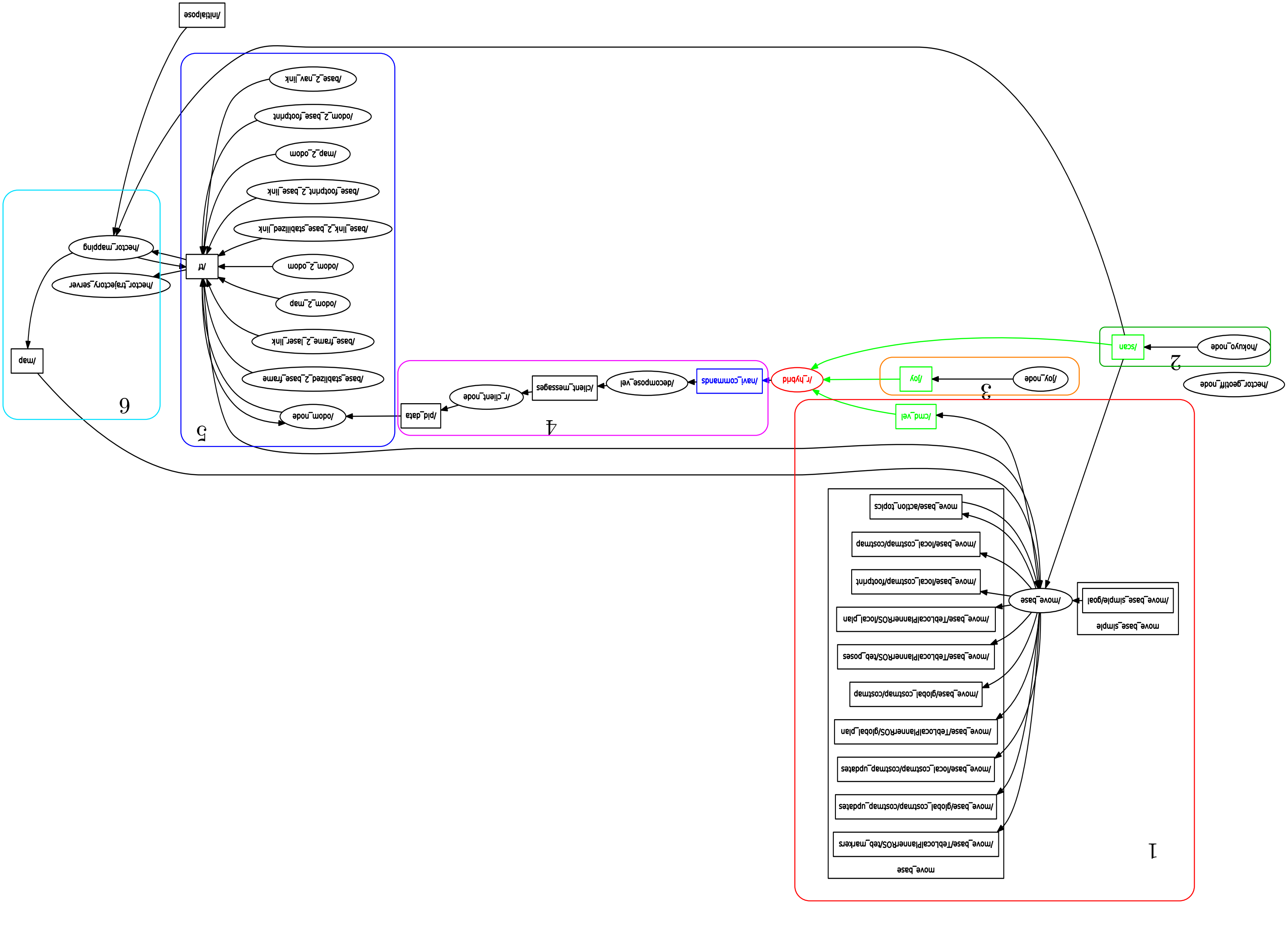
Desta forma, o sistema está apto a realizar a navegação, no modo automático e semi-manual, bem como a executar o movimento sinusoidal. Para executar este movimento, basta pressionar o botão *Dead man's switch* (durante todo o movimento), e clicar apenas uma vez no "botão X" . Caso o *Dead man's switch* deixe de ser pressionado, o sistema vai parar, e para reiniciar o movimento basta pressionar novamente os botões referidos.

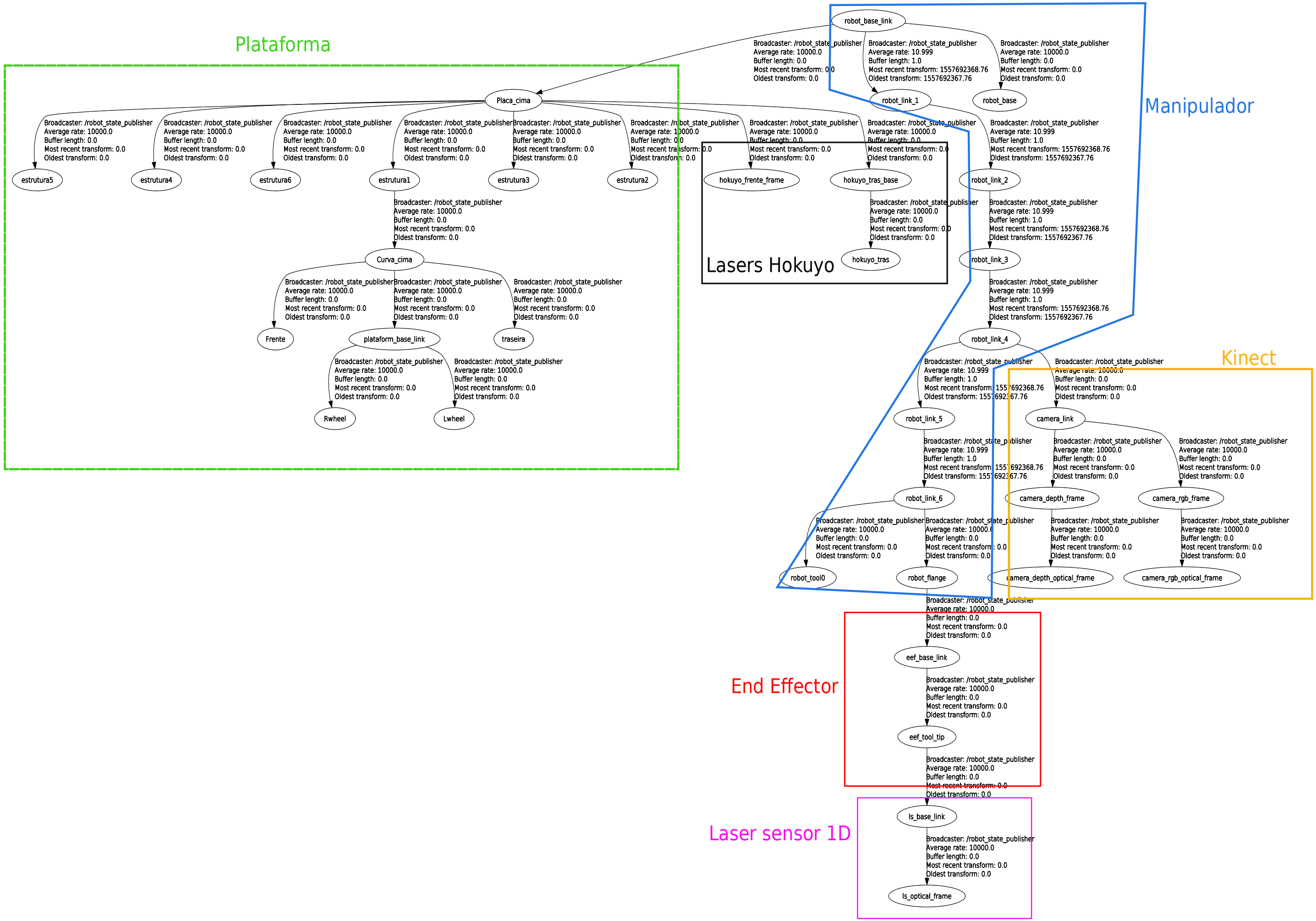
Apêndice B

Ficheiros Auxiliares

B.1 Diagrama de Navegação

B.2 Árvore de transformações do ROBONUC


/initialpose
/hector_mapping
/hector_trajectory_server
/map
6
/tf
/odom_2_map
/base_frame_2_laser_link
/base_stabilized_2_base_frame
/odom_node
/base_link_2_base_stabilized_link
/base_footprint_2_base_link
/map_2_odom
/odom_2_base_footprint
/base_2_nav_link
5
/pid_data
/r_client_node
/client_messages
/decompose_vel
/nav_commands
/r_hybrid
4
/joy
/joy_node
3
/scan
/hokuyo_node
2
/hector_geotiff_node
/cmd_vel
move_base
/move_base/TebLocalPlannerROS/teb_markers
/move_base/global_costmap/costmap_updates
/move_base/local_costmap/costmap_updates
/move_base/TebLocalPlannerROS/global_plan
/move_base/global_costmap/costmap
/move_base/TebLocalPlannerROS/teb_poses
/move_base/TebLocalPlannerROS/local_plan
/move_base/local_costmap/footprint
/move_base/local_costmap/costmap
/move_base/action_topics
/move_base
move_base_simple
/move_base_simple/goal
1


Plataforma
Placa_cima
estrutura5
estrutura4
estrutura6
estrutura1
estrutura3
estrutura2
Curva_cima
Frente
plataform_base_link
traseira
Rwheel
Lwheel
Manipulador
robot_base_link
robot_link_1
robot_base
robot_link_2
robot_link_3
robot_link_4
robot_link_5
robot_link_6
robot_tool0
robot_flange
Lasers Hokuyo
hokuyo_frente_frame
hokuyo_tras_base
hokuyo_tras
Kinect
camera_link
camera_depth_frame
camera_rgb_frame
camera_depth_optical_frame
camera_rgb_optical_frame
End Effector
eef_base_link
eef_tool_tip
Laser sensor 1D
ls_base_link
ls_optical_frame