FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sistema de navegação para plataforma móvel omnidirecional

Fernando Jorge Marques de Sá

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Doutor Paulo Costa

Coorientador: Eng. Héber Sobreira

28 de Julho de 2016

Resumo

Os robôs móveis omnidirecionais para aplicações industriais estão em grande crescimento como se pode constatar pelos novos modelos de marcas consagradas como a KUKA. No entanto os custos associados ainda são muito elevados. Este trabalho enquadra-se num projeto que pretende desenvolver um robô móvel omnidirecional mas com conceitos Lean, ou seja usando sensores e atuadores adequados de modo a ter-se um produto a custos adequados ao mercado das pequenas e médias empresas e não apenas às grandes empresas.

Com este trabalho pretende-se estudar as plataformas omnidirecionais com rodas Mecanum e as suas vantagens em relação a outro tipo de plataformas. Foi calculado o modelo cinemático deste tipo de configuração e um modelo de odometria que permite a estimação da posição do robô baseada no deslocamento das rodas.

Pretende-se ainda estudar a utilização do protocolo CANopen como meio de comunicação para o controlador de motores e a sua integração em ROS (*Robot Operating System*). Foi ainda desenvolvida uma aplicação para validação da comunicação com um dos controlador dos motores utilizados no projeto da plataforma.

Foi ainda, desenvolvido e testado um seguidor de caminhos para plataformas omnidirecionais que pretende seguir um determinado caminho. Os caminhos implementados foram do tipo segmento de reta e o arco de circunferência.

Por fim, desenvolveu-se um protótipo de uma plataforma omnidirecional com rodas Mecanum, tendo como base a plataforma Discovery Q2 da Hangfa. Nesta plataforma foi testado o sistema de navegação desenvolvido, que consiste na integração de um sistema de localização baseado em contornos e odometria e a aplicação do seguidor de caminhos desenvolvido.

Abstract

Omnidirectional mobile robots are dedicated to industrial applications and are expanding as we can see by the new models of established brands, such as KUKA. However, the associated cost is still very high. This project aims to develop an omnidirectional mobile robot but with concepts referent to Lean, this means that by using suitable sensors and actuators we can produce an omnidirectional robot at reasonable and suitable costs to be able to reach the small and medium enterprises, and not only the big companies.

This project work was intended to study the platforms Mecanum omnidirectional wheels and their advantages over other types of platforms. Based on the movement of the wheels a kinematic model of this type of configuration and odometry model was calculated.

The aim is also to study the use of CANopen protocol as a method of communication to the motor controller and its integration in ROS. An application was developed for validation of the comunication with the motor controllers used with the project platform.

It was also developed and tested one path controller for omnidirectional platforms that aims to follow a certain direction. Straight lines and circular arcs were implemented as paths.

Then, an omnidirectional platform prototype was developed with Mecanum wheels, having in consideration the Discovery Q2 platform from Hangfa. In this platform, the localization system based on contours and odometry was integrated with the developed application of the path controller. At last, this navigation system was tested.

Agradecimentos

Agradeço especialmente aos meus pais por todo o apoio, compreensão e "investimento"ao longo do meu curso e especialmente nos momentos de mais difíceis.

Agradeço ao Prof. Doutor Paulo Costa por todo o apoio prestado ao longo do projeto e em especial pelos conhecimentos e orientações proferidas.

Ao Eng. Héber Sobreira, surfista e "grandiosíssimo deus mestre da engenharia", pelo o apoio prestado, paciência e disponibilidade durante todo o projeto.

Ao Prof. Doutor António Paulo Moreira pela oportunidade de fazer este trabalho no INESC TEC.

E por fim, e nada menos importante, agradeço aos meus colegas que me acompanharam ao longo deste curso e que contribuíram para parte daquilo que sou hoje como pessoa. Em especial Edgar Bicho, Ivo Sousa, Joana Dias, João Correia Pinto, João Sá, Miguel Freitas, Nelson Silva, Nuno Rodrigues, Patrick de Sousa, Renato Fonseca e Ricardo F. da Silva.

Fernando Sá

"Success is not final, failure is not fatal.
It is the courage to continue that counts."

Winston Churchill

Conteúdo

1 Introdução **1**
1.1 Estado da Arte . . . 1
1.1.1 Mobilidade omnidirecional . . . 3
1.2 Enquadramento do projeto . . . 7
1.3 Estrutura da Dissertação . . . 7

2 Locomoção omnidirecional **9**
2.1 Cinemática . . . 10
2.2 Modelo de odometria . . . 12

3 Seguimento de Caminhos **15**
3.1 Definição de Caminho . . . 15
3.2 Controlador . . . 17
3.2.1 Segmento de reta . . . 18
3.2.2 Arco de circunferência . . . 21
3.3 Implementação em ROS . . . 23
3.4 Teste do controlador . . . 24
3.4.1 Resultados do controlador para segmentos de reta . . . 25
3.4.2 Resultados do controlador para arco de circunferência . . . 28
3.4.3 Conclusões . . . 31

4 Protótipo Industrial **33**
4.1 CANopen . . . 35
4.1.1 Rede de Campo CAN . . . 36
4.1.2 Camada de aplicação . . . 37
4.1.3 CiA 402 . . . 38
4.2 CANopen em ROS . . . 39
4.2.1 Implementação . . . 40

5 Protótipo Discovery Q2 **41**
5.1 Prototipo inicial . . . 41
5.1.1 Rodas . . . 42
5.1.2 Motores . . . 42
5.1.3 Bloco de Apoio . . . 43
5.1.4 Fonte de energia e Comunicações . . . 43
5.1.5 Controlador dos motores . . . 43
5.2 Modificações . . . 44
5.2.1 Interface Discovery Q2 . . . 46

5.2.2 Laser . . . 48
5.2.3 Computador . . . 48
5.3 Software Implementado . . . 49
5.3.1 Microcontrolador . . . 49
5.3.2 Nó Driver Discovery Q2 . . . 51
5.3.3 Nó Omnijoy . . . 52
5.4 Teste do protótipo . . . 55
5.4.1 Mapeamento . . . 55
5.4.2 Sistema de navegação . . . 55

6 Conclusões e Trabalho Futuro **65**
6.1 Conclusões . . . 65
6.2 Trabalho Futuro . . . 65

A Ficheiros de arranque **67**
A.1 Nó Driver Discovery Q2 . . . 67
A.2 Nó Omnijoy . . . 67
A.3 Nó hector_mapping . . . 68
A.4 Nó hokuyo_node . . . 69
A.5 Simulador . . . 69

B Código do microcontrolador **71**

C Esquemático Interface Discovery Q2 **75**

Referências **77**

Lista de Figuras

1.1 Roomba- Robô móvel de limpeza . . . 2
1.2 AGV com tração traseira reboca 22,5 toneladas num depósito de alimentos . . . 2
1.3 KMP OMNIMOVE - Plataforma omnidirecional da KUKA . . . 3
1.4 Rodas convencionais: (a) Rodízio, (b) Roda direcional . . . 4
1.5 Rodas especiais: (a) Universal, (b) Mecanum . . . 4
1.6 Configurações: (a) Robô com três rodas, (b) Robô com quatro rodas . . . 5
1.7 Configuração mecânica de robô com rodas Mecanum . . . 6

2.1 Graus de liberdade da roda Mecanum . . . 9
2.2 Configuração das rodas e velocidades do robô . . . 11
2.3 Deslocamento associado a um robô omnidirecional com quatro rodas Mecanum . 13

3.1 Definição da posição do robô . . . 16
3.2 Arco de circunferência . . . 17
3.3 Entradas e Saídas do controlador . . . 18
3.4 Erros associados ao segmento de reta . . . 18
3.5 Diagrama de Atividade do controlador segmento de reta e arco de circunferência 19
3.6 Erros associados ao arco de circunferência . . . 22
3.7 Interface do nó Path Controller (/Controller_path) . . . 24
3.8 Simulador Stage com robô discoveryq2 . . . 25
3.9 Nós e tópicos da simulação . . . 25
3.10 Simulação 1 - posição x,y . . . 26
3.11 Simulação 1 - erro rpy . . . 26
3.12 Simulação 1 - $erro_\theta$ . . . 27
3.13 Simulação 2 - posição x,y . . . 27
3.14 Simulação 2 - erro rpy . . . 28
3.15 Simulação 3 - $erro_\theta$ . . . 28
3.16 Simulação 4 - posição x,y . . . 29
3.17 Simulação 4 - erro rpy . . . 29
3.18 Simulação 5 - posição x,y . . . 30
3.19 Simulação 5 - erro rpy . . . 30
3.20 Simulação 5 - $erro_\theta$ . . . 31

4.1 Estrutura do AVG sem rodas . . . 33
4.2 AVG em fase de montagem . . . 34
4.3 Controlador e placa de interface . . . 34
4.4 Adaptador CAN . . . 35
4.5 Estrutura das tramas de dados CAN . . . 36
4.6 Acesso prioritário à rede . . . 37

4.7 Estrutura "ros_canopen" 39

5.1 Dimensões da plataforma 41
5.2 Estrutura da plataforma 42
5.3 Esquema elétrico do módulo IFB1205 44
5.4 Visão geral do protótipo desenvolvido 44
5.5 Protótipo Discovery Q2 45
5.6 Protótipo Discovery Q2 (vista topo) 45
5.7 Placa Interface Discovery Q2 47
5.8 Esquema parcial da placa Discovery Q2 47
5.9 Interface Discovery Q2 montada no robô 48
5.10 Sinais A e B dos encoders 50
5.11 Interface do nó Driver Discovery Q2 (/driver_motors) 51
5.12 *Joystick* utilizado para movimentar o robô 53
5.13 Interface do nó Omnijoy (/omnijoy) 53
5.14 Nós e tópicos ativos durante mapeamento 55
5.15 Mapa resultante 56
5.16 Nós e tópicos ativos durante teste do controlador de caminhos 56
5.17 Teste 1 - erro rpy 57
5.18 Teste 2 - posição x,y 58
5.19 Teste 2 - erro rpy 58
5.20 Teste 3 - posição x,y 59
5.21 Teste 3 - erro rpy 59
5.22 Teste 4 - erro rpy 60
5.23 Teste 4 - $erro_\theta$ 60
5.24 Teste 5 - posição x,y 61
5.25 Teste 5 - erro rpy 61
5.26 Teste 5 - $erro_\theta$ 62
5.27 Teste 6 - $erro_\theta$ 62
5.28 Teste 7 - posição x,y 63
5.29 Teste 7 - erro rpy 63
5.30 Teste 8 - posição x,y 64
5.31 Teste 8 - erro rpy 64

Lista de Tabelas

1.1 Tabela comparativa das características das rodas 6

3.1 Elementos de caminho do tipo segmento de reta 16
3.2 Elementos de caminho do tipo arco de circunferência 17

4.1 Caraterísticas do motor 35

5.1 Características das rodas 42
5.2 Características dos motores 43
5.3 Características do Arduino Mega 2560 46
5.4 Características do Laser URG-04LX-UG01 48
5.5 Características do "Raspberry Pi3" 49
5.6 Incremento do valor da posição 50
5.7 Formato do pacote de dados 52
5.8 Parâmetros do nó Driver Discovery Q2 53
5.9 Parâmetros do nó Omnijoy 54

Abreviaturas e Símbolos

A	Amperes
IDE	Ambiente de desenvolvimento integrado
Kg	Kilograma
ppr	Impulsos por rotação
m	Metros
mm	Milímetros
m/s	Metros por segundo
rad	Radianos
rpm	Rotações por minuto
V	Volts
AGV	Veículos Guiados Automaticamente

Capítulo 1

Introdução

A evolução da robótica tem permitido a sua integração no nosso quotidiano. A sua utilização na indústria oferece um compromisso excelente entre produtividade e flexibilidade, assim, os sistemas robotizados realizam muitas vezes tarefas repetitivas e penosas para o ser humano com maior rapidez e qualidade, sendo uma mais-valia a sua integração nos processos de fabrico. O papel desempenhado por estes robôs nas indústrias mais recentes permite-nos destacar quatro vantagens: o aumento da produtividade, a alta flexibilidade, a elevada qualidade e a melhoria da segurança [1]. Os robôs móveis surgiram com a necessidade de transporte eficiente de materiais entre diferentes zonas da cadeia de produção.

Neste trabalho, pretende-se estudar a configuração Mecanum para plataformas omnidirecionais, bem como, a utilização de CANopen como protocolo de comunicação para o controlador dos motores. Será ainda implementado um controlador de caminhos e desenvolvido um protótipo onde serão efetuados teste do controlador.

1.1 Estado da Arte

A robótica móvel é uma área em grande expansão e crescimento como se pode constatar pela aposta de grandes marcas a nível da indústria como a KUKA, Savant automation, America in Motion, entre outros, mas também em aplicações domésticas e de entretenimento.

Na figura 1.1 é possível observar um robô móvel doméstico de grande sucesso que tem como função a limpeza e aspiração do piso.

Figura 1.1: Roomba- Robô móvel de limpeza

A utilização de robôs móveis na industria tem adquirido cada vez mais importância em tarefas como o transporte de cargas. Este tipo de robôs, muitas vezes denominado por AGVs, é normalmente autónomo e dotado de sensores que permitem a sua localização e a deteção de obstáculos, possuindo também capacidade de navegação. Estes operam em rotas predefinidas executando de forma autónoma as tarefas para o qual foram designados, como o transporte de cargas para linhas de produção ou armazenamento.

Na figura 1.2 é possível observar um robô móvel utilizado para rebocar grandes cargas dentro de uma fábrica de alimentos.

Figura 1.2: AGV com tração traseira reboca 22,5 toneladas num depósito de alimentos

Um dos requisitos fundamentais destes robôs é a capacidade de se movimentar em todo o espaço operacional. Desta forma, devido a sua elevada manobrabilidade, os robôs omnidirecionais possuem um grande potencial de aplicação na área da robótica móvel.

As plataformas móveis são normalmente concebidas para se movimentarem num plano, como um piso de armazém, sendo que neste espaço bidimensional, um corpo tem três graus de liberdade.

O termo omnidirecional descreve a capacidade de uma plataforma se mover em qualquer direção a partir de qualquer configuração de forma instantânea, ou seja, de controlar de forma independente todos os graus de liberdade no espaço bidimensional.

As plataformas móveis com rodas convencionais não são capazes de se mover paralelamente ao seu eixo, como é o caso dos robôs de tração diferencial ou tipo carro. Esta restrição é chamada de não holonómica e impede que os veículos como um carro se movam na perpendicular. Embora consigam atingir todas as configurações possíveis num espaço bidimensional, normalmente exigem manobras e planeamento complexo para atingirem a configuração pretendida.

As plataformas omnidirecionais não têm restrições não holonómicas, sendo que se podem movimentar em todas as direções com qualquer orientação. Esta capacidade é conhecida como mobilidade omnidirecional. Sendo assim a utilização de robôs omnidirecionais tem grandes vantagens em relação a plataformas convencionais, com tração diferencial ou com direção baseada em geometria de Ackermann, para se mover em áreas apertadas [2]. Estas plataformas são capazes de executar facilmente tarefas em ambientes estáticos, com obstáculos dinâmicos e corredores estreitos devido à sua versatilidade. Ambientes como estes são encontrados em escritórios, armazéns, hospitais, etc. pelo que o uso destas plataformas se adequa devido à sua flexibilidade.

A figura 1.3 apresenta uma plataforma omnidirecional desenvolvida pela KUKA para manobrar cargas volumosas e pesadas, superiores a noventa toneladas, e com precisão de localização de cerca de um milímetro.

Figura 1.3: KMP OMNIMOVE - Plataforma omnidirecional da KUKA

1.1.1 Mobilidade omnidirecional

Para construir uma plataforma com mobilidade omnidirecional é preciso ter em consideração as rodas utilizadas na construção da parte mecânica. As rodas que permitem obter este tipo de mobilidade podem ser divididas em duas categorias: rodas de desenho convencional e rodas de desenho especial.

As rodas de desenho convencional com capacidades omnidirecionais, ilustradas na Figura 1.4, utilizadas em robôs móveis são divididas em dois tipos: as rodas direcionais e os rodízios.

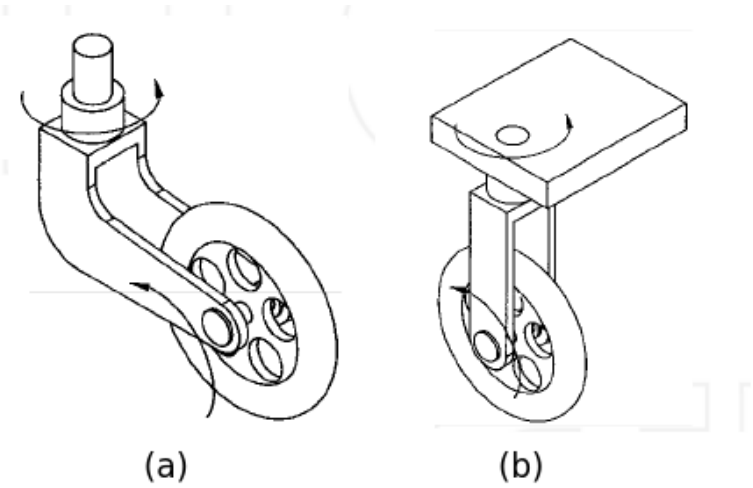


Figura 1.4: Rodas convencionais: (a) Rodízio, (b) Roda direcional

Comparando com as rodas de desenho especial, estas possuem uma maior capacidade de carga e uma maior tolerância a irregularidades no solo, no entanto, e devido à sua natureza não holonómica, não são consideradas verdadeiras rodas omnidirecionais, isto é, num seguimento de um caminho quando uma curva não contínua é encontrada, há uma quantidade finita de tempo em que a direção dos motores se reorienta para coincidir com a curva desejada. Assim sendo, estas rodas não podem ser consideradas verdadeiramente omnidirecionais [3]. Contudo, pela capacidade de seguir trajetórias de raio zero (rotação sobre si mesmo), o termo omnidirecional mantém-se aplicável a estas rodas.

São consideradas rodas de desenho especiais, aquelas que têm tração num sentido e permitem movimento passivo noutro. As rodas deste tipo, ilustradas na Figura 1.5, são divididas em rodas universais e rodas Mecanum.

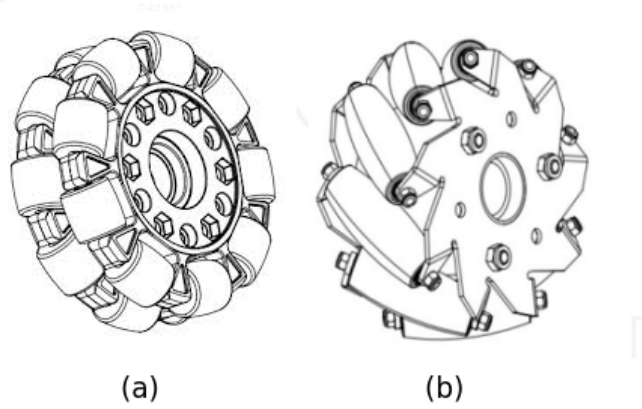


Figura 1.5: Rodas especiais: (a) Universal, (b) Mecanum

A roda universal é composta por pequenos rolos montados em torno do diâmetro exterior e perpendicularmente ao eixo de rotação da roda, permitindo assim a rotação normal da roda e também o movimento livre na direção paralela ao eixo da roda. A mobilidade omnidirecional obtém-se através da combinação de duas ou mais destas rodas numa plataforma móvel. A figura 1.6 representa as configurações mecânicas mais conhecidas com três e quatro rodas universais. No sistema com três rodas, estas encontram-se separadas por 120º graus e no sistema de quatro rodas por 90º graus.

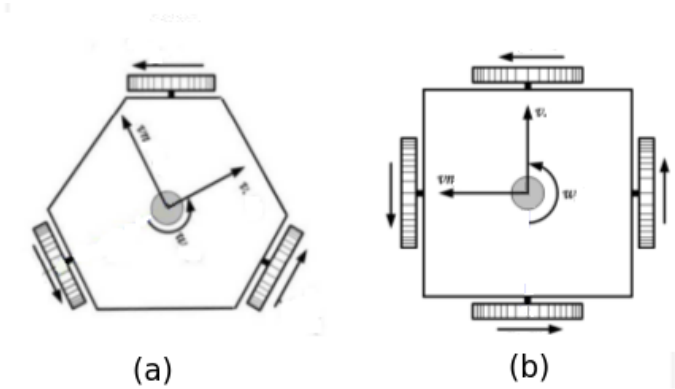


Figura 1.6: Configurações: (a) Robô com três rodas, (b) Robô com quatro rodas

As rodas mecanum possuem um desenho semelhante à roda universal, mas os rolos exteriores em vez de serem colocados perpendicularmente ao eixo da mesma são colocados com ângulos. A configuração da roda transmite uma parte da força na direção de rotação da roda e outra porção perpendicular ao eixo de rotação dos rolos. A mobilidade omnidirecional da plataforma obtém-se através da combinação de pelo menos quatro rodas mecanum, posicionadas de forma semelhante a um carro e orientadas estrategicamente para que a combinação de velocidade permitam a translação e rotação em qualquer direção. A Figura 1.7 representa a configuração mais comum utilizando rodas Mecanum.

A tabela 1.1, apresentada na obra de Doroftei *et al.*(2007) [4], faz uma comparação das características das rodas aqui apresentadas. Podemos destacar que as rodas convencionais exigem grande complexidade mecânica na montagem com motores que proporcionam o deslocamento ao contrário das rodas especiais, por outro lado, as rodas especiais são bastante sensíveis às irregularidades do piso podendo derrapar facilmente.

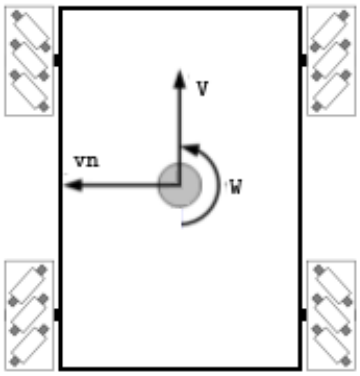


Figura 1.7: Configuração mecânica de robô com rodas Mecanum

Tabela 1.1: Tabela comparativa das características das rodas

Roda	**Caraterísticas positivas**	**Caraterísticas negativas**
Universal	Compacta e leve Conceção mecânica simples Maior disponibilidade comercial	Contacto descontínuo com o piso ou variação do raio da roda Sensibilidade às irregularidades do piso
Mecanum	Compacta Alta capacidade de carga	Contacto descontínuo com o piso Alta sensibilidade às irregularidades do piso Conceção complexa
Direcional	Contacto constante com o piso Alta capacidade de carga Robusta às condições do piso	Volumosa e pesada Alta força de fricção durante viragem Conceção mecânica complexa
Rodízio	Contacto constante com o piso Alta capacidade de carga Baixa força de fricção durante viragem Robusta às condições do piso	Volumosa Transmissão de sinal e alimentação através de juntas rotativas Conceção mecânica complexa

1.2 Enquadramento do projeto

Este trabalho está enquadrado num projeto do INESC TEC de desenvolvimento de uma plataforma móvel omnidirecional de baixo custo para cooperação com manipuladores industriais, onde o veículo guiado automaticamente (AGV) funcionará como um sistema de eixos adicionais que permite movimentar a peça a ser trabalhada pelo manipulador. Assim este trabalho contribuirá para o projeto com:

- Estudo da cinemática e modelo de odometria;
- Desenvolvimento de um controlador de caminhos para plataformas omnidirecionais e implementação em ROS;
- Estudo e validação da comunicação utilizando CANopen para o controlador dos motores;
- Desenvolvimento de um protótipo para testes.

1.3 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 5 capítulos.

No capítulo 2, é estudada a locomoção omnidirecional com rodas Mecanum e são apresentados trabalhos relacionados. Em seguida, no capítulo 3, é apresentado um controlador de caminhos omnidirecional e os seus resultados em simulação. No capítulo 4 foi abordada a utilização de CANOpen como protocolo comunicação com os controladores dos motores. Posteriormente, no capítulo 5, apresenta-se um protótipo de um robô omnidirecional e são demonstradas as suas capacidades. No capítulo 6, são apresentadas as conclusões e trabalhos futuros.

Capítulo 2

Locomoção omnidirecional

Neste capítulo pretende-se estudar a locomoção omnidirecional obtida através da utilização de rodas Mecanum. Estas rodas foram inventadas em 1973 pelo engenheiro Bengt Ilon [5], quando trabalhava na empresa sueca Mecanum AB, sendo por isso chamadas também de rodas suecas. A roda é composta por uma parte central que gira em torno de um eixo e um conjunto de rolos montados na periferia da mesma. Cada rolo possui uma superfície convexa na direção longitudinal e este é montado com o eixo posicionado de forma oblíqua em relação ao eixo de rotação da parte central. O espaçamento entre os rolos e o ângulo definido entre eixo longitudinal do rolo e o eixo de rotação é selecionado de modo a que o conjunto de rolos definam uma periferia circular ininterrupta vista de um ponto sobre a extensão do eixo de rotação. O ângulo entre o eixo dos rolos e o eixo central da roda pode ter qualquer valor, mas o convencional é de 45°.

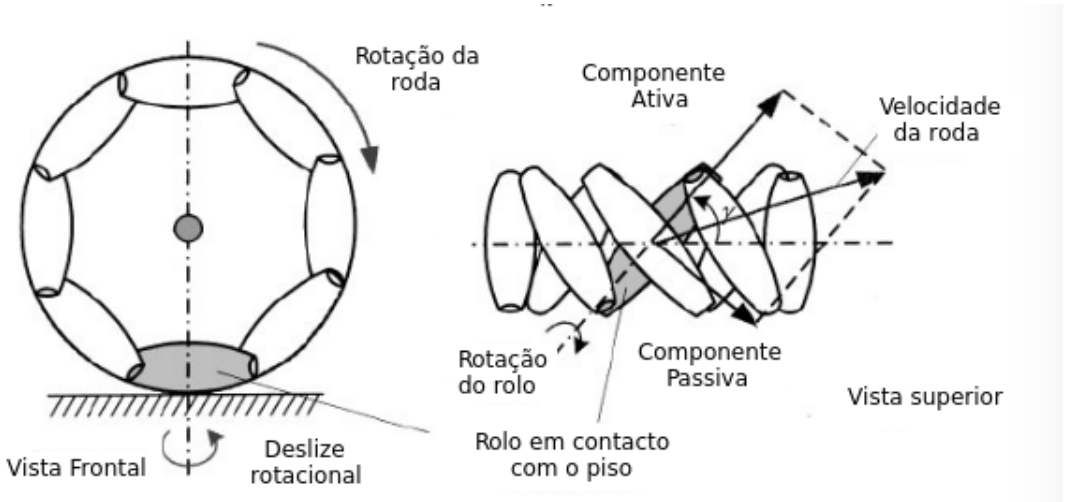


Figura 2.1: Graus de liberdade da roda Mecanum

Na figura 2.1 é possível visualizar os graus de liberdade associados ao movimento de rotação de uma roda Mecanum. Este tipo de roda possui 3 graus de liberdade: a rotação da roda, a rotação do rolo e o deslizamento vertical em torno do ponto de contacto. Nas rodas omnidirecionais, a

velocidade da roda pode ser divida em duas componentes, a componente ativa coincidente com o eixo do rolo em contacto com o solo e a componente passiva perpendicular ao eixo do rolo.

A principal vantagem deste projeto é que, embora a única rotação da roda seja alimentada ao longo do eixo principal, a roda pode cineticamente mover-se com muito pouco atrito ao longo muitas trajetórias possíveis, não apenas para a frente e para trás. [6]

A combinação de quatro rodas Mecanum proporciona um movimento omnidirecional para uma plataforma sem necessidade de um sistema de direção convencional.

O primeiro robô móvel com este tipo de rodas foi o Urano desenvolvido em Carnegie Mellon University por Patrick Muir e Charles Neuman no ano de 1987 [7]. Este robô não tinha um sistema de suspensão, o que é absolutamente necessário em pisos não planos. Muitos outros projetos com quatro rodas Mecanum foram apresentados por Diegel et al.(2002) [8]; Koestler e Braunl(2004) [9]; Siegwart e Nourbakash(2004) [6]; Doroftei, Grosu e Spinu(2004) [4]; etc.

Este tipo de configuração tem como principais desvantagens o facto do ponto de contacto com o piso se mover de forma paralela criando vibrações horizontais aquando do movimento lateral e ainda o facto da capacidade de superar obstáculos não ser independente da direção [4].

2.1 Cinemática

Na robótica, a cinemática desempenha um papel fundamental para definir posição, orientação, velocidade e aceleração dos robôs. A equação da cinemática direta permite-nos prever o movimento do robô sabendo a velocidade angular de cada roda. Por outro lado, a cinemática inversa possibilita o cálculo da velocidade de cada roda necessária para produzir a velocidade e rotação desejada para o robô. Este tópico já foi amplamente estudado e mais recentemente disponibilizado em publicações como a de Wakchaure *et al.* [10] ou de Taheri, Qiao e Ghaeminezhad, 2015 [11].

Na Figura 2.2 estão representados os vetores de velocidade e posição associados a uma configuração de quatro rodas.

Os parâmetros de configuração e velocidades são definidos da seguinte forma:

- $V_{iw}(i = 1,2,3,4) \in \mathbb{R}$ é o vetor de velocidade correspondente à rotação das rodas, onde: $V_{iw} = R_w * W_i$,
- R_w é o raio da roda, W_{iw} é a velocidade angular da roda;
- $V_{ir}(i = 1,2,3,4) \in \mathbb{R}$ é o vetor de velocidade perpendicular ao eixo de rotação do rolo em contacto com o piso;
- $V_{iX}, V_{iY}(i = 1,2,3,4) \in \mathbb{R}$ são os vetores de velocidade correspondentes ao referencial do robô;
- $V_X, V_Y \in \mathbb{R}$ corresponde à velocidade linear do robô;
- $W_Z \in \mathbb{R}$ corresponde à velocidade angular do robô;
- α corresponde ao ângulo entre o eixo da roda e o eixo do rolo;

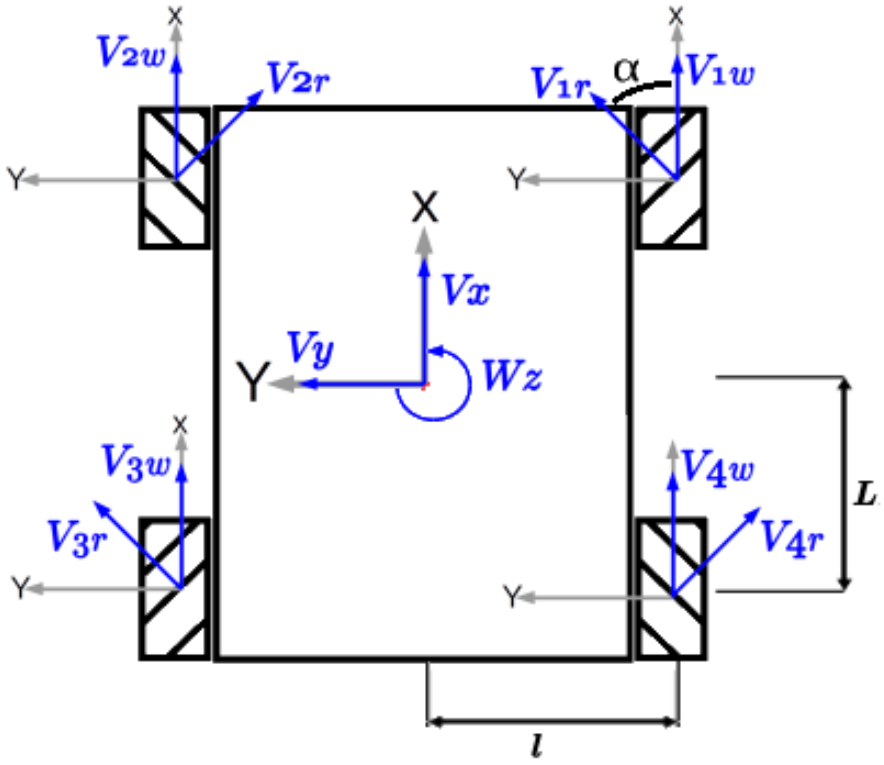


Figura 2.2: Configuração das rodas e velocidades do robô

- l, L corresponde à distância entre o centro do robô e o centro da roda, no eixo X e Y, respetivamente.

Quando as rodas Mecanum são atuadas através do acionamento dos motores, os rolos convertem uma parte da força na direção de rotação da roda para uma força normal à direção da roda. Dependendo da direção dos rolos e velocidade de cada roda, a combinação resultante de todas as forças produz um vetor de força total na direção desejada. A força motriz de cada uma das rodas pode ser decomposta em duas componentes de força, uma na direção do rolo e outra no sentido de rotação da roda.

Considerando $\alpha = 45°$, a velocidade individual da roda número um pode ser definida por :

$$V_{1X} = V_{1w} + (\frac{V_{1r}}{\sqrt{2}}), V_{1Y} = \frac{V_{1r}}{\sqrt{2}} \tag{2.1}$$

e ainda que, em função da velocidade do robô, temos:

$$V_{1X} = V_X + l * W_z, V_{1Y} = V_Y + L * W_z \tag{2.2}$$

Considerando a direção de cada roda e exprimindo as componentes V_{iX}, V_{iY} em função de V_X, V_Y e W_z, podemos obter as seguintes equações:

$$V_{1w} = V_X + V_Y + (l+L) * W_z \tag{2.3}$$
$$V_{2w} = V_X - V_Y - (l+L) * W_z \tag{2.4}$$
$$V_{3w} = V_X + V_Y - (l+L) * W_z \tag{2.5}$$
$$V_{4w} = V_X - V_Y + (l+L) * W_z \tag{2.6}$$

As equações anteriores podem ser escritas da seguinte forma:

$$\begin{bmatrix} V_{1w} \\ V_{2w} \\ V_{3w} \\ V_{4w} \end{bmatrix} = \begin{bmatrix} 1 & 1 & (l+L) \\ 1 & -1 & -(l+L) \\ 1 & 1 & -(l+L) \\ 1 & -1 & (l+L) \end{bmatrix} \begin{bmatrix} V_X \\ V_Y \\ W_z \end{bmatrix} \tag{2.7}$$

A equação 2.7 representa a cinemática inversa do robô apresentado na figura 2.2.

A cinemática direta representa um sistema de equações linear sobredeterminado (mais equações do que incógnitas). "Em termos físicos isto significa que se quatro velocidades rotacionais arbitrárias forem escolhidas para as rodas, não existe no geral um movimento do veículo que não envolva deslizamento de pelo menos uma roda no piso."Através do método dos mínimos quadrados podemos encontrar a solução que melhor se ajusta. A equação 2.8 representa a cinemática direta do robô.

$$\begin{bmatrix} V_X \\ V_Y \\ W_z \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4(l+L)} & -\frac{1}{4(l+L)} & -\frac{1}{4(l+L)} & \frac{1}{4(l+L)} \end{bmatrix} \begin{bmatrix} V_{1w} \\ V_{2w} \\ V_{3w} \\ V_{4w} \end{bmatrix} \tag{2.8}$$

2.2 Modelo de odometria

A odometria é uma das mais importantes formas de localização na robótica móvel. Esta tem como base a integração da informação incremental do movimento rotacional das rodas. Apesar de não possuir grande precisão a longo prazo devido à acumulação de erros, tanto de natureza aleatória como sistemática, esta permite a curto prazo obter uma boa estimativa do deslocamento do robô. Contudo, como é um método barato de implementar e com elevada frequência de amostragem, a sua conjugação com outros métodos de posicionamento absoluto contribui para uma estimativa de posição mais fidedigna.

A figura 2.3 ilustra a pose do robô, dada pela sua posição (x,y) e pela orientação (θ), em relação ao referencial cartesiano XY. $U_i(i = 1,2,3,4)$ representa o deslocamento horizontal de cada uma das rodas. As distâncias L e l representam a diferença entre o centro do robô e as rodas.

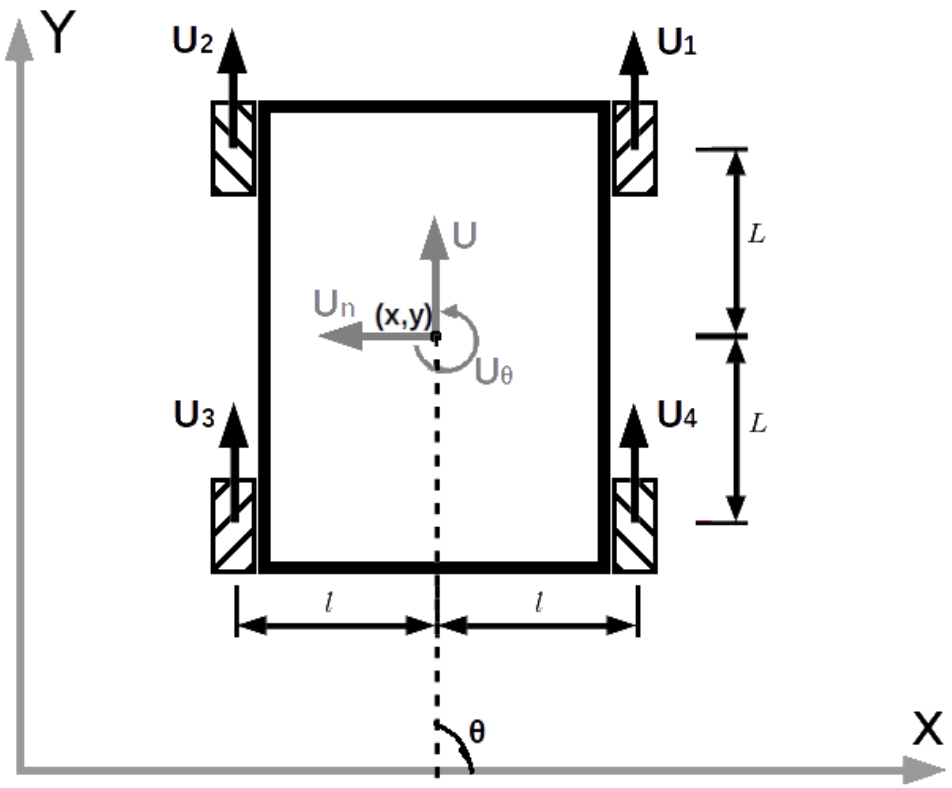


Figura 2.3: Deslocamento associado a um robô omnidirecional com quatro rodas Mecanum

A partir da informação periódica dos encoders ($N_i(n)(n \in \mathbb{R})$), correspondente aos impulsos do encoder durante um determinado período de amostragem (T), com o modelo de odometria é possível estimar a pose do robô. A relação entre $N_i(n)$ e o deslocamento efetuado por cada roda durante o período de amostragem, $\Delta U_i(n)$, é dada pela expressão:

$$\Delta U_i(n) = \frac{2\pi * R}{Imp_r} * N_i(n), \tag{2.9}$$

onde R corresponde ao raio da roda em metros e Imp_r corresponde ao número de impulsos por volta completa da roda. O valor de Imp_r é calculado tendo em conta a resolução do encoder e a relação da caixa redutora do motor.

Com base no modelo de cinemática direta, a variação dos deslocamentos é dada por:

$$\Delta U(n) = \frac{U_1 + U_2 + U_3 + U_4}{4} \quad (m) \tag{2.10}$$

$$\Delta U_n(n) = \frac{U_1 - U_2 + U_3 - U_4}{4} \quad (m) \tag{2.11}$$

$$\Delta U_\theta(n) = \frac{U_1 - U_2 - U_3 + U_4}{4 * (l + L)} \quad (rad), \tag{2.12}$$

onde $\Delta U(n)$ e $\Delta U_n(n)$ são dados em metros e $\Delta U_\theta(n)$ em radianos.

A pose do robô é atualizada da seguinte forma:

- Se $\Delta U_\theta(n) = 0$

$$x(n+1) = x(n) + \Delta U(n) * \cos(\theta(n)) - \Delta U_n(n) * \sin(\theta(n)) \quad (m) \tag{2.13}$$

$$y(n+1) = y(n) + \Delta U(n) * \sin(\theta(n)) + \Delta U_n(n) * \cos(\theta(n)) \quad (m) \tag{2.14}$$

$$\theta(n+1) = \theta(n) + \Delta U_\theta(n) \quad (rad) \tag{2.15}$$

- Se $\Delta U_\theta(n) \neq 0$

$$\begin{aligned} x(n+1) = x(n) + \frac{1}{\Delta U_{theta}(n)} * [\Delta U(n) * \sin(\Delta U_{theta}(n)) \\ + \Delta U_n(n) * (\cos(\Delta U_{theta}(n)) - 1)] * \cos(\theta(n) + \frac{\Delta U_{theta}(n)}{2}) \\ - \frac{1}{\Delta U_{theta}(n)} * [\Delta U(n) * (1 - \cos(\Delta U_{theta}(n))) \\ + \Delta U_n(n) * \sin(\Delta U_{theta}(n))] * \sin(\theta(n) + \frac{\Delta U_{theta}(n)}{2}) \quad (m) \end{aligned} \tag{2.16}$$

$$\begin{aligned} y(n+1) = y(n) + \frac{1}{\Delta U_{theta}(n)} * [\Delta U(n) * \sin(\Delta U_{theta}(n)) \\ + \Delta U_n(n) * (\cos(\Delta U_{theta}(n)) - 1)] * \sin(\theta(n) + \frac{\Delta U_{theta}(n)}{2}) \\ + \frac{1}{\Delta U_{theta}(n)} * [\Delta U(n) * (1 - \cos(\Delta U_{theta}(n))) \\ + \Delta U_n(n) * \sin(\Delta U_{theta}(n))] * \cos(\theta(n) + \frac{\Delta U_{theta}(n)}{2}) \quad (m) \end{aligned} \tag{2.17}$$

$$\theta(n+1) = \theta(n) + \Delta U_\theta(n) \quad (rad) \tag{2.18}$$

Capítulo 3

Seguimento de Caminhos

Neste capítulo é ilustrada a implementação do seguidor de caminhos omnidirecional.

O seguidor de caminhos, trata-se de um controlador que tem como objetivo fazer com que o robô siga uma determinado percurso minimizando o erro entre o percurso efetuado pelo robô e o percurso de referência previamente definido.

O controlador desenvolvido faz parte de um controlo em cascata, tendo como entrada uma "definição de caminho"e como saída a velocidade para o robô. Neste caso, pretendemos desenvolver um controlador para um robô omnidirecional pelo que não existem restrições holonómicas, isto é, podemos nos deslocar em qualquer direção independentemente da orientação do robô. Pretende-se que este controlador possa ser implementado em qualquer robô omnidirecional pelo que terá como saídas a velocidade (V), a velocidade normal (V_n) e a velocidade angular (ω) do robô como representadas na figura 1.7.

3.1 Definição de Caminho

É muito comum ouvirmos falar em controlo e geração de trajetória, pelo que é importante perceber a diferença entre um caminho e uma trajetória. Quando falamos em caminho referimos a um determinado percurso sem dependência temporal, isto é, não existe um momento no espaço temporal predefinido para o robô se encontrar num determinado ponto do percurso. Por outro lado, quando falamos em trajetória estamos a definir uma dependência temporal associada a todo o percurso efetuado pelo robô.

Apenas se pretende analisar o erro no seguimento do percurso pelo que não existem restrições temporais. Contudo, não pretendemos que o robô permaneça imóvel tentando minimizar o erro num determinado local do percurso, pelo que também foi definida uma velocidade média (vel) com que o robô deve efetuar o percurso.

O robô irá movimentar-se num espaço bidimensional pelo que a sua posição é dada pelas coordenadas x, y e a sua orientação pelo ângulo θ dadas em função de um referencial global externo ao robô e fixo no espaço, como definido na figura 3.1.

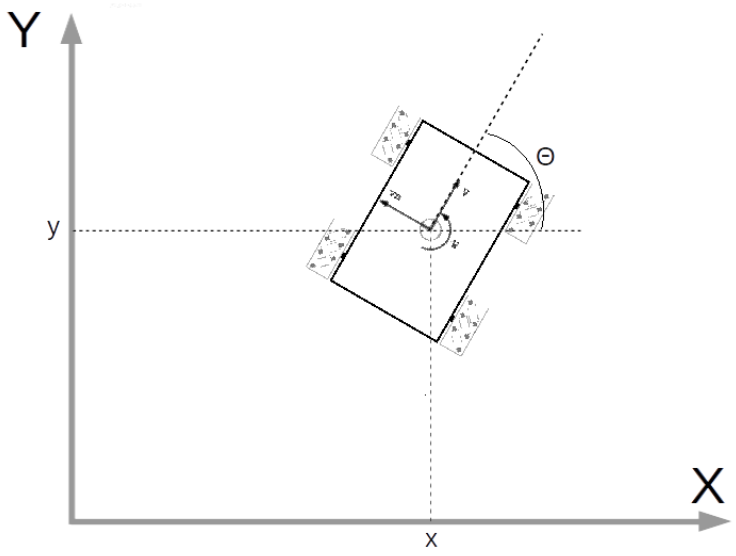


Figura 3.1: Definição da posição do robô

Para o controlador implementado foram definidos dois tipos de caminhos que este deve ser capaz de percorrer, segmento de reta e arco de circunferência.

A forma mais fácil de definir um segmento de reta é através dos pontos inicial e final. As propriedades deste caminho incluem também a velocidade com que o robô deve efetuar o percurso e a velocidade final. Será ainda definida uma orientação segundo a qual o robô deverá terminar o percurso. A tabela 3.1 indica todos os elementos necessários para definir um caminho do tipo segmento de reta.

Tabela 3.1: Elementos de caminho do tipo segmento de reta

Elemento	**Descrição**
(x_i, y_i)	Ponto inicial do segmento de reta (posição inicial do robô)
(x_f, y_f)	Ponto final do segmento
(θ_f)	Orientação pretendida (orientação final do robô)
vel	Velocidade para o percurso
vel_f	Velocidade final (velocidade no final do robô)

A forma mais simples de definirmos um arco de circunferência é através do seu centro, do raio do arco e do ângulo inicial e final do arco. Na figura 3.2 é possível observar os elementos que de definem um arco.

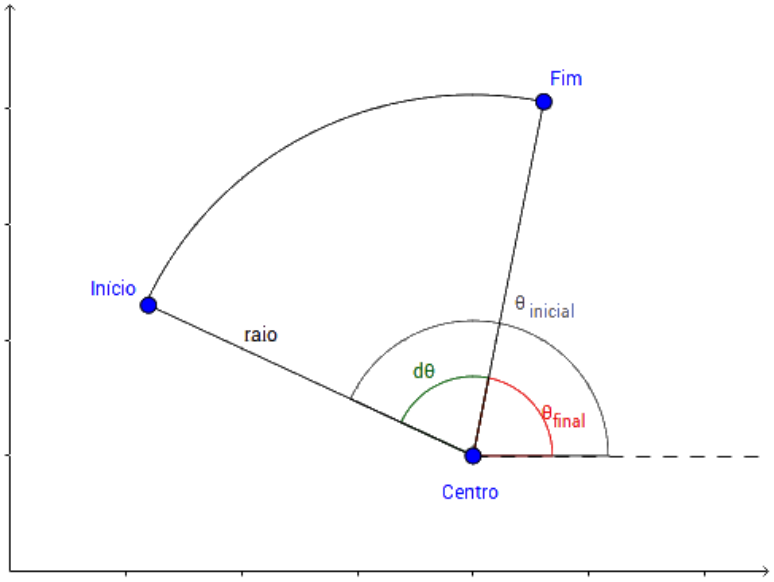


Figura 3.2: Arco de circunferência

O ponto inicial do arco deverá ser sempre a posição inicial do robô pelo que ao definirmos um centro poderíamos estar a descrever um raio errado do pretendido, pelo que optamos por definir o arco pelo seu ponto inicial, o raio e os ângulos iniciais e finais do arco. Outros elementos deste tipo de caminho são a orientação inicial e final do robô.

A tabela 3.2 indica todo os elementos necessários para definir um caminho do tipo arco de circunferência.

Tabela 3.2: Elementos de caminho do tipo arco de circunferência

Elemento	Descrição
(x_i, y_i)	Ponto inicial do arco de circunferência (posição inicial do robô)
(θ_i)	Ângulo inicial do robô
(θ_f)	Ângulo final do robô
$(r\theta_i)$	Ângulo inicial do arco
$(r\theta_f)$	Ângulo final do arco
r	Raio da circunferência
vel	Velocidade para o percurso

3.2 Controlador

O controlador pretende minimizar o erro entre o caminho de referência e o caminho real percorrido pelo robô. O controlador tem como entradas a definição de um caminho e a pose atual do robô e calcula as velocidades para serem aplicadas no robô. A figura 3.3 representa as entradas e saídas do controlador.

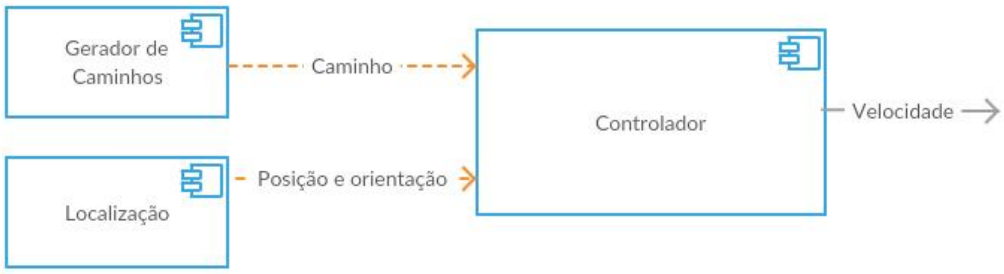


Figura 3.3: Entradas e Saídas do controlador

3.2.1 Segmento de reta

Segmento de reta é definido por dois pontos, início e fim tal como na figura 3.4. Assume-se que o robô parte da posição inicial e se desvia do percurso definido pelo segmento. O vetor $\overrightarrow{IF}$ é o vetor entre a posição inicial e a posição final desejada. O vetor $\overrightarrow{RF}$ é o vetor entre a posição atual do robô e a posição final desejada.

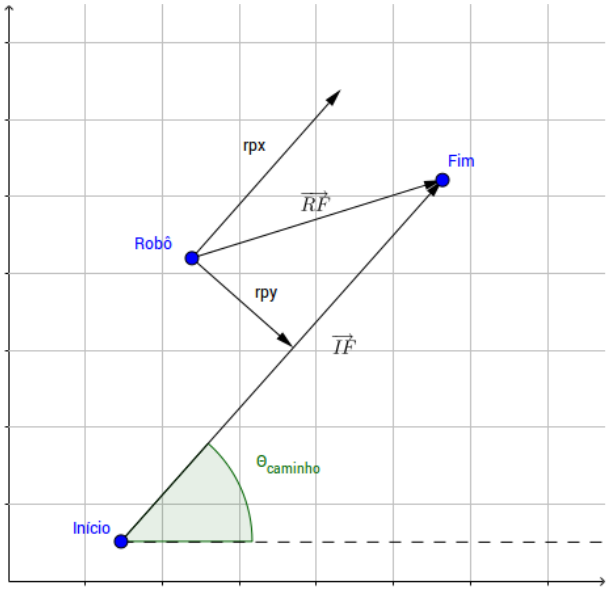


Figura 3.4: Erros associados ao segmento de reta

O vetor $\overrightarrow{RF}$ pode ser decomposto nas componentes rpx, rpy, onde rpy é a distância ao ponto mais próximo do percurso desejado e rpx corresponde à distância entre o ponto mas próximo do percurso e o ponto final.

A estratégia de controlo adotada passa por o robô manter a velocidade pretendida na direção do vetor $\overrightarrow{IF}$ e corrigir na direção paralela (direção do rpy) conforme o valor do erro. O valor do erro é dado pela distância ao ponto mais perto do segmento de reta que se pretende seguir, ou seja, valor de rpy.

O robô omnidirecional pode movimentar-se em qualquer direção independentemente da sua orientação, desta forma, o controlo da orientação do robô, ao longo do percurso, é realizado recorrendo ao valor do erro entre a orientação desejada e a atual do robô ($erro_\theta$).

O diagrama da figura 3.5 representa a execução do controlador. As etapas que o constituem são as seguintes:

- **Cálculo do percurso -** Com base na posição inicial (que deve coincidir com posição do robô) e na posição final é definido o percurso a ser efetuado. Este percurso é definido com base no ângulo do vetor $\overrightarrow{IF}$ faz com o eixo do X ($\theta_{caminho}$).

- **Obter estado do robô -** É lida a posição e orientação atual do robô.

- **Cálculo dos erros -** Com base na posição atual é calculado o valor dos erros rpx, rpy e $erro_\theta$.

- **Cálculo da Velocidade -** Assumindo que o robô se encontra com a orientação do vetor $\overrightarrow{IF}$ são calculadas as velocidades (no referencial do robô) V_x, V_y e ω .

  A velocidade V_x corresponde à velocidade na direção do ponto final assim sendo a velocidade aplicada será correspondente à componente *vel* definida pelo caminho. Esta velocidade é diferente na aproximação do ponto final. Quando o robô se encontra a uma distância inferior a um determinado valor é definida uma rampa que este deve seguir de modo a atingir a velocidade vel_f definida pelo caminho.

  A velocidade V_y é proporcional ao erro rpy.

  A velocidade ω é proporcional ao erro $erro_\theta$.

- **Ajuste de velocidade -** As velocidades V_x e V_y são ajustadas para a orientação atual do robô resultando nas velocidades V e V_n a ser aplicadas ao robô.

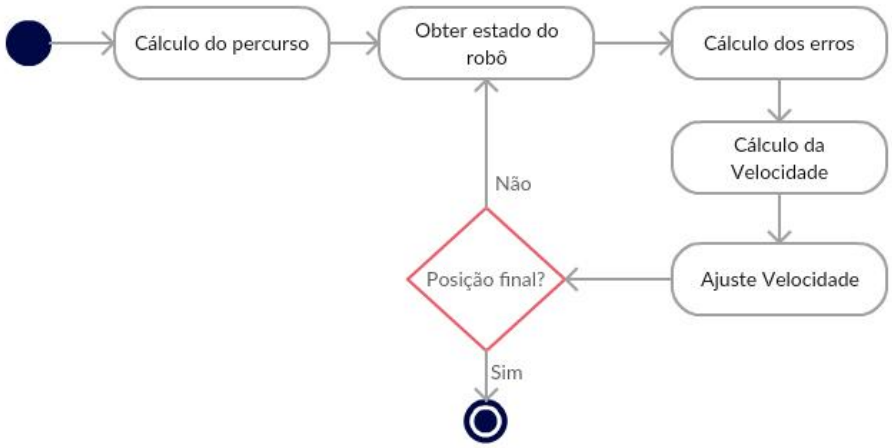


Figura 3.5: Diagrama de Atividade do controlador segmento de reta e arco de circunferência

O controlador executa como no algoritmo 1, onde Ky, Kw são ganhos possíveis de configurar e $e1, e2$ correspondem aos erros a partir dos quais o controlador considera terminado o caminho, sendo $e1$ o erro de distância ao ponto final e $e2$ o erro da orientação.

Algoritmo 1: Controlador de caminhos para segmentos de reta

Dados: $x_i, y_i, x_f, y_f, \theta_f, vel, vel_f$

início

    // Cálculo do percurso
    $\theta_{caminho}$ = arctan2($y_f - y_i, x_f - x_i$);
    repita
        // Obter estado do robô
        $[x, y, \theta]$ = PoseRobot;
        // Cálculo dos erros
        $[rpx, rpy]$ = VectorTranslateAndRotate$(x, y, -x_f, -y_f, -\theta_{caminho})$;
        $rp\theta$ = DiffAngle$(\theta, \theta_{caminho})$;
        $erro_\theta$ = DiffAngle(θ_f, θ);
        d = Dist(rpx, rpy);
        // Cálculo da velocidade
        se $vel_f > vel$ **então**
            V_x = **Máximo entre** vel **ou** $((vel - vel_f)/0.1) *$ Módulo$(rpx) + vel_f$;
        senão
            V_x = **Mínimo entre** vel **ou** $((vel - vel_f)/0.1) *$ Módulo$(rpx) + vel_f$;
        fim
        $V_x = -V_x *$ Sign(rpx);
        $V_y = -Ky * rpy$;
        $\omega = Kw * erro_\theta$;
        // Ajuste Velocidade
        $[V, Vn]$ = VectorTranslateAndRotate$(V_x, V_y, 0, 0, -rp_\theta)$;
    até $d < e1$ *e* $error_\theta < e2$;

fim

Algoritmo 2: Função Dist

Função *Dist* **é**

    Resultado: Retorna a distância ao ponto (x,y)
    Dados: x, y
    $aux = \sqrt{x^2 + y^2}$;
    retorna *aux*

fim

Algoritmo 3: Função VectorTranslateAndRotate

Função *VectorTranslateAndRotate* **é**
  Resultado: Retorna o resultado a adição dos vetores (x1,y1) e (x2,y2) seguido de uma rotação segundo o ângulo θ
  Dados: $(x1, y1, x2, y2, \theta)$
  $x3 = x1 + x2$;
  $y3 = y1 + y2$;
  $x = x3 * \cos(\theta) - y3 * \sin(\theta)$;
  $y = x3 * \sin(\theta) + y3 * \cos(\theta)$;
  retorna *[x,y]*
fim

Algoritmo 4: Função DiffAngle

Função *DiffAngle* **é**
  Resultado: Retorna a valor da diferença de dois ângulos normalizado
  Dados: (θ_1, θ_2)
  $\arctan 2(\sin(\theta_1 - \theta_2), \cos(\theta_1 - \theta_2))$;
  retorna θ
fim

Algoritmo 5: Função Sign

Função *Sign* **é**
  Resultado: Retorna o sinal de x
  Dados: x
  se $x > 0$ **então**
    $aux = 1$;
  senão se $x < 0$ **então**
    $aux = -1$;
  senão
    $aux = 0$;
  fim
  retorna aux
fim

3.2.2 Arco de circunferência

O caminho do tipo arco de circunferência é definido através do ponto inicial e dos ângulos inicial e final. É através destes parâmetros que podemos calcular o centro e o ponto final do arco.

Na figura 3.6 está representada a situação em que o robô se afasta do caminho pretendido. O vetor *rpy* é o vetor entre a posição atual do robô e o ponto mais próximo do arco. O valor de *rpy* é calculado pela diferença entre o raio do arco e a distância do centro à posição do robô.

O vetor rpx é paralelo ao vetor rpy e consequentemente é paralelo à tangente do arco no ponto mais próximo do robô.

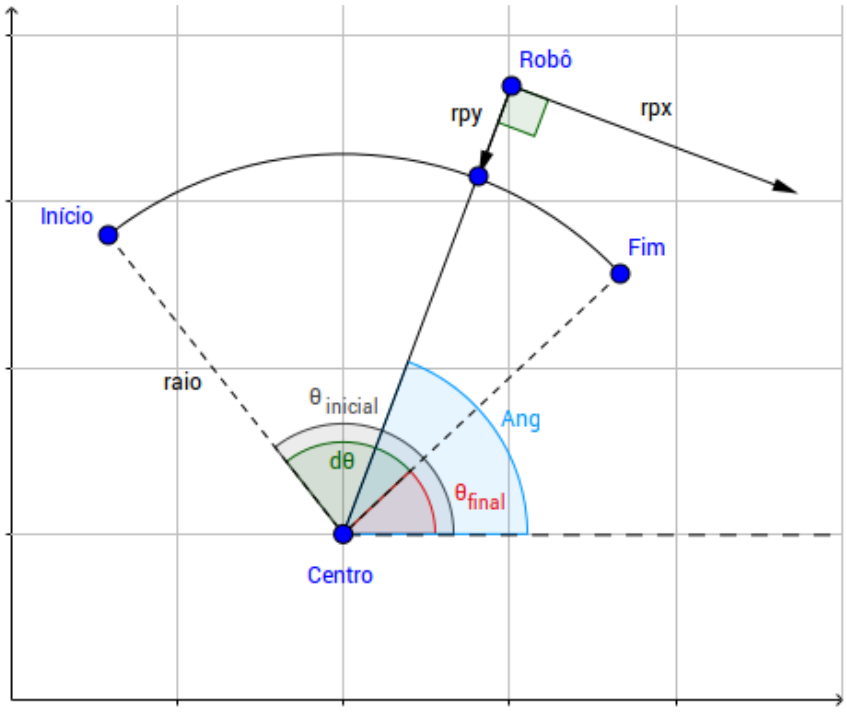


Figura 3.6: Erros associados ao arco de circunferência

A estratégia de controlo adotada passa por minimizar o valor de rpy que representa o erro entre o robô e o ponto do arco mais próximo e ao mesmo tempo avançar ao longo do arco, mantendo a velocidade pretendida (vel) na direção de rpx.

Este caminho é também definido pelo orientação inicial e final pretendida do robô. A rotação de (θ_i) para (θ_f) é efetuada de forma gradual e ao longo do percurso.

O controlador apresenta um funcionamento similar ao controlo da reta representado no diagrama da figura 3.5. As etapas deste controlador são:

- **Cálculo do percurso -** Com base na posição inicial e nos ângulos do arco são calculados os pontos central e final do arco. É ainda calculada a amplitude do arco a percorrer.
- **Obter estado do robô -** É lida a posição e orientação atual do robô.
- **Cálculo dos erros -** Com base na posição atual é calculado o valor do erro rpy. É calculada ainda a orientação desejada conforme a percentagem do arco já percorrida. O erro $erro_\theta$ é a diferença entre a orientação desejada e a atual do robô.
- **Cálculo da Velocidade -** Assumindo que o robô se encontra com a orientação do vetor rpx são calculadas as velocidades (no referencial do robô) V_x, V_y e W.

A velocidade V_x aplicada será correspondente à componente *vel* definida pelo caminho.

A velocidade V_y é proporcional ao erro rpy.

A velocidade ω é proporcional ao erro $erro_\theta$.

- **Ajuste de velocidade -** As velocidades V_x e V_y são ajustadas para a orientação atual do robô resultando nas velocidades V e V_n a ser aplicadas ao robô.

O algoritmo 6 apresenta o controlador de caminhos para arco de circunferência, onde Ky e Kw são ganhos configuráveis.

Algoritmo 6: Controlador de caminhos para arco de circunferência

Dados: $x_i, y_i, \theta_i, \theta_f, r\theta_i, r\theta_f, r, vel, vel_f$

início

`// Cálculo do percurso`

$x_c = x_i - \cos(r\theta_i) * r$;

$y_c = y_i - \sin(r\theta_i) * r$;

$path_r\theta$ = DiffAngle$(r\theta_f, r\theta_i)$;

$path_\theta$ = DiffAngle(θ_f, θ_i);

repita

`// Obter estado do robô`

$[x, y, \theta]$ = PoseRobot;

`// Cálculo dos erros`

$ang = \arctan 2(y - y_c, x - x_c)$;

rpy = Sign$(path_r\theta)$*$(r$−Dist$(y - y_c, x - x_c)))$;

$completed$ = DiffAngle$(ang, r\theta_i)/path_r\theta$;

$wanted_\theta$ = DiffAngle$(\theta_i + path_\theta * completed, 0)$;

$rp\theta$ = DiffAngle$(\theta$,DiffAngle$((ang + (\pi/2)) *$Sign$(path_r\theta)))$;

$erro_\theta$ = DiffAngle$(\theta, wanted_\theta)$;

`// Cálculo da velocidade`

$V_x = vel$;

$V_y = -Ky * rpy$;

$\omega = -Kw * erro_\theta$;

`// Ajuste Velocidade`

$[V, Vn]$ = VectorTranslateAndRotate$(V_x, V_y, 0, 0, -rp_\theta)$;

até $completed >= 1$;

fim

3.3 Implementação em ROS

A implementação em ROS passou pela criação de uma classe em C++, TPath_Calculations, e a sua posterior integração no nó Path Controller. A classe é utilizada para guardar a informação

do caminho que está a ser seguido naquele momento, assim existe uma função que permite definir o caminho a ser seguido, e para calcular a velocidade a aplicar ao robô conforme a pose do robô.

A função que permite definir o caminho é executada sempre que se pretende configurar um novo caminho e corresponde à etapa Cálculo do percurso. Existe uma outra função que executa as restantes etapas e retorna um booleano indicando se o caminho está ao não completo.

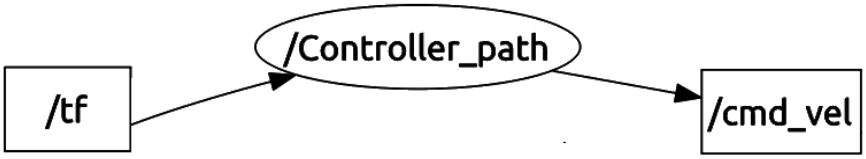


Figura 3.7: Interface do nó Path Controller (/Controller_path)

O nó Path Controller define o caminho que o controlador deve seguir e com um determinado período executa o cálculo das novas velocidades e atualiza as velocidades para o robô. As velocidades para o robô são publicadas no tópico "/cmd_vel"do tipo "geometry_msgs/Twist". A pose do robô é obtida através da subscrição do tópico "tf". Este nó permite a configuração de três parâmetros, o período com que é executado o controlador, o referencial base e o referencial referente à pose do robô.

3.4 Teste do controlador

Os testes ao controlador implementado foram efetuados com recurso ao simulador Stage, uma ferramenta disponibilizada para ROS. Este simulador simula a cinemática (neste caso omnidirecional) do robô, as colisões com os obstáculos e ainda publica através de tópicos a odometria e os dados do laser. A figura 3.8 é uma imagem do simulador onde a preto estão representados os obstáculos do mapa e a verde o robô que estamos a simular, neste caso o Discovery Q2 apresentado no capítulo (5).

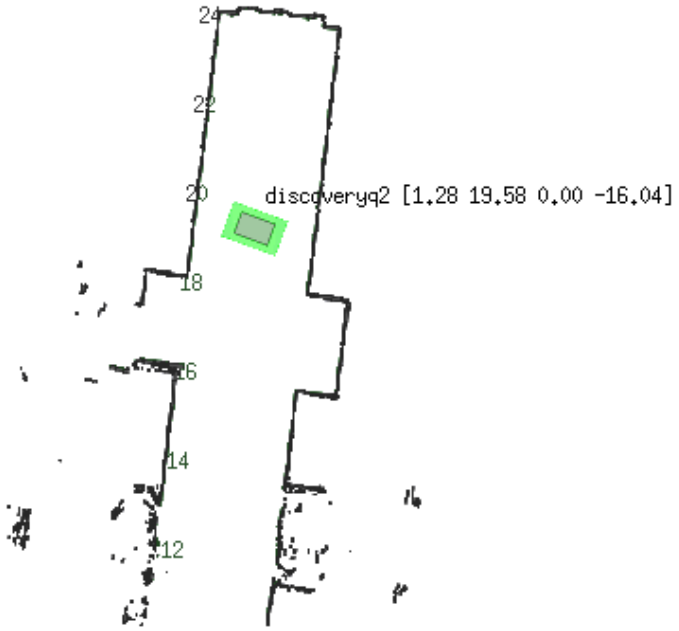


Figura 3.8: Simulador Stage com robô discoveryq2

Na figura 3.9 encontram-se representados os nós e tópicos envolvidos na simulação. O nó "/discoveryq2_simulator"é o simulador (do tipo "stageros") que subscreve o tópico "/cmd_vel", com a velocidade da o robô e publica para o tópico "/tf"a pose do robô. O nó "/map_sever"fornece o mapa ao simulador. O *script* de arranque do simulador encontra-se no anexo A.5.

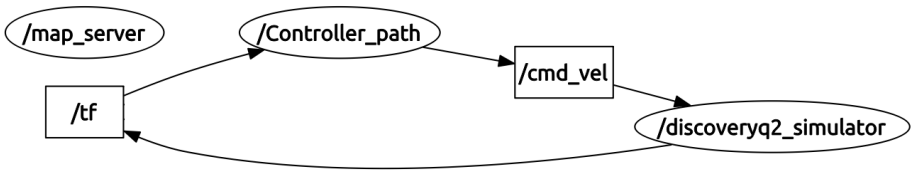


Figura 3.9: Nós e tópicos da simulação

3.4.1 Resultados do controlador para segmentos de reta

A figura 3.10 representa a simulação do seguimento de um caminho do tipo segmento de reta. A reta do caminho efetuada pelo robô omite a reta pretendida. A figura 3.11 representa o erro rpy,

ou seja a distância entre o robô e o ponto mais próximo do segmento de reta que se pretende seguir. O valor máximo deste erro nesta simulação é praticamente nulo. O erro entre a orientação do robô e a orientação pretendida é visto na figura 3.12 e nesta simulação o valor é aproximadamente zero.

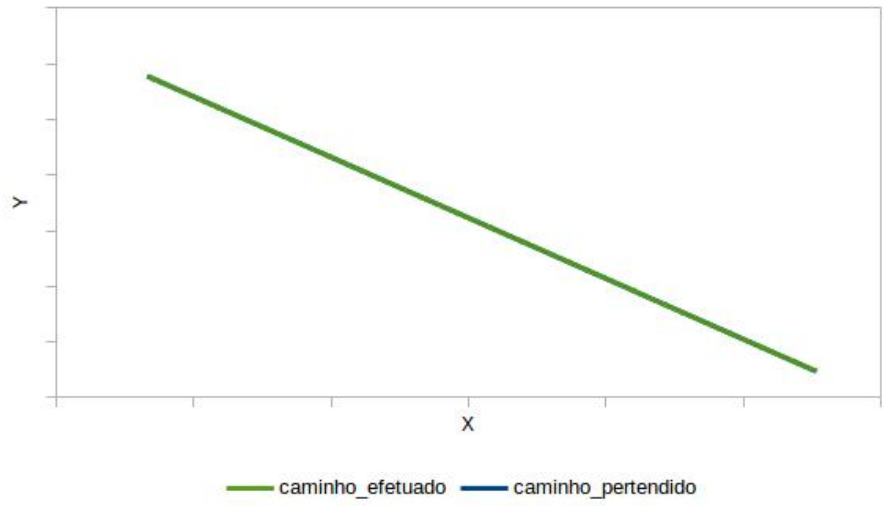


Figura 3.10: Simulação 1 - posição x,y

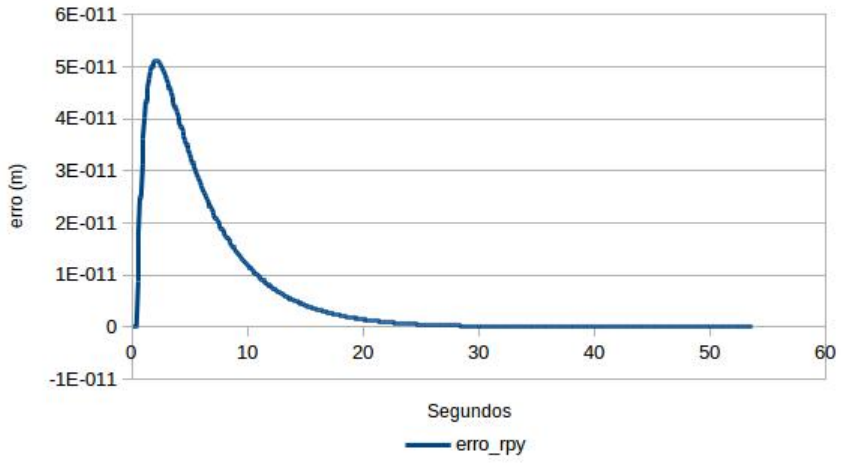


Figura 3.11: Simulação 1 - erro rpy

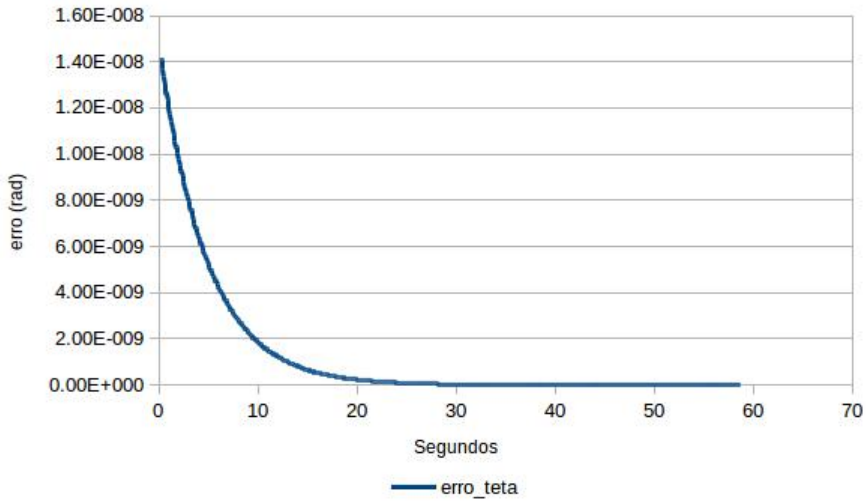


Figura 3.12: Simulação 1 - $erro_\theta$

Na figura 3.13 está representada uma segunda simulação, onde por dois momentos o robô foi alterado da sua posição para uma posição longe do segmento de reta. É possível observar que o robô converge para o caminho pretendido. A figura 3.14 representa o erro ao ponto mais próximo da reta ao longo do tempo. É possível observar que ao longo do tempo, nos momentos em que a posição foi alterada, este volta a convergir para o caminho pretendido, ou seja o erro converge para zero.

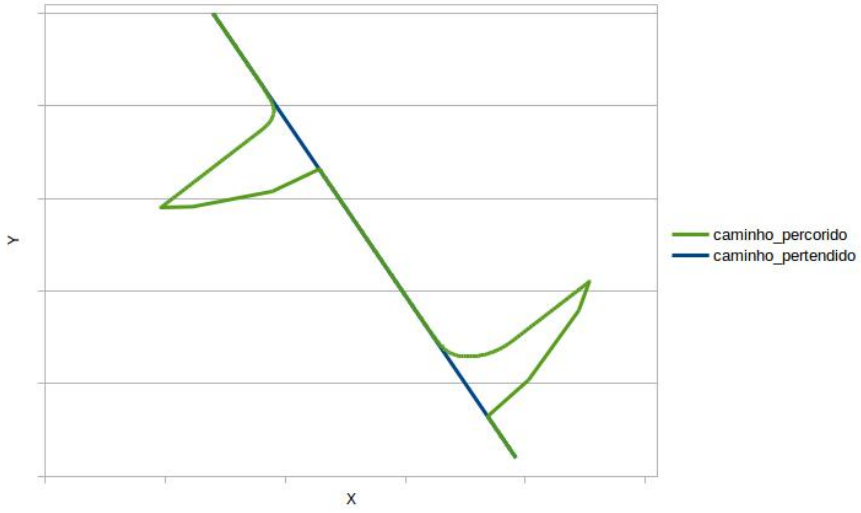


Figura 3.13: Simulação 2 - posição x,y

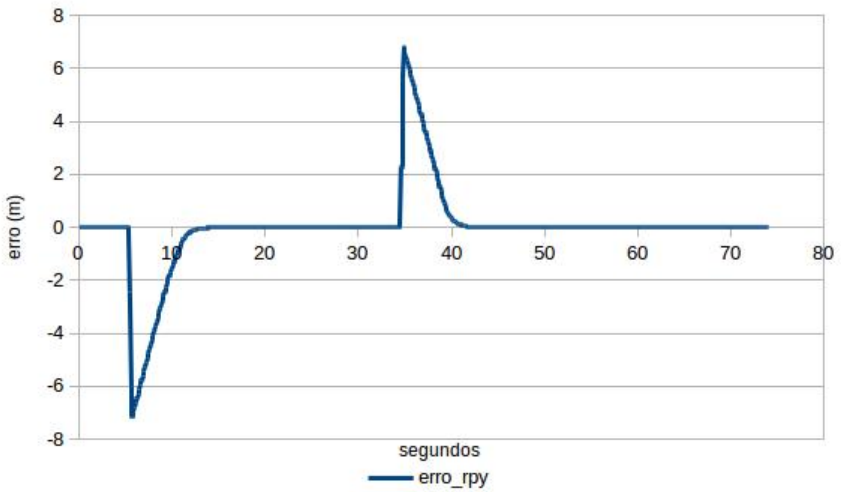


Figura 3.14: Simulação 2 - erro rpy

A figura 3.15 representa uma terceira simulação onde foi testado o erro entre a orientação final pretendida e a orientação do robô. É possível observar que este erro converge para zero ao longo do percurso.

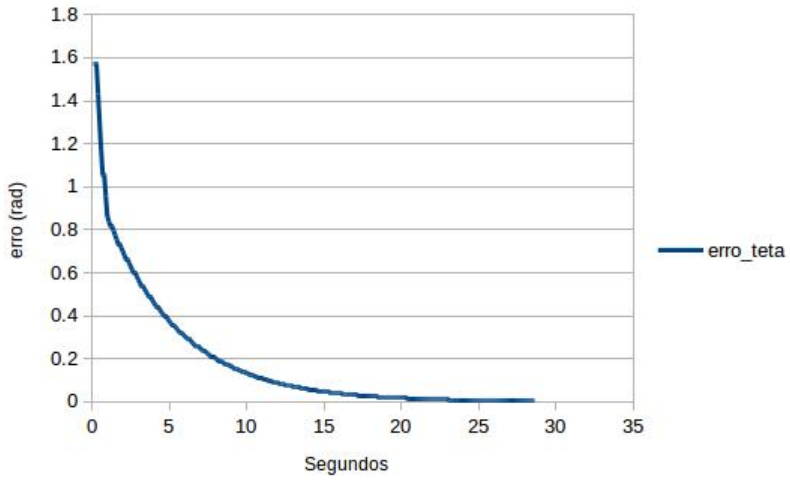


Figura 3.15: Simulação 3 - $erro_\theta$

3.4.2 Resultados do controlador para arco de circunferência

A figura 3.16 representa a simulação do seguimento de um caminho do tipo arco de circunferência. O arco pretendido é omitido pela caminho efetuada pelo robô. A figura 3.17 representa o erro rpy, ou seja a distância entre o robô e o ponto mais próximo do arco de circunferência que se pretende seguir. O valor deste erro nesta simulação cresce inicialmente e mantêm-se constante

ao longo do tempo devido ao movimento circular que o robô descreve, no entanto, obtêm-se um valor aceitável.

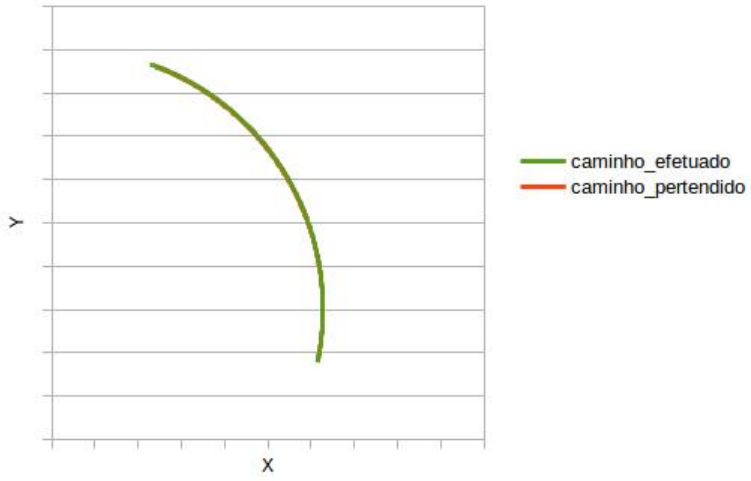


Figura 3.16: Simulação 4 - posição x,y

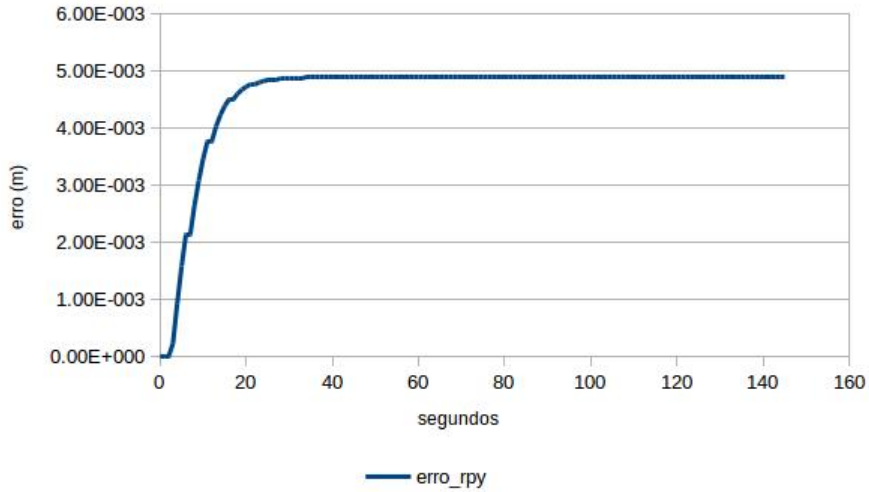


Figura 3.17: Simulação 4 - erro rpy

Na figura 3.18 está representada uma segunda simulação, onde por dois momentos o robô foi alterado da sua posição para uma posição longe do arco de circunferência. É possível observar que o robô converge para o caminho pretendido. A figura 3.19 representa o erro ao ponto mais próximo da reta ao longo do tempo. É possível observar que ao longo do tempo, nos momentos em que a posição foi alterada, este volta a convergir para o caminho pretendido, ou seja o erro converge para zero. O erro entre a orientação do robô e a orientação pretendida, apresentado na

figura 3.20, é afetado sempre que o robô foi alterado de posição, no entanto, o seu valor máximo nesta simulação é quase zero.

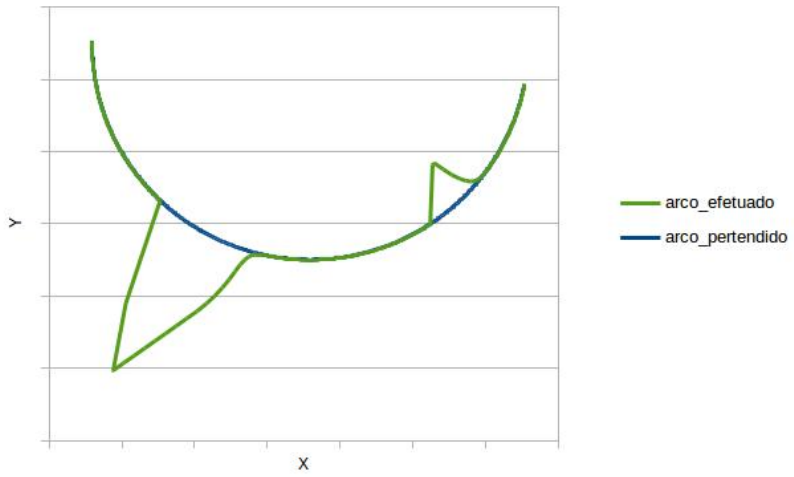


Figura 3.18: Simulação 5 - posição x,y

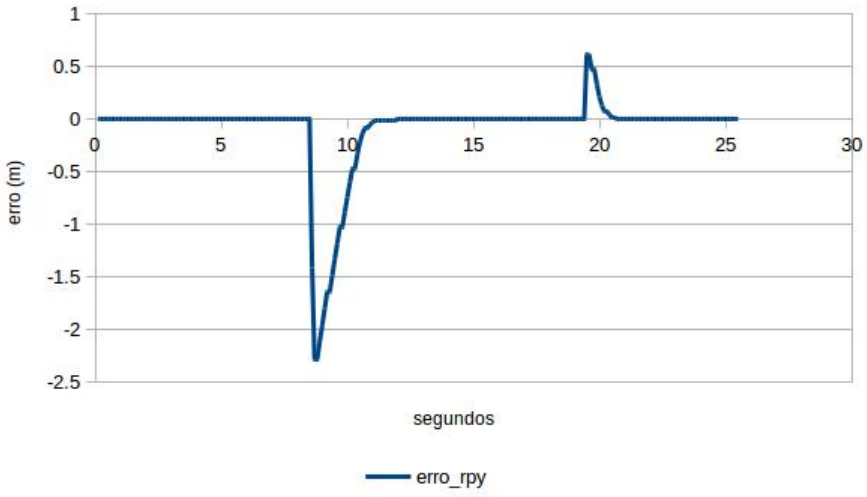


Figura 3.19: Simulação 5 - erro rpy

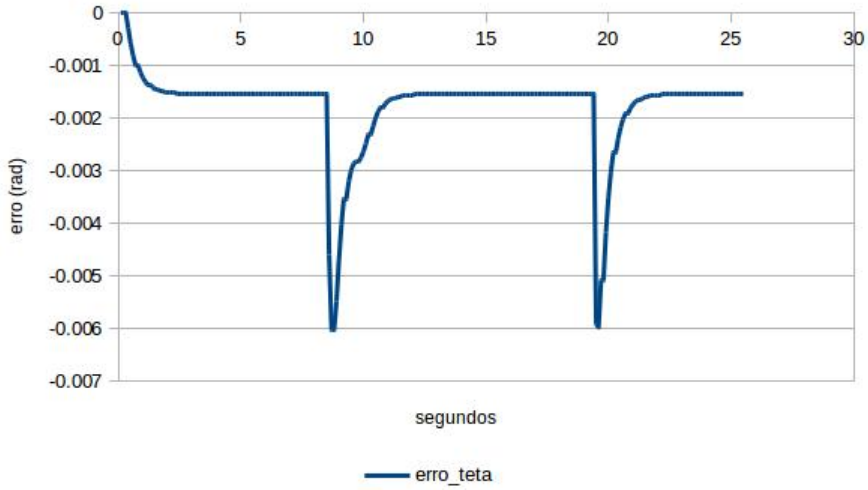


Figura 3.20: Simulação 5 - $erro_\theta$

3.4.3 Conclusões

O teste em simulador deste controlador permitiu concluir que o controlador consegue minimizar o erro em relação ao caminho de referência com um erro aceitável. É possível verificar que existem alguns erros que se mantém constantes e não diminuem como por exemplo o *errorpy* e $erro_\theta$ no seguimento arco, isto acontece devido ao movimento de rotação sobre si mesmo que o robô possui e também devido ao arco que é descrito. É importante referir que o simular apenas nos permitiu validar o funcionamento do controlador, sendo que o facto do simulador não possuir dinâmica faz com que os erros apresentados não sejam os esperados em ambiente real.

Capítulo 4

Protótipo Industrial

O protótipo industrial de uma plataforma omnidirecional com rodas Mecanum está a ser desenvolvido pelo INESC TEC. A plataforma tem dimensões 1200x785mm e com suspensão coaxial. As figuras 4.1 e 4.2 representam o protótipo do Veículo Guiado Automaticamente (AGV) que se encontra em desenvolvimento.

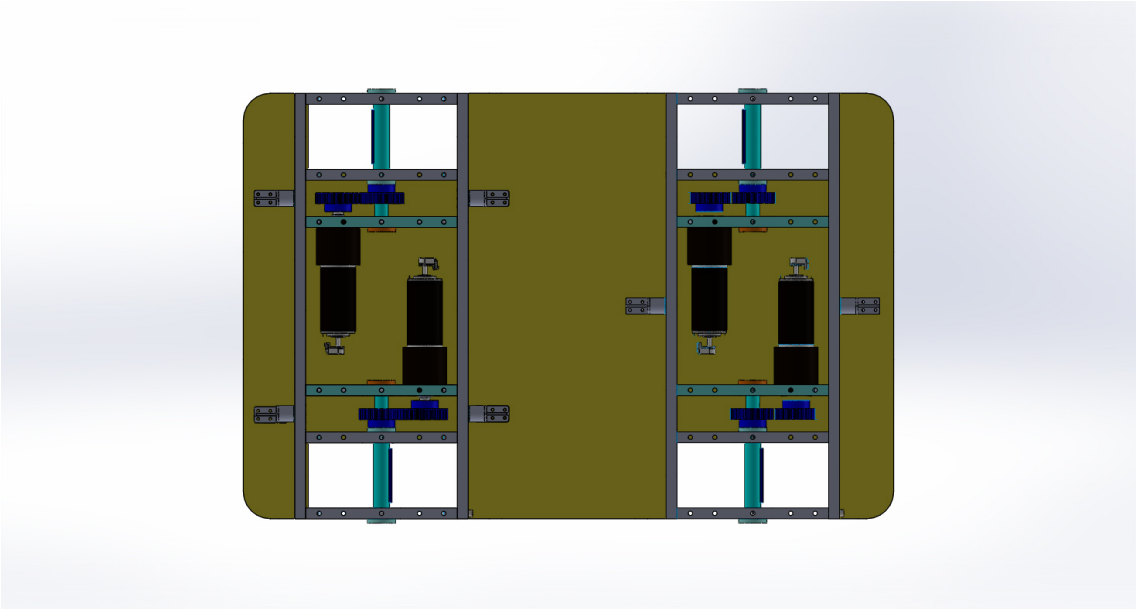


Figura 4.1: Estrutura do AVG sem rodas

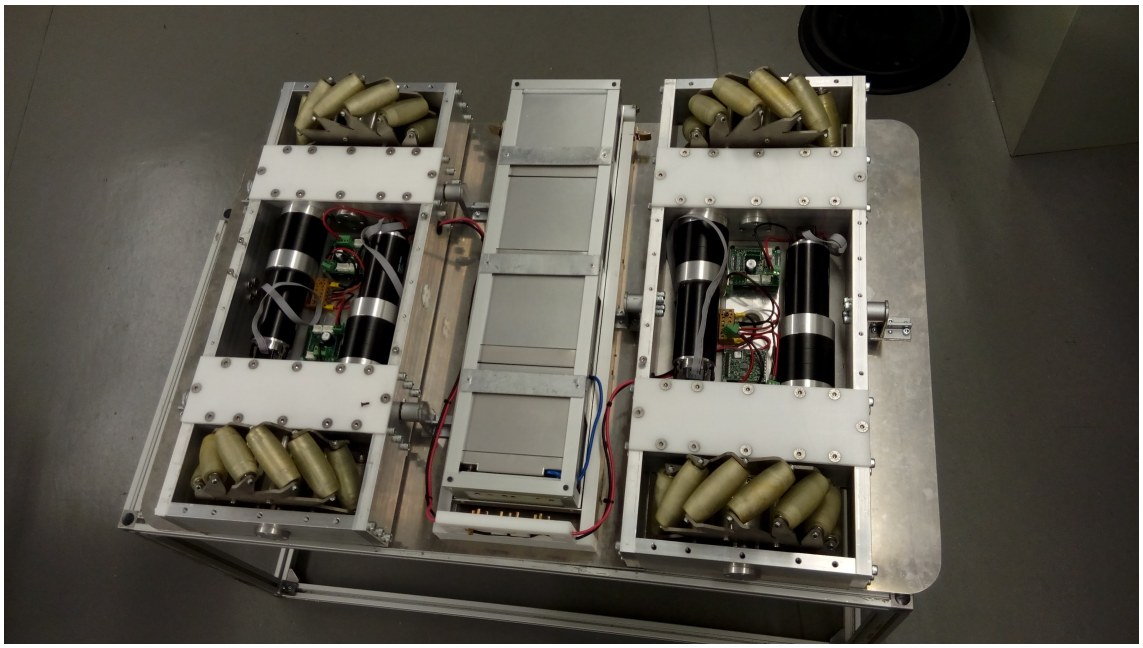


Figura 4.2: AVG em fase de montagem

Pretende-se estabelecer comunicação com o controlador de motores DZCANTE-012L080 da Advanced Montion Controls . É um controlador desenvolvido para o controlo de motores com ou sem escovas, com *design* compacto ideal para aplicações embarcadas. O controlador foi montado com uma placa de interface MC1XDZC02-QD como representado na figura 4.3. O controlador suporta comunicação baseada em CANopen seguindo o protocolo CiA 301 e o perfil de dispositivo CiA 402.

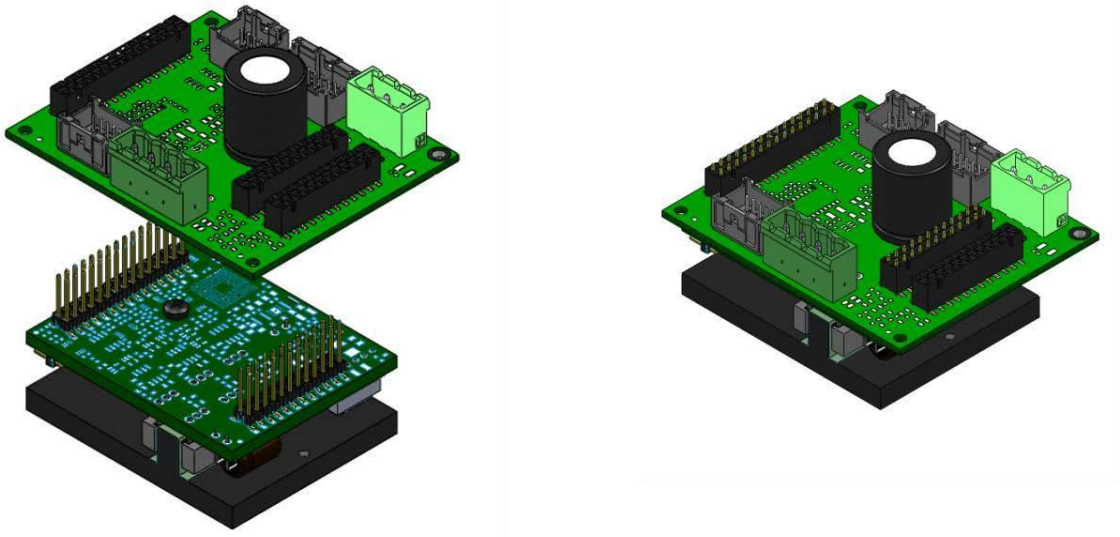


Figura 4.3: Controlador e placa de interface

O motor a controlar é o RE 65 (250W) da Maxon Motor. A tabela 4.1 representa as caraterísticas do motor.

Tabela 4.1: Caraterísticas do motor

Tipo	Motor DC de ímanes permanentes
Tensão Nominal	48V
Velocidade Nominal	3270 rpm
Torque Nominal	746
Corrente Nominal	6.02A
Eficiência máxima	81%

O adaptador CAN utilizado é o VSCom NET-CAN 110 um adaptador que funciona através da Internet. Trata-se de um adaptador com suporte para Linux e permite velocidades de transmissão de 1Mbit/s.

Figura 4.4: Adaptador CAN

4.1 CANopen

CANopen é um sistema de comunicação baseado na rede de campo CAN (*Controller Area Network*), e define os protocolos de comunicação e o perfil dos dispositivos. Este tipo de protocolo foi desenvolvido como um padrão para sistemas embarcados e possui uma grande capacidade de configuração. Foi inicialmente desenvolvido para a automação e em especial para o controlo de motores, mas hoje é usado em equipamentos médicos, caminhos-de-ferro, domótica, etc. O objetivo deste padrão é a agilização do processo de desenvolvimento, para isso, fornece um conjunto de objetos padronizados de comunicação (COB) para processos de tempo crítico, configuração e gestão de dados da rede [12].

Em termos do modelo de interconexão de sistemas abertos (OSI), a rede de campo CAN cobre as duas primeiras camadas, a camada física e a de ligação de dados, e o CANopen cobre as cinco camadas superiores desde da camada de rede à camada de aplicação. As especificações da camada de aplicação são definidas pela norma CiA DS 301 [13].

4.1.1 Rede de Campo CAN

Relativamente ao CAN, este é definido pelo protocolo padrão ISO 11898 e cobre duas camadas do modelo OSI, que são:

- **Camada Física -** Especifica as características elétricas e físicas do barramento, bem como a forma como o hardware converte os carateres da mensagem em sinais elétricos para sua transmissão e a forma como converte os sinais recebidos em carateres. Esta é a única camada que apenas pode ser implementada a nível de hardware. A tipologia de rede utilizada pelo CAN é em barramento e utiliza um par entrelaçado como meio de transmissão. O sinal elétrico é transmitido em modo diferencial e com codificação NRZ o que permite menor interferência eletromagnética a velocidades elevadas. É possível atingir velocidades máximas de 1Mbps em distâncias inferiores a 50 metros.
- **Camada de Ligação de Dados -** Esta camada é responsável pela codificação das mensagens para serem enviadas pela camada física e descodificação das mensagens recebidas.

O CAN tem assumido uma posição relevante, levando muitas empresas a abandonar os seus protocolos proprietários, adotando CAN devido à sua versatilidade na rede. A disponibilidade de circuitos integrados dedicados a este protocolo, aliada à robustez e eficiência do CAN ditou o seu alargamento a mercados como a automação industrial, domótica, robôs, navios, equipamentos médicos, etc [14].

4.1.1.1 Trama de dados

A figura 4.5 ilustra a constituição da trama de dados CAN.

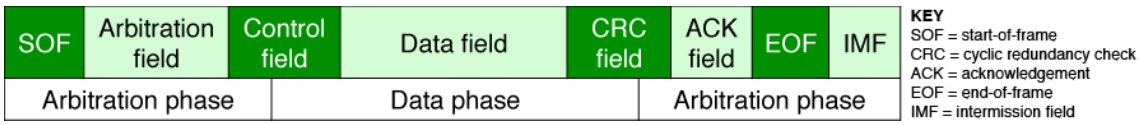


Figura 4.5: Estrutura das tramas de dados CAN

A trama é constituída pelos seguintes elementos:

- **SOF -** Início da trama;
- **Arbitration field -** Corresponde a um identificador único e representa a prioridade da mensagem;
- **Control field -** Corresponde ao campo de controlo e indica o número de bytes de dados;
- **Data field -** Corresponde aos dados;
- **CRC field-** Corresponde à verificação de redundância cíclica e é um método de identificação de erros;

- **ACK field -** Corresponde a um sinal de reconhecimento;
- **IMF -** Indica o fim da trama.

A trama apresentada possui um identificador de 11 bits, no entanto existe um formato estendido deste tipo de trama com um identificador de 29 bits. O controlo de acesso à rede é feito através do identificador. Qualquer dispositivo da rede pode tentar transmitir uma trama para a rede caso esta esteja livre num determinado instante. Caso dois dispositivos iniciem a transmissão de uma trama exatamente ao mesmo tempo, apenas o que tiver o identificador mais prioritário irá completar o envio da trama. Isto acontece porque quando enviam uma trama, os dispositivos escutam a rede, e no momento em que o identificador da trama que está a ser transmitida não coincide com o da trama que estão a querer transmitir, isso implica que um outro dispositivo com maior prioridade está a aceder a rede, e então para de transmitir. O valor 0 indica a trama mais prioritária. A figura 4.6 representa a concorrência entre o "Node 15" e o "Node 16". Assim que o "Node 16" verifica que existe outro dispositivo com maior prioridade na rede não transmite mais.

	Start Bit	ID Bits										The Rest of the Frame
		10	9	8	7	6	5	4	3	2	1	0
Node 15	0	0	0	0	0	0	0	0	1	1	1	1	
Node 16	0	0	0	0	0	0	0	1	Stopped Transmitting				
CAN Data	0	0	0	0	0	0	0	0	1	1	1	1	

Figura 4.6: Acesso prioritário à rede

O CAN possui diversos mecanismos de controlo de erros, tornando assim a rede mais confiável e com índice de erros não detetados bastante baixo. Cada dispositivo deve ser capaz de detetar a ocorrência de erros e informar os restantes dispositivos da rede [15].

4.1.2 Camada de aplicação

O CANopen é um protocolo de alto nível padronizado pela CiA (CAN in Automation). Um dos pontos deste protocolo é a existência de perfis para os dispositivos da rede que funcionam como um molde para as empresas que os constroem. Estes perfis indicam as funcionalidades de carácter obrigatório que garantem a operacionalidade e interconectividade do dispositivo enquanto membro da rede. O dicionário de objetos é o principal ponto destes perfis e permitem definir o acesso aos dados de forma uniforme entre dispositivos. Cada objeto pode ser acedido através de um índice de 16 bits e um sub-índice de 8 bits.

Cada rede CANopen deve possuir um mestre responsável pelos serviços de gestão da rede. Cada rede suporta até 127 dispositivos escravos e cada dispositivo na rede é normalmente chamado de nó. Cada nó é identificado da rede através de um identificar único para cada escravo na rede. O mestre é responsável por um conjunto de serviços que controla a comunicação na rede e o estado dos escravos.

Os objetos responsáveis pela comunicação são denominados COBs e são agrupados de acordo com os tipos de dados e a forma como são enviados ou recebidos por um determinado dispositivo. Os tipos de COBs existentes são:

- **SDO -** *Service Data Object* são objetos que permitem o acesso direto ao dicionário de objetos através do seu índice. São de baixa prioridade na rede, no entanto, não menos importantes e sendo utilizados para configuração de um nó da rede. Existem dois tipos de SDOs: cliente SDO, que é responsável por efetuar um pedido de leitura ou escrita num objeto e o servidor SDO que atende este pedido;

- **PDO -** *Process Data Object* são os principais objetos utilizados para a transmissão de dados durante o processo normal de operação do dispositivo sendo por isso mais prioritários na rede. Estes permitem a transferência de dados sem necessidade de indicar explicitamente qual os objetos que estão a ser transmitidos,sendo por isso necessário uma configuração prévia do mapeamento dos dados;

- **NMT -** *Network Management* corresponde as mensagens de mecanismos de inicialização, monitorização, configuração e tratamento de erros. Com este objeto o mestre da rede pode, por exemplo, forçar um escravo da rede a transmitir um mensagem de estado;

- **EMCY -** *Emergency Object* corresponde a um objeto responsável pelo envio de mensagens que indicam a ocorrência de erros num dispositivo. Quando é detetado um erro é enviada uma mensagem pelo dispositivo que o detetou, denominado produtor EMCY, e caso exista um dispositivo a monitorizar a rede, consumidor EMCY, é possível programar uma ação desejada;

- **SYNC -** *Synchronization Object* é um sinal periódico de sincronização entre todos os nós da rede. O nó que produz é denominado produtor SYNC e os restantes consumidores SYNC.

Para cada dispositivo de uma rede CANopen existe um ficheiro EDS que contém informações sobre o funcionamento desse dispositivo, bem como todos os objetos existentes no dicionário desse dispositivo.

4.1.3 CiA 402

CiA 402 é uma norma que define um conjunto de especificações do perfil e o comportamento funcional dos controladores de motores servo, inversores de frequência e motores de passo. Este perfil de dispositivo define vários modos de operação e parâmetros de configuração correspondentes. O perfil do dispositivo inclui um estado autómato finito (FSA) que define o seu comportamento externo e interno. Este estado decide as palavras de comandos (*control-word*) que são aceites e quando é aplicada potência aos motores. O estado pode ser alterado através de comandos de controlo enviados por um controlador superior, ou devido a inventos internos. Este estado atual é definido pela palavra de status (*status-word*). As palavras de comando e os diferentes valores de comando (por exemplo velocidade) encontram-se mapeados por defeito em RPDOs (Objetos de

receção de dados do processo), enquanto que, as palavras de status e os diferentes valores reais (por exemplo posição e velocidade) são mapeados por defeito em TPDOs (Objetos de transmissão de dados do processo). [16]

A norma CiA 402 é um dos perfis de controlo de movimento mais especificados, mas por outro lado, a quantidade de funções e parâmetros opcionais limita a substituição de alguns dispositivos compatíveis para esta norma. Isto implica que alguns dos fabricantes de controladores afirmam conformidade mas apenas aplicam um subconjunto de funções obrigatórias.

4.2 CANopen em ROS

Existe em fase experimental um *package* para ROS que permite a implementação genérica de CANopen. Este *package* denominado "ros_canopen" incluiu a abstração da camada CAN, fornece um mestre CANopen com gestão de objetos e dispositivos e inclui suporte para dispositivos de perfil CiA 402. [17]

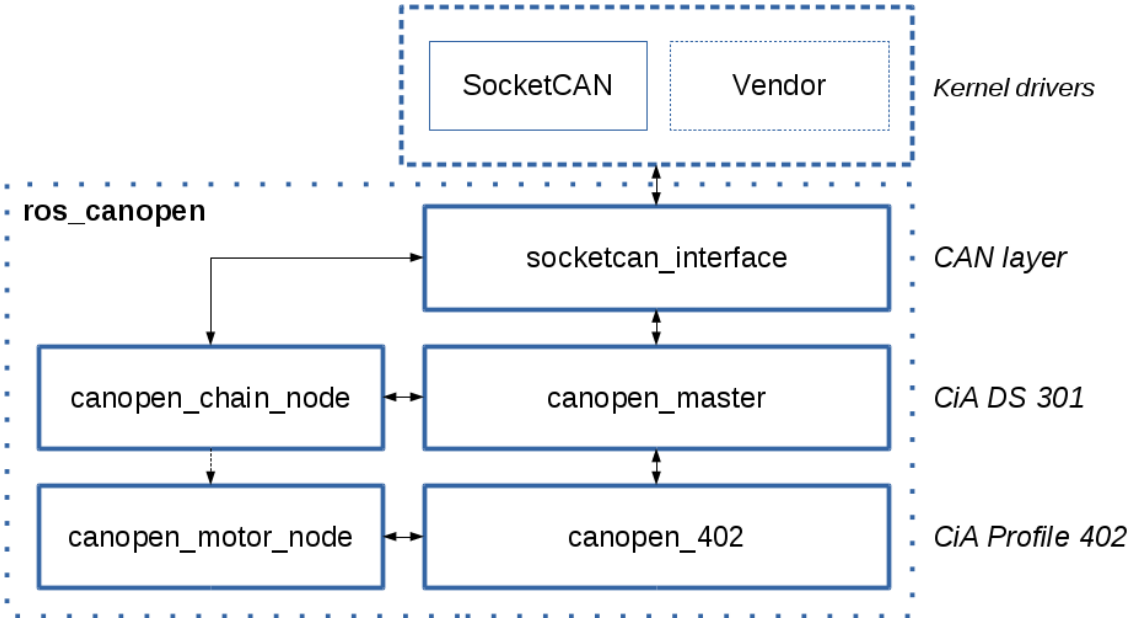


Figura 4.7: Estrutura "ros_canopen"

A figura 4.7 representa a estrutura do "ros_canopen". Os *subpackages* que o constituem são:

- **socketcan_interface -** Este *package* fornece uma interface CAN genérica;
- **canopen_master -** Contém a implementação de um mestre para o protocolo CANopen DS 301. Este fornece implementação para interfaces e serviços CANopen tais como: dicionário de alto nível para todos objetos simples, leitura de ficheiros EDS, configuração automática de objetos de comunicação, implementação de SDO cliente, implementação de PDO publicador e subscritor, Sincronização SYNC , entre outros.

- **canopen_chain_node -** Faz a gestão da rede CAnopen com um ou vários nós, cada um destes nós precisa de ser configurado com um ficheiro EDS, e é responsável pelo sinal de sincronismo SYNC;
- **canopen_402 -** Implementa o protocolo CiA 402;
- **canopen_motor_node-** Este *package* fornece uma interface "ros_control" para motores compatíveis com o perfil CiA 402. Este herda de "canopen_chain node" as funcionalidades de gestão da rede.

4.2.1 Implementação

Para a implementação em ROS foi utilizado o *package* "canopen_motor_node" para estabelecer a comunicação com o controlador do motor. A interface com este nó é realizada através do *package* "ros_control" e como pretendemos controlar a velocidade do motor é necessário utilizar uma interface do tipo "hardware_interface::VelocityJointInterface". Uma vez que o mestre CANopen apenas suporta os tipos simples de dados, foi necessário modificar o ficheiro EDS das drives e eliminar todos os dados com formatos não padronizados que o fabricante acrescentou.

Foi necessário ainda criar um ficheiro URDF (*Unified Robot Description Format*) que define a junta do robô que se pretende controlar. Isto permite que de forma automática o nó "canopen_motor_node" leia os limites associados a esta junta. Como se pretende controlar a rotação das rodas trata-se de uma junta rotacional contínua sem limite de posição.

O ficheiro de arranque do nó "canopen_motor_node" necessita de incluir parâmetros do barramento como o nome do dispositivo CAN que se vai utilizar e o intervalo do sinal de sincronismo. É ainda necessário definir para cada nó da rede diversos parâmetros para o seu funcionamento, tais como: Ficheiro EDS com configurações, o modo de operação (controlo de velocidade neste caso), o nome do nó CAN que deve corresponder ao nome da junta definida no ficheiro URDF e ID (identificador do nó CAN que o motor tem na rede). É possível ainda definir parâmetros como taxas de conversão entre os valores (velocidade, posição e torque) do dicionário de objetos e os valores expostos através da interface do "ros_control".

Foi necessário implementar um nó de ROS que recebe a velocidade pretendida através de um tópico e utiliza a interface "hardware_interface::VelocityJointInterface" para enviar o comando ao controlador das drives.

Contudo, estes controladores não respeitam na totalidade a norma CiA 402, tendo algumas restrições no dicionário de objetos que impedem o funcionamento do nó "canopen_motor_node", assim sendo, é necessário alterar o código fonte deste nó para permitir o funcionamento do controlador. No entanto foi possível comprovar o funcionamento da comunicação CANopen através do nó "canopen_chain_node", a partir do qual foi possível aceder aos objetos do dicionário do controlador dos motores.

Capítulo 5

Protótipo Discovery Q2

5.1 Prototipo inicial

A plataforma Discovery Q2, construída pela empresa Hangfa, é uma plataforma omnidirecional com rodas Mecanum, o chassi em aço tem a forma de um bloco retangular e as dimensões exteriores são $374x320x114mm$.

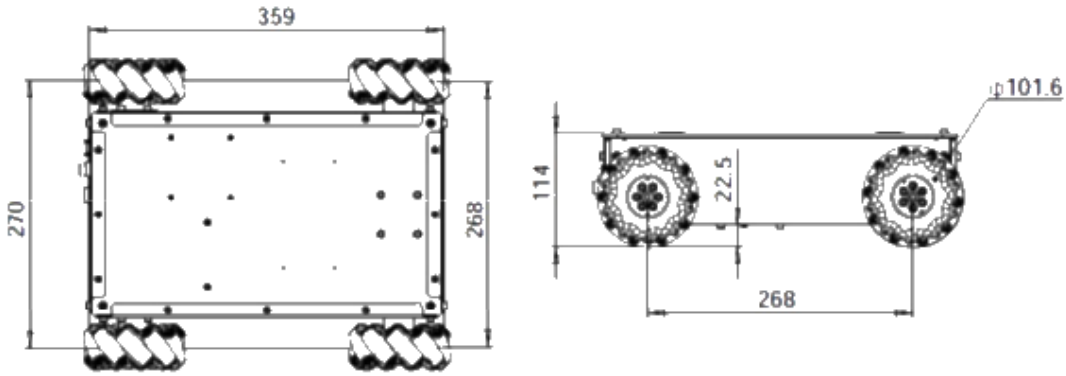


Figura 5.1: Dimensões da plataforma

A estrutura da plataforma pode ser vista na Figura 5.1. É de salientar que na parte frontal da plataforma (parte superior da figura) existe um sistema de suspensão pendular coaxial que permite que as quatro rodas mantenham o contacto com o piso aumentando assim o desempenho em piso irregular.

A plataforma tem um peso de cerca de $7Kg$ e uma capacidade de carga nominal de $20Kg$. A sua velocidade máxima linear é de $0.5m/s$ e a velocidade de rotação máxima é $90°/s$. A alimentação da plataforma é fornecida por uma bateria de lítio de $12V$ e com uma capacidade de $10.4Ah$ possibilitando uma autonomia superior a 10 horas com uma carga nominal de $10Kg$.

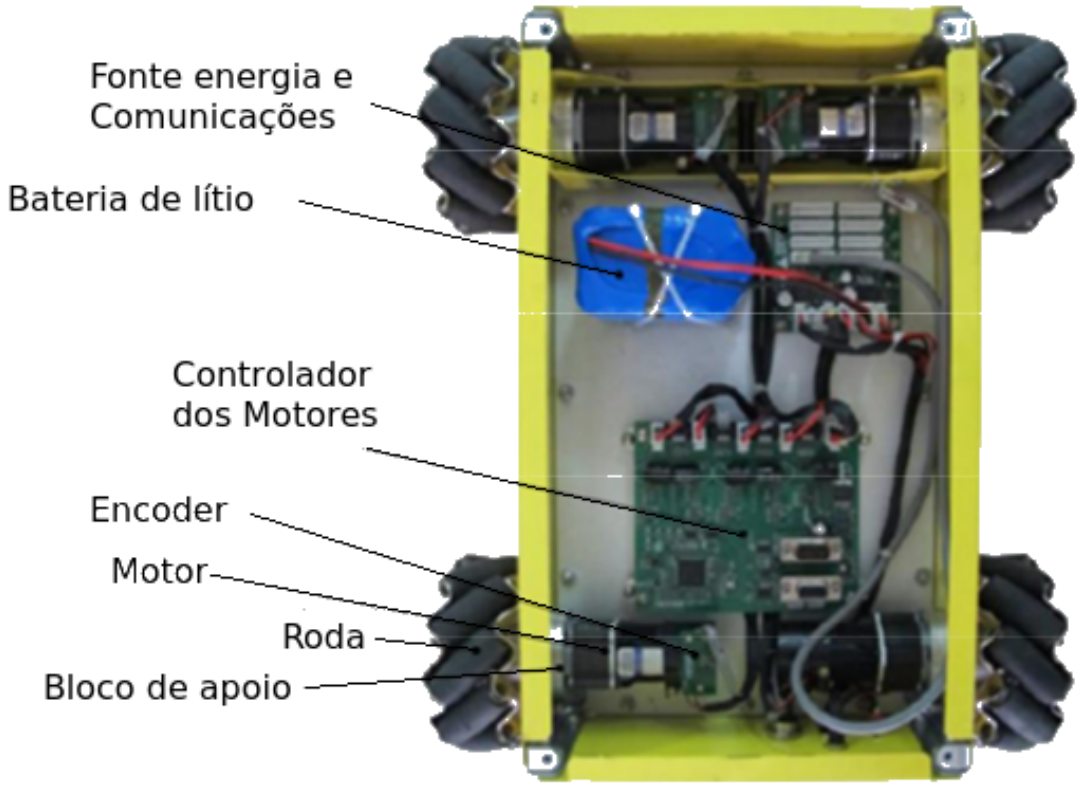


Figura 5.2: Estrutura da plataforma

5.1.1 Rodas

As rodas do protótipo são QMA-10 construídas pela Hangfa. Na tabela 5.1 estão descritas as características desta roda. É de salientar que cada rolo de borracha possui 2 rolamentos internos permitindo que este se mova de forma constante e suave.

Tabela 5.1: Características das rodas

Roda	QMA-10 Omni
Diâmetro	101.6mm
Largura	45.7mm
Número de rolos	10
Número de rolamentos	20
Material do suporte	Aço
Capacidade carga	45Kg

5.1.2 Motores

A plataforma possui quatro motores DC, um para cada roda, necessário para o controlo individual de cada roda. O motor usado é o Faulhaber 2342 equipado com caixa redutora e com encoder

de eixo incremental. Trata-se de um motor de corrente continua sem núcleo e com alta densidade de potência proporcionando torque elevado e rápida velocidade de resposta. As características do motor estão descritas na tabela 5.2

Tabela 5.2: Características dos motores

Tensão nominal	12V
Corrente nominal	1.1A
Potência nominal	45.7mm
Velocidade de rotação sem carga	6800rpm
Velocidade nominal de rotação	5800rpm
Relação de redução	64:1
Resolução do encoder	12 ppr
Tensão nominal do encoder	5V

5.1.3 Bloco de Apoio

Cada roda está fixada no bloco de apoio pelo veio de transmissão. Este veio de transmissão está ligado ao eixo do motor, em vez de a roda estar ligada ao eixo do motor diretamente. Isto reduz a carga radial do eixo do motor melhorando significativamente a capacidade de carga do robô e a condição de funcionamento do motor, prolongando a vida útil do motor.

5.1.4 Fonte de energia e Comunicações

A módulo IFB1205 é usado para concentrar a fonte de energia e de comunicação. A alimentação ao módulo é fornecida pela bateria e este alimenta os restantes módulos. A figura 5.3 representa o esquema elétrico do módulo.

Os conectores J8 a J15 são interface que permitem interligar outros módulos, tendo com ligações a alimentação, um barramento CAN e uma interface serial.

5.1.5 Controlador dos motores

Este módulo controla a velocidade dos quatro motores com precisão em malha fechada. É possível controlar a velocidade do robô diretamente (V_X, V_Y e W_z) ou a velocidade de cada roda individualmente através de comunicação via barramento CAN ou interface serial RS-232.

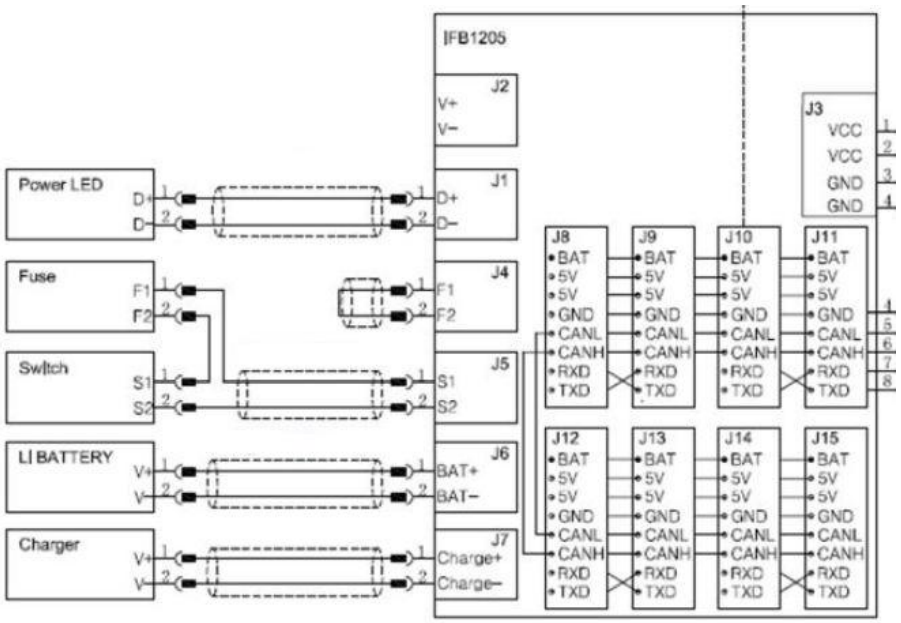


Figura 5.3: Esquema elétrico do módulo IFB1205

5.2 Modificações

O protótipo inicial foi alterado de modo a ser possível torná-lo num robô autónomo e testar o controlador de caminhos omnidirecional desenvolvido. A figura 5.4 representa o sistema desenvolvido, onde a azul está representado o protótipo inicial e a cinzento o que foi acrescentado ao protótipo.

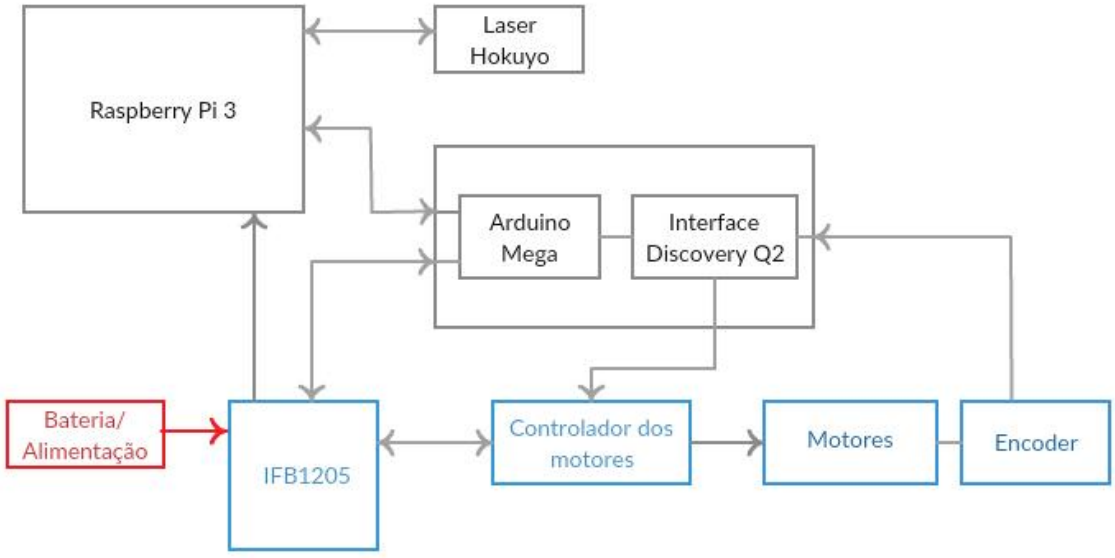


Figura 5.4: Visão geral do protótipo desenvolvido

As figuras 5.5 e 5.6 carateriza o robô depois das alterações efetuadas.

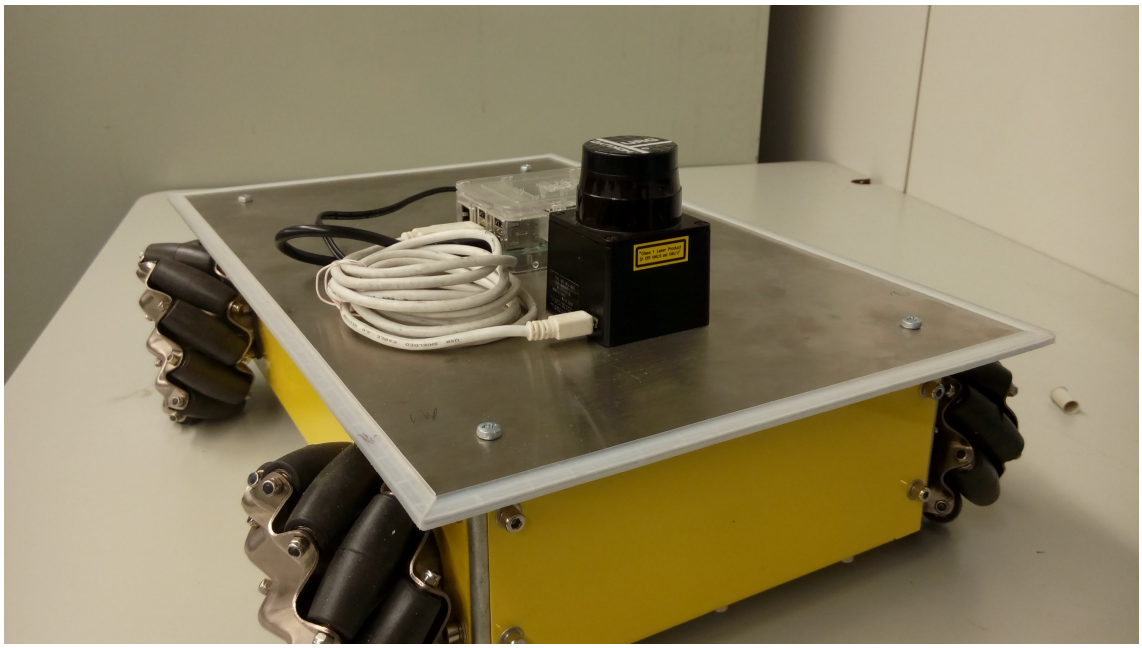


Figura 5.5: Protótipo Discovery Q2

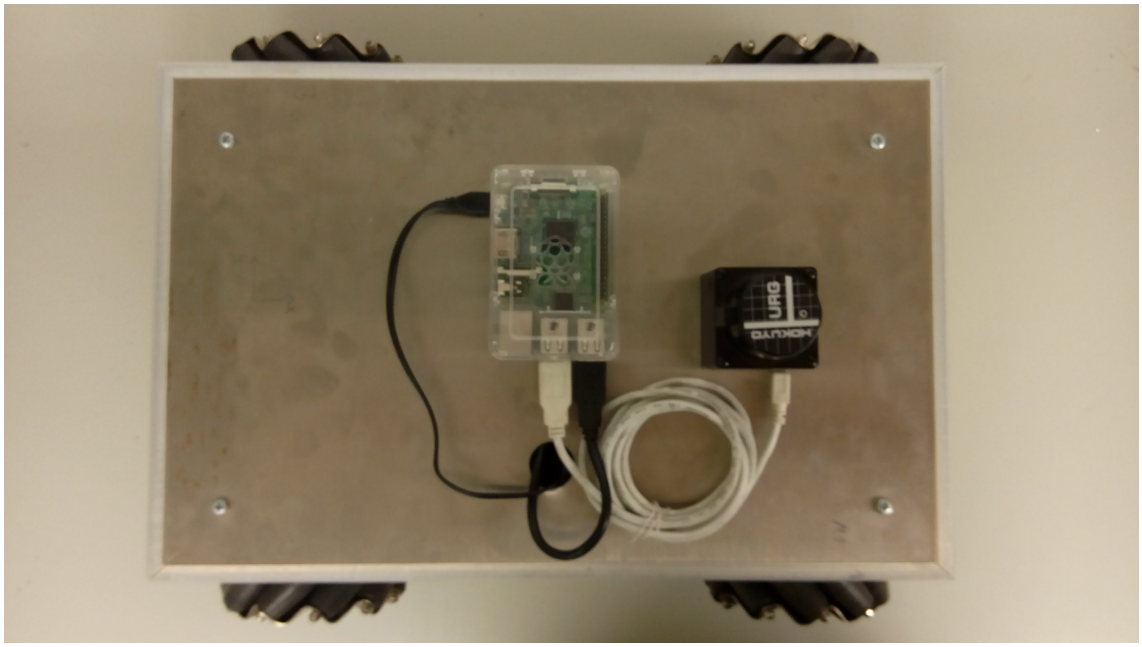


Figura 5.6: Protótipo Discovery Q2 (vista topo)

5.2.1 Interface Discovery Q2

O controlador dos motores utiliza o valor dos encoders para fechar a malha de controlo, no entanto, o valor deste não está acessível ao utilizador. Uma vez que a odometria é de grande importância na localização do robô, uma das soluções passava pela substituição do controlador dos motores, no entanto implicava um preço mais elevado para o projeto. A solução adotada implica derivar os sinais dos encoders e efetuar a leitura através de um microcontrolador. Este microcontrolador seria ainda responsável por comunicar com o controlador dos motores.

Para controlar com os motores foi escolhido o protocolo RS-232 desta forma seriam necessário que o microcontrolador tivesse duas portas serial, uma para comunicar com o controlador dos motores e outra para comunicar com o computador do robô. Foi escolhida a placa Arduino Mega 2560 para fazer esta interface entre o computador e o controlador dos motores. A tabela 5.3 apresenta as principais caraterísticas deste placa.

Tabela 5.3: Características do Arduino Mega 2560

Tensão de operação	5V
Microcontrolador	ATmega2560
Pinos digitais	54
Pinos analógicos	16
UARTs	4
Conexão ao Computador	Cabo Usb

Foi desenvolvida uma placa (PCB), nomeada de "Interface Discovery Q2", que assenta no topo do Arduino Mega. Esta placa recebe os sinais dos quatro encoders através de quatro conetores IDC e disponibiliza esses mesmo sinais noutros quatro conetores IDC. Os sinais de interesse são derivados até ao pinos digitais do microcontrolador. Na placa existe ainda um conetor que faz a interface com a placa IFB1205, utilizado para obter alimentação para o Arduino Mega e também para comunicar com o controlador dos motores. A comunicação é feita por RS-232, tendo sido por isso utilizado o um circuito integrado MAX232 para fazer a interface para o microcontrolador.

A figura 5.8 representa parte do esquema da placa Interface Discovery Q2. Do lado direito é possível ver as ligações ao circuito integrado MAX232 e a sua ligação ao conetor "RS232" que representa a interface de ligação com a placa IFB1205. Do lado esquerdo encontra-se dois conetores, onde um recebe os sinais do *encoder* e o outro liga ao controlador dos motores, estes encontram-se ligados de forma a passar os sinais e existe um replica dos sinais A4 e B4 para os pinos do microcontrolador. O esquema completo da placa está no anexo C.

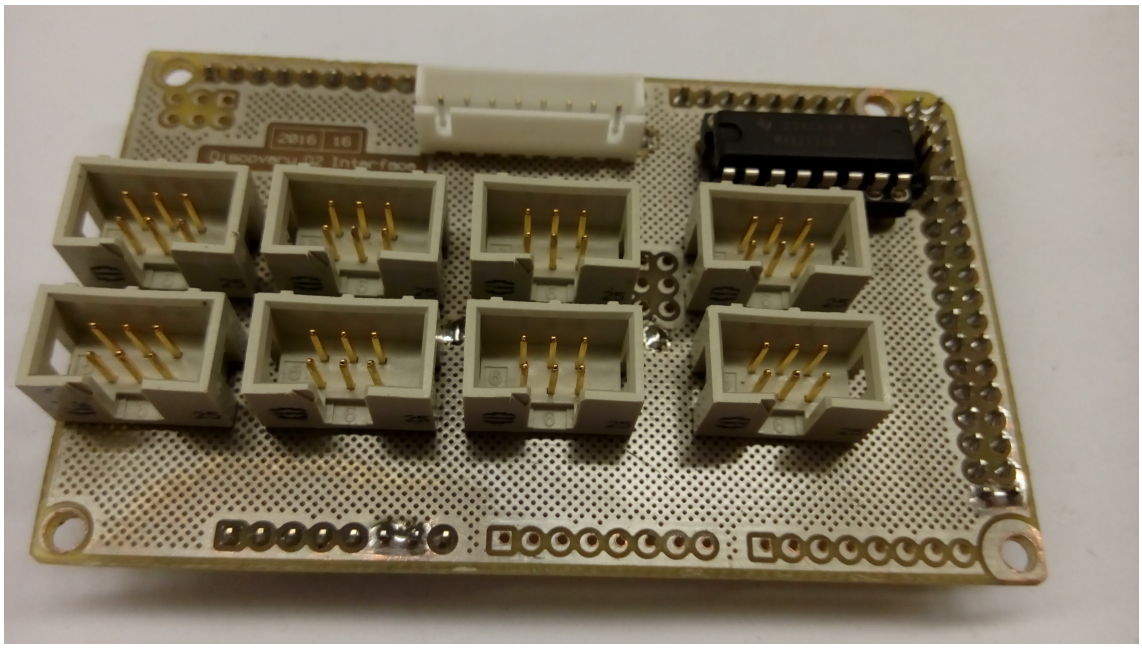


Figura 5.7: Placa Interface Discovery Q2

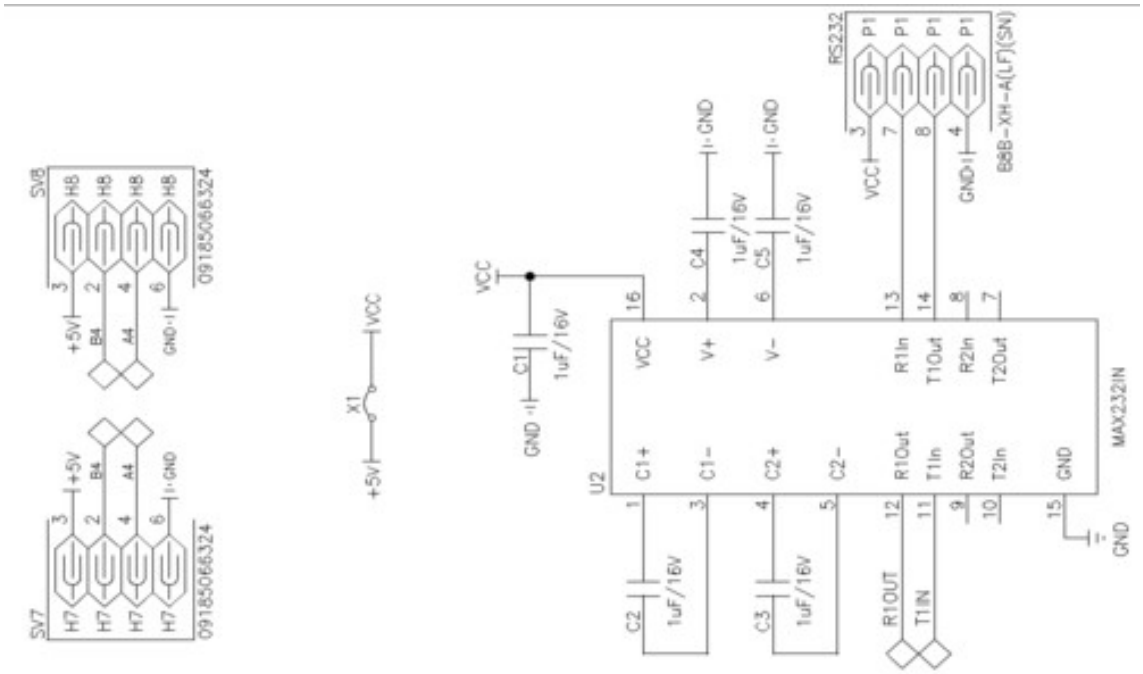


Figura 5.8: Esquema parcial da placa Discovery Q2

A figura 5.9 representa a montagem da Interface Discovery Q2 por cima do controlador dos motores ficando assim dentro da carcaça do robô.

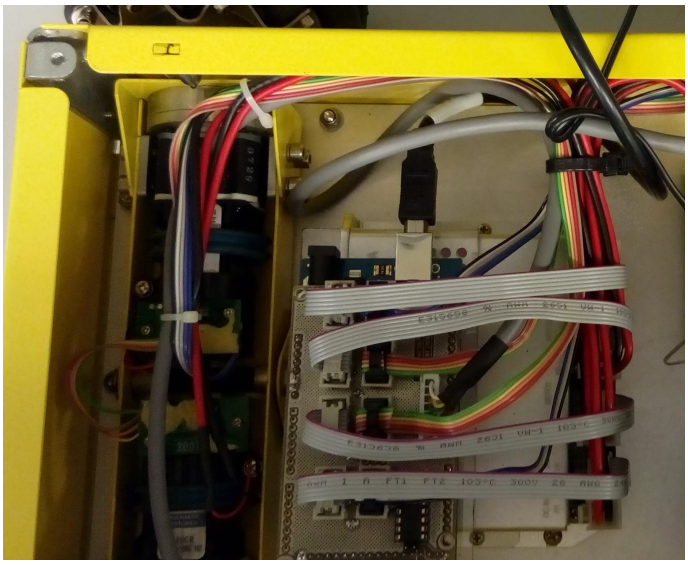


Figura 5.9: Interface Discovery Q2 montada no robô

5.2.2 Laser

Foi instalado um laser no protótipo de forma a permitir a localização absoluta do robô(em relação a um referencial externo). Em comparação com a odometria estes têm uma frequência de amostragem baixa e por vezes podem ser inconclusivos quanto ao posicionamento do robô. No entanto a sua fusão com a odometria do robô, permite a correção de erros acumulados por esta, obtendo assim um posição do robô mais fidedigna.

O laser utilizado corresponde ao modelo URG-04LX-UG01 da Hokuyo. Na tabela 5.4 são apresentadas as principais caraterísticas deste sensor.

Tabela 5.4: Características do Laser URG-04LX-UG01

Consumo de energia	2.5W
Alcance	5600mm
Abertura	240°
Precisão	±30mm
Resolução	0.352°
Conexão de Dados	Cabo Usb
Alimentação	Cabo Usb

5.2.3 Computador

O computador selecionado para o projeto é o "Raspberry Pi3", trata-se de um computador de placa única (*SBC - Single board computer*), de baixo custo e com um desempenho razoável para o projeto pretendido. O sistema operativo utilizado foi o "Raspbian Jessie"que têm compatibilidade com ROS. As principais caraterísticas do computador encontram-se descritas na tabela 5.5.

Tabela 5.5: Características do "Raspberry Pi3"

Processador	1.2GHZ 64-bit quad-core ARMv8
Memória RAM	1GB
Conetividade	802.11n Wireless LAN Porta Internet Bluetooth 4 portas USB 40 pinos programáveis (GPIO)
Saída Vídeo	HDMI
Alimentação	Micro USB

5.3 Software Implementado

O software implementado para o protótipo visa a abstração de hardware do robô. A camada de abstração de hardware (HAL) será desenvolvida em C++ e terá como objetivo a sua integração em ROS. Foi ainda necessário desenvolver o código para o microcontrolador utilizado na Interface Discovery Q2.

5.3.1 Microcontrolador

O microcontrolador tem com principal objetivo a leitura dos *encoders*. Será ainda responsável por fazer a interface entre o computador e o controlador dos motores. A leitura dos *encoders* é efetuada de forma periódica a uma frequência de amostragem adequada e a sua posição é enviada de forma periódica para o computador. O protocolo de comunicação com o controlador dos motores é implementado no computador e por isso o microcontrolador apenas é responsável por reencaminhar os pacotes para o controlador. O código implementado no IDE do Arduino encontra-se no anexo B. Em resumo temos:

- **Configuração portas digitais -** Definição dos pinos dos *encoders* e configuração como entradas. Ter em atenção para configurar como entrada sem *pull up* para não interferir com o controlador dos motores.

- **Leitura dos *encoders* -** Definir um temporizador com um período de $200\mu s$ para gerar uma função de interrupção. Essa função é responsável por verificar o valor dos sinais atuais e incrementar a posição dos encoders.

- **Ciclo contínuo -** Definir um ciclo contínuo responsável por encaminhar os pacotes do computador para o controlador e por enviar a posição dos *encoders* para o computador com um determinado intervalo de tempo.

5.3.1.1 Leitura encoders

Trata-se de um *encoder* incremental onde estamos interessados em ler os sinais *A* e *B* que se encontram representados na figura 5.10. Os sinais são impulsos que se encontram desfasados cerca de 90°.

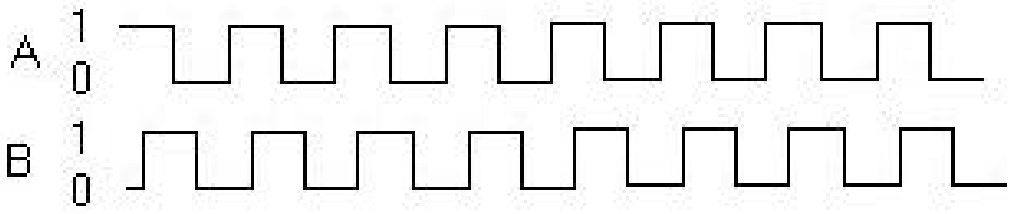


Figura 5.10: Sinais A e B dos encoders

O incremento da posição dos *encoders* vai depender do valor dos sinais na amostragem anterior e da amostragem atual. A tabela 5.6 representa o valor do incremento em função do valor desses sinais.

Tabela 5.6: Incremento do valor da posição

Anterior		**Atual**		**Incremento**
A	**B**	**A**	**B**	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	-1
0	0	1	1	0
0	1	0	0	-1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	-1
1	1	0	0	0
1	1	0	1	-1
1	1	1	0	1
1	1	1	1	0

O período de amostragem mínimo foi calculado tendo em conta a velocidade máxima do motor. Sendo que o valor desta é de $6800rpm$, cerca de $133,33$ rotações por segundo, e o *encoder* tem uma resolução de 12 impulsos por rotação, a frequência máxima do sinal é de $1600Hz$. Pelo teorema da amostragem, a frequência de amostragem tem de ser pelo menos 2 vezes superior à frequência de sinal, desta forma a frequência mínima de amostragem é de $3200Hz$. Este valor

corresponde a um período máximo de amostragem de $312,5\mu s$, sendo por isso escolhido um valor ainda inferior ($200\mu s$).

5.3.1.2 Ciclo contínuo

A comunicação entre o microcontrolador e o computador é feita através da porta serial 0 e a comunicação com o controlador dos motores através da porta serial 3. Como o computador apenas envia mensagens que são para o controlador dos motores, o microcontrolador durante o ciclo verifica a existência dessas mensagens através da função *Serial.available* e caso existam estas são lidas e enviadas para a porta serial 3 através da função *Serial3.write*. Caso o controlador envie mensagens, estas são enviadas para o computador.

A posição dos *encoders* é enviada de forma periódica para o computador. Durante o ciclo contínuo é verificado a diferença entre o tempo atual e o tempo em que foi enviado a última mensagem sobre a posição e quando este valor for superior ao intervalo definido é enviada uma nova mensagem.

5.3.2 Nó Driver Discovery Q2

O Nó "Driver Discovery Q2"é o principal nó que permite a abstração de hardware. Este é responsável por comunicar com o microcontrolador, estimar a pose do robô, enviar os comandos ao controlador dos motores e ainda definir a posição do laser em relação ao referencial do robô.

Ao nível do ROS este nó subscreve um tópico de velocidade e publica um tópico de pose para a "tf", do tipo "geometry_msgs/Pose2D". A "tf"é um sistema distribuído de controlo de múltiplos referenciais e permite ao utilizador manter a relação entre os referencias e transformar pontos entre qualquer dois referenciais. O referencial do laser em relação ao do robô é configurado neste nó através da indicação à "tf"da sua posição e orientação. Na figura 5.11 é possível observar tópicos publicados e subscritos por este nó e que permite a interação com outros nós. O nó tem a forma oval e os tópicos são retangulares.

Figura 5.11: Interface do nó Driver Discovery Q2 (/driver_motors)

O microcontrolador envia periodicamente o valor da posição dos *encoders* para o computador através da porta serial. Este nó controla o tráfego da porta serial, desta forma sempre que chega trama de dados ao computador é executada uma função responsável por verificar a validade da trama de dados e executar as funções . Esta trama tanto pode ser uma trama com origem no

controlador dos motores como no microcontrolador, assim a função deve ser capaz de destingir os dois tipos de tramas. Como o microcontrolador não interpreta as trama dos motores, a trama da posição dos *encoders* pode surgir no meio da trama dos motores, no entanto é garantido que a trama da posição tem tamanho fixo e é enviada sem interrupção. Sempre que recebida uma trama com posição dos *encoders* recorrendo ao modelo de odometria é atualizada para a "tf"a nova estimação de pose do robô.

Este nó subscreve a velocidade a ser enviada aos motores, assim foi definido um ciclo periódico com que este nó lê a velocidade publicada por um outro nó, no tópico "/cmd_vel"(do tipo "geometry_msgs/Twist"), e codifica a trama de dados, com as novas velocidades pretendidas, para enviar ao controlador dos motores. Neste ciclo é ainda verificada a validade do tópico "/cmd_vel", isto é, é verificado o instante de tempo em que a última informação foi publicada neste tópico, assim caso esta informação tenha sido publicada a mais de um tempo definido esta é considerada inválida e é enviada velocidade zero para os motores. Este permite que em caso de falha do nó que impõe a velocidade aos motores, estes param para prevenir o descontrolo do robô.

Para comunicar com o controlador dos motores é necessário implementar o protocolo definido pelo construtor. O formado do pacote de dados é definido como na tabela 5.7.

Tabela 5.7: Formato do pacote de dados

	Descrição
Caractere inicial	1 byte, indica o início do pacote de dados e tem o valor de 0xAA
Tipo de dispositivo	1 byte, indica o tipo de dispositivo e para este controlador tem o valor de 0x40
Endereço do dispositivo	1 byte, indica o endereço do dispositivo e por padrão tem o valor de 0x01
Código da função	1 byte, indica o comando a ser executado pelo dispositivo
Tamanho dos dados	1 byte, Número de bytes de dados válidos
Dados	N bytes de dados, onde N é o número definido pelo tamanho dos dados
CRC	2 bytes, onde CRC significa verificação de redundância cíclica e é um método de verificação de erros.
Caractere final	1 byte, indica o fim do pacote e tem o valor de 0x0D

A tabela 5.8 apresenta os parâmetros possíveis de configurar e a sua descrição.

O *script* de arranque deste nó com as configurações do protótipo encontra-se no anexo A.1

5.3.3 Nó Omnijoy

O nó Omnijoy foi desenvolvido para poder movimentar o robô. Trata-se de um nó que subscreve um tópico que dá informações sobre um *joystick* e conforme as teclas pressionadas e a intensidade dos botões analógicos gera um comando de velocidade que é publicado. O *joystick*

Tabela 5.8: Parâmetros do nó Driver Discovery Q2

Parâmetro	Descrição
port	Indicação do nome da porta serial onde está ligado o microcontrolador
period	Período em milissegundos com que é enviada a velocidade aos motores
data_time_out	Validade temporal em segundos do comando "/cmd_vel"
wheel_radius	Raio da roda Mecanum
enc_pulses_per_revolution	Número de impulsos por revolução da roda
dist_to_wheel_x	Distância no eixo dos X ao centro da roda
dist_to_wheel_y	Distância no eixo dos Y ao centro da roda
odom_frame_id	Nome do referêncial da odometria
base_frame_id	Nome do referêncial da base
laser_frame_id	Nome do referêncial do laser
laser_x_offset	Distância no eixo do X do referencial do laser ao referencial base
laser_y_offset	Distância no eixo do Y do referencial do laser ao referencial base
laser_z_offset	Distância no eixo do Z do referencial do laser ao referencial base

que foi utilizado foi o "F710 Wireless Joystick"da Logitech (ver figura 5.12) e permite manobrar o robô sem necessidade de qualquer cabo associado.

Figura 5.12: *Joystick* utilizado para movimentar o robô

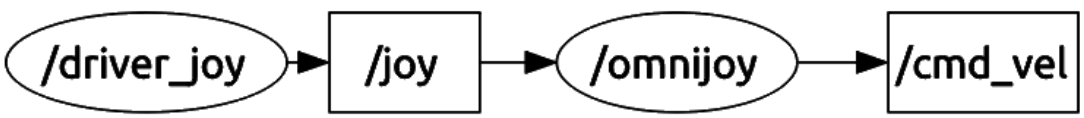


Figura 5.13: Interface do nó Omnijoy (/omnijoy)

A figura 5.13 representa a interface do nó Omnijoy, que subscreve o tópico "/joy" e publica um tópico "/cmd_vel". O tópico subscrito é do tipo "sensor_msgs/Joy"e é publicado pelo nó representado na figura como "/driver_joy". Este nó pertence ao pacote do ROS "joystick_drivers" e efetua a abstração de hardware do comando. O tópico publicado é do tipo "geometry_msgs/Twist" e indica a velocidade pretendida. Este tópico é subscrito pelo nó Driver Discovery Q2 e permite assim a movimentação do robô.

Sempre que o nó "/driver_joy" faz uma publicação com o estado do comando, o nó Omnijoy guarda os valores associados aos botões analógicos e aos botões de interesse. Existe uma função do programa que corre de forma periódica e que é responsável pelo cálculo e publicação do tópico "/cmd_vel".

Existe um botão no *joystick* que é responsável pela ativação do movimento, caso este esteja ativo a velocidade calculada é proporcional ao valor dos botões analógicos, caso contrário o valor é zero. Existem dois botões analógicos, um controla a velocidade linear (V e V_n) e outro controla a velocidade de rotação (ω) do robô. Existe ainda outro botão que permite a ativação de uma função de turbo, isto é, aumenta a relação de proporção entre a velocidade e o valor do controlador direcional. O valor desta proporção é possível controlar através outros dois botões utilizados para aumentar e diminuir este valor. A tabela 5.9 representa os parâmetros configuráveis deste nó.

Tabela 5.9: Parâmetros do nó Omnijoy

Parâmetro	Descrição
axis_linear_x	Eixo do botão analógico que controla a velocidade do robô
axis_linear_y	Eixo do botão analógico que controla a velocidade normal do robô
axis_angular	Eixo do botão analógico que controla a velocidade de rotação do robô
axis_deadman	Botão que ativa o movimento do robô
axis_turbo	Botão que ativa o turbo do robô
axis_turbo_up	Botão que aumenta a relação de proporção no modo turbo para as velocidades lineares
axis_turbo_down	Botão que reduz a relação de proporção no modo turbo para as velocidades lineares
scale_angular	Velocidade de rotação.
scale_linear	Velocidades lineares.
turbo_scale_linear	Velocidades lineares iniciais em modo turbo.
turbo_max_scale_linear	Velocidades lineares máximas em modo turbo.
turbo_scale_angular	Velocidade de rotação em modo turbo.

O *script* de arranque deste nó com as configurações utilizadas encontra-se no anexo A.2.

5.4 Teste do protótipo

Para testar o correto funcionamento do protótipo e do sistema de navegação do robô foi inicialmente utilizado os sensores do robô para mapear o local onde este iria se movimentar, e posteriormente testado o sistema de navegação que inclui o seguidor de caminhos desenvolvido e um sistema de localização baseado em contornos.

5.4.1 Mapeamento

Para efetuar os testes do protótipo foi necessário mapear o espaço de trabalho deste, neste caso o laboratório. Para isso foi utilizado o pacote de ROS "hector_mapping" que nos permitiu construir o mapa com base nas leituras da odometria e do laser. O *script* para arranque deste nó encontra-se no anexo A.3. Para execução do mapeamento é necessário ainda arrancar o nó Driver Discovery Q2, o nó Omnijoy para mover o robô ao longo do espaço e ainda arrancar o nó do laser "hokuyo_node". O *script* de arranque do laser encontra-se no anexo A.4.

A figura 5.14 representa os tópicos e nós ativos durante o mapeamento. O mapa resultante é apresentado na figura 5.15.

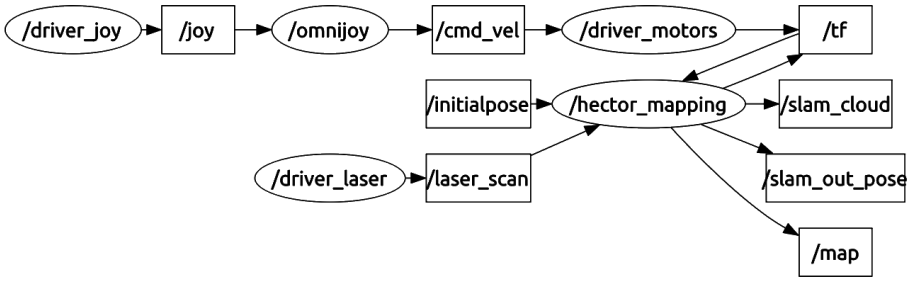


Figura 5.14: Nós e tópicos ativos durante mapeamento

Para que o processo de mapeamento seja efetuado de forma correta é necessário que o processamento da informação sensorial do robô ocorra de forma contínua sem falhas. Neste sentido o resultado do mapeamento, apresentado na figura 5.15, permite-nos concluir o correto funcionamento do robô e dos sistemas desenvolvidos.

5.4.2 Sistema de navegação

Para testar o sistema de navegação foi necessário executar os nós Path_controller e Driver-Discovery Q2, e ainda um outro nó que permitisse a localização do robô. Foi utilizado o nó "localization_perfect_match" para calcular a localização do robô, sendo que este utiliza os dados da odometria e do laser para obter uma pose do robô o mais fidedigna possível em relação ao referencial do mapa. Este nó desenvolvido no INESC TEC implementa um método de localização

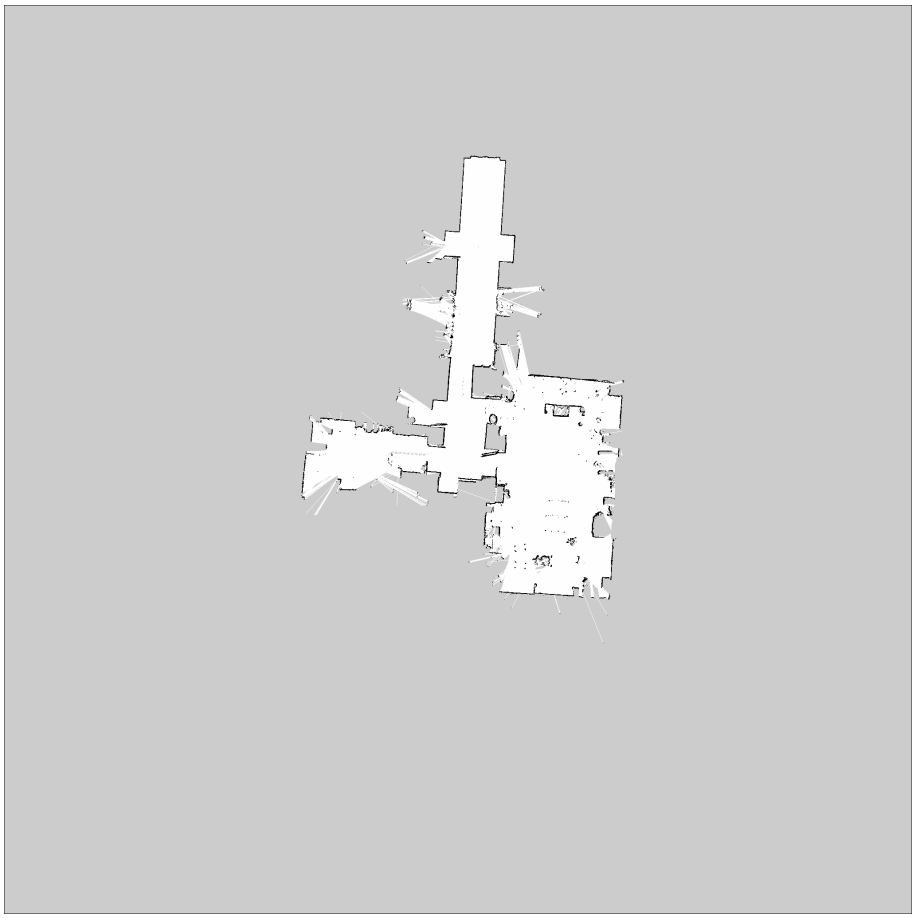


Figura 5.15: Mapa resultante

baseada em contornos. Este nó foi alterado para poder utilizar a odometria de robôs omnidirecionais. A figura 5.16 representa os tópicos e nós ativos durante o teste deste sistema.

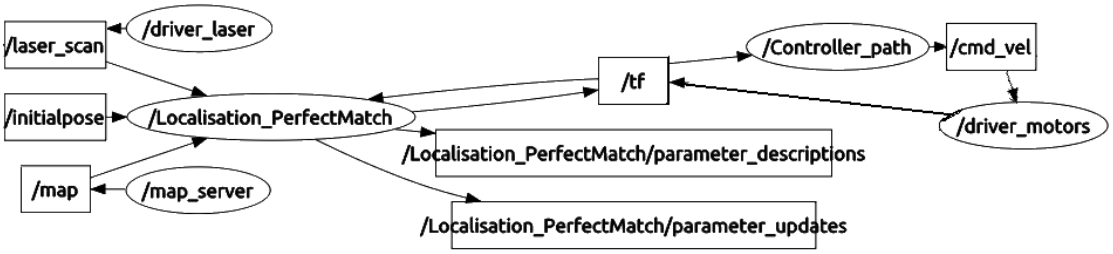


Figura 5.16: Nós e tópicos ativos durante teste do controlador de caminhos

Os testes efetuados apenas pretendem avaliar o desempenho do seguidor de caminhos. Estes testes foram efetuados com o controlador a executar com um período de 50ms.

5.4.2.1 Segmento de reta

A figura 5.17 representa o erro de seguimento de um caminho do tipo segmento de reta com uma velocidade de $\approx 0.35m/s$. O valor máximo deste erro neste percurso é relativamente baixo $\approx 3.1 * 10^{-3}m$.

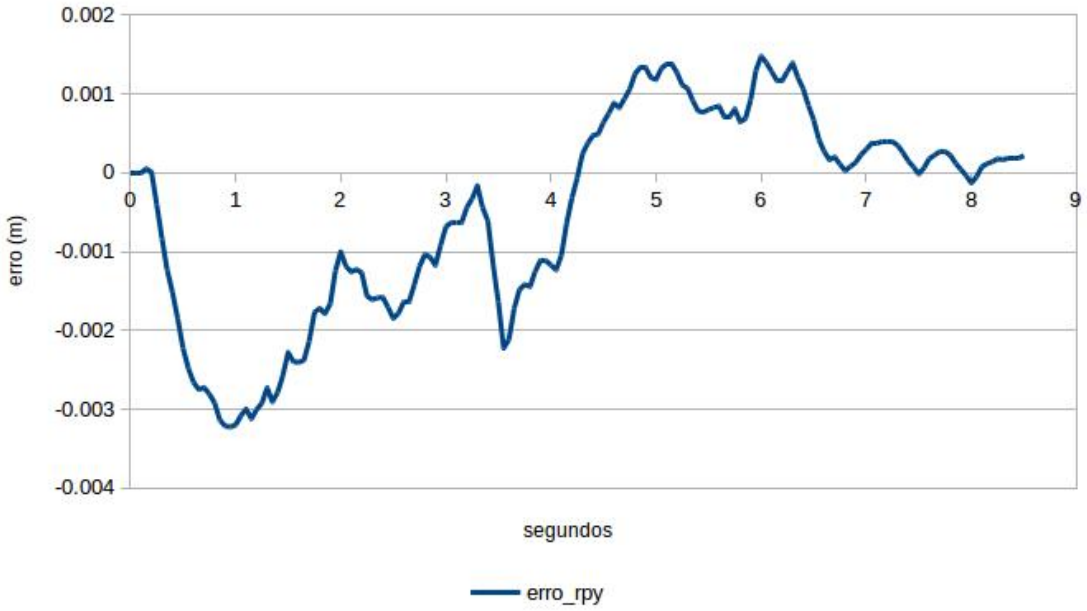


Figura 5.17: Teste 1 - erro rpy

De modo a testar o comportamento do robô quando se desvia do caminho pretendido, foram criados dois caminhos paralelos e durante o percurso do robô foi alterando aleatoriamente o caminho que o robô deveria seguir. Foram efetuados dois testes deste tipo com duas velocidades diferentes. A figura 5.18 representa os caminhos pretendidos e o caminho efetuado pelo robô com uma velocidade de $\approx 0.25m/s$. A figura 5.19 representa o erro ao ponto mais próximo do caminho pretendido. Os dois picos visíveis no gráfico representam a alteração do caminho pretendido e é possível observar que o robô converge para o caminho pretendido quando este é alterado.

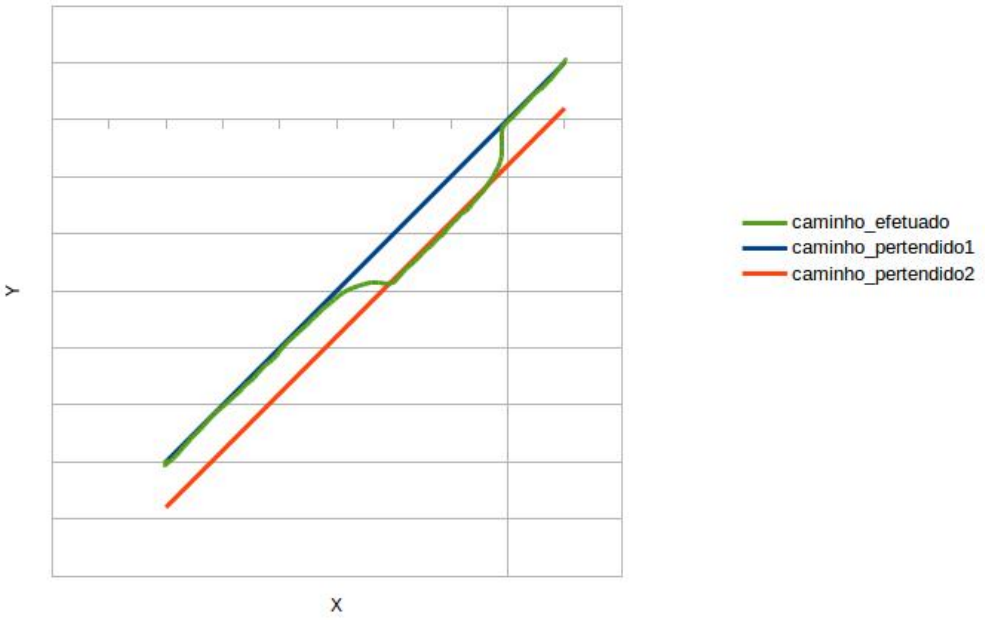


Figura 5.18: Teste 2 - posição x,y

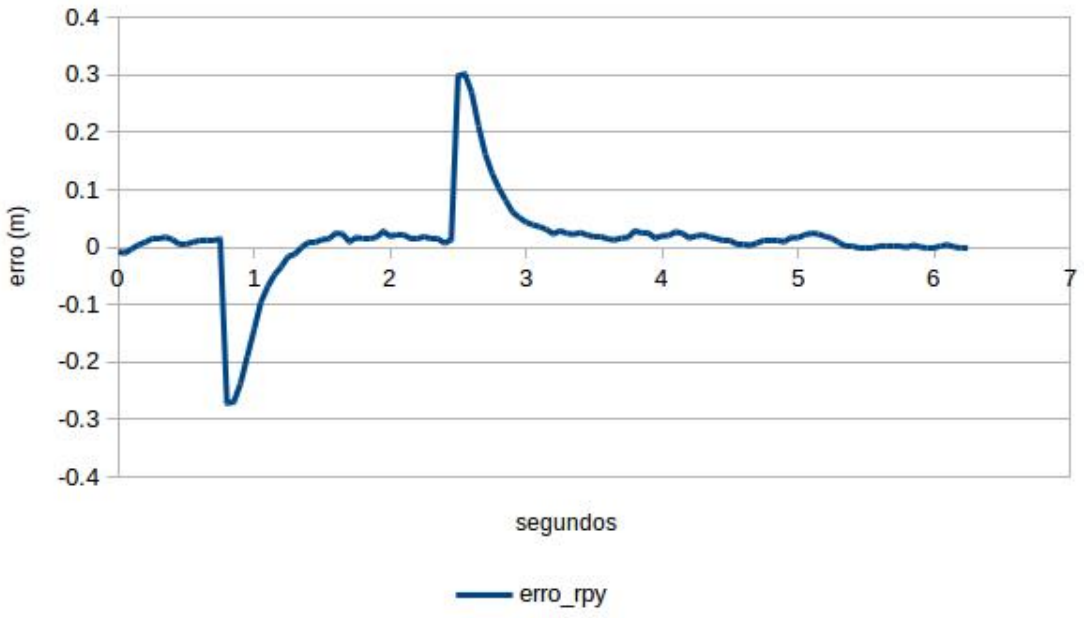


Figura 5.19: Teste 2 - erro rpy

A figura 5.20 representa os caminhos pretendidos e o caminho efetuado pelo robô com uma velocidade de $\approx 0.10 m/s$. A figura 5.21 representa o *errorpy* durante o teste efetuado ao robô.

Em relação ao teste efetuado anteriormente com uma velocidade superior, como era esperado, a velocidade mais baixa o robô têm um erro de seguimento mais baixo.

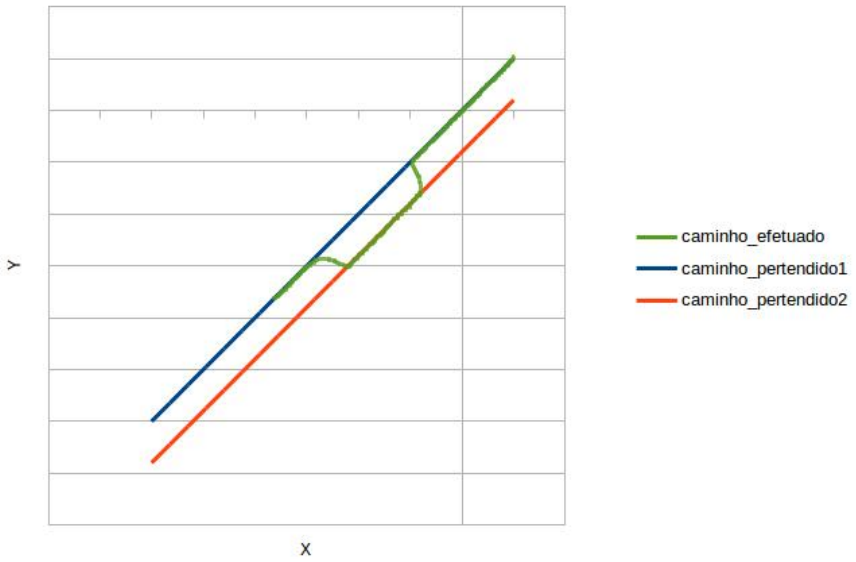


Figura 5.20: Teste 3 - posição x,y

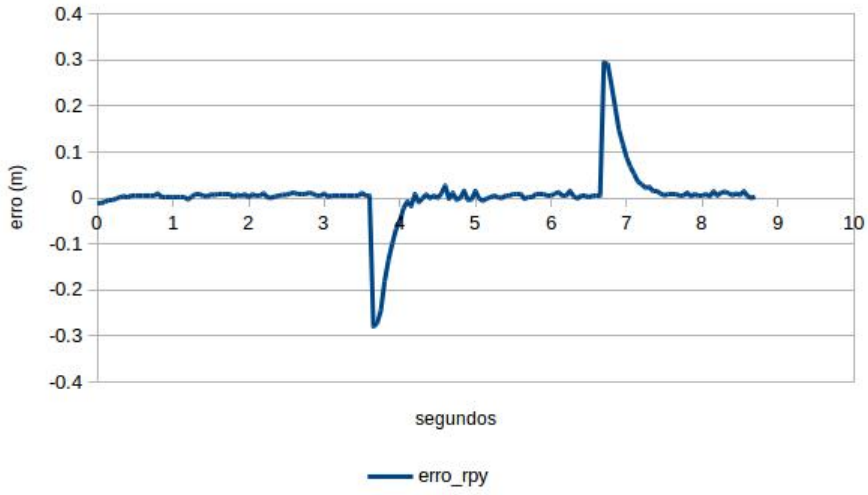


Figura 5.21: Teste 3 - erro rpy

Foi efetuado um teste em relação ao erro de rotação ao longo do seguimento da reta. O teste efetuado consiste na rotação de cerca de 180° enquanto segue um determinado segmento de reta com uma velocidade de $\approx 0.10 m/s$. A figura 5.22 representa o erro de seguimento da reta durante a rotação. A figura 5.23 representa a relação entre o ângulo desejado e o ângulo do robô. É possível observar que o robô converge para o ângulo desejado. A rotação do robô interfere com

o seguimento da reta, tornando o erro de seguimento um pouco alto. Com uma velocidade mais elevada este erro toma valores superiores.

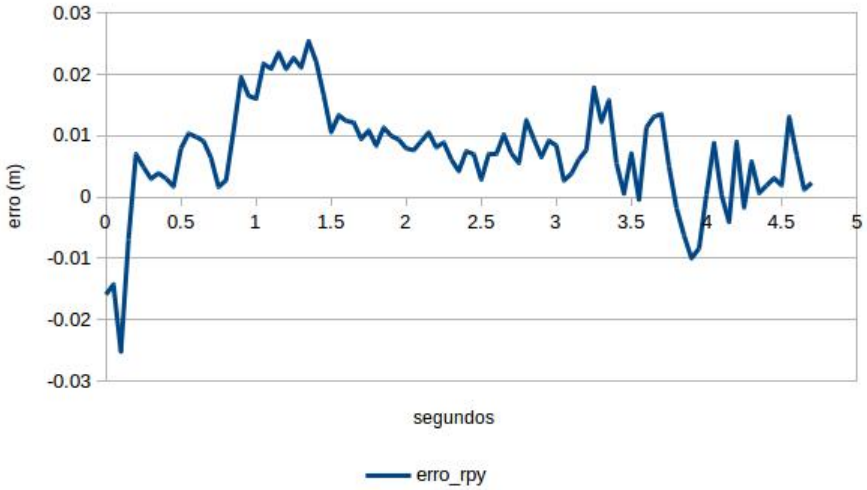


Figura 5.22: Teste 4 - erro rpy

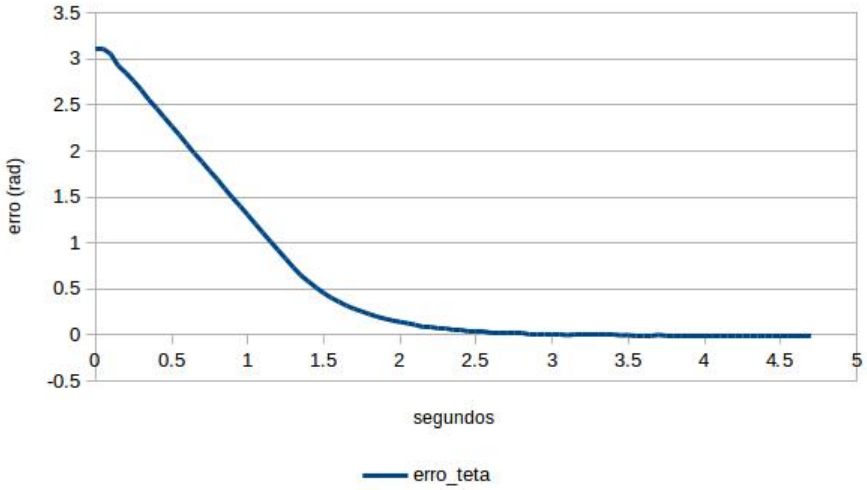


Figura 5.23: Teste 4 - $erro_\theta$

5.4.2.2 Arco de circunferência

A figura 5.25 representa o erro de seguimento de um caminho do tipo arco de circunferência com uma velocidade de $\approx 0.25m/s$. O valor máximo deste erro neste percurso é de $\approx 0.015m$. O valor do erro entre o ângulo desejado e o ângulo atual do robô e tem um valor de $\approx 0.1rad$, e devido à sua variação dinâmica este não converge para 0 ao longo do tempo. Assim foi necessário alterar o controlador inicialmente projetado, e além da componente proporcional que controla a rotação o robô, foi adicionada uma componente integral. A velocidade de rotação do robô é dada

por $W = -Kw * erro_\theta - Ki * integral$, onde $integral = integral + erro_\theta / dt$ (dt é o tempo de ciclo do controlador). A figura 5.27 representa o $erro_\theta$ para o mesmo caminho com a alteração efetuada ao calculo da velocidade de rotação.

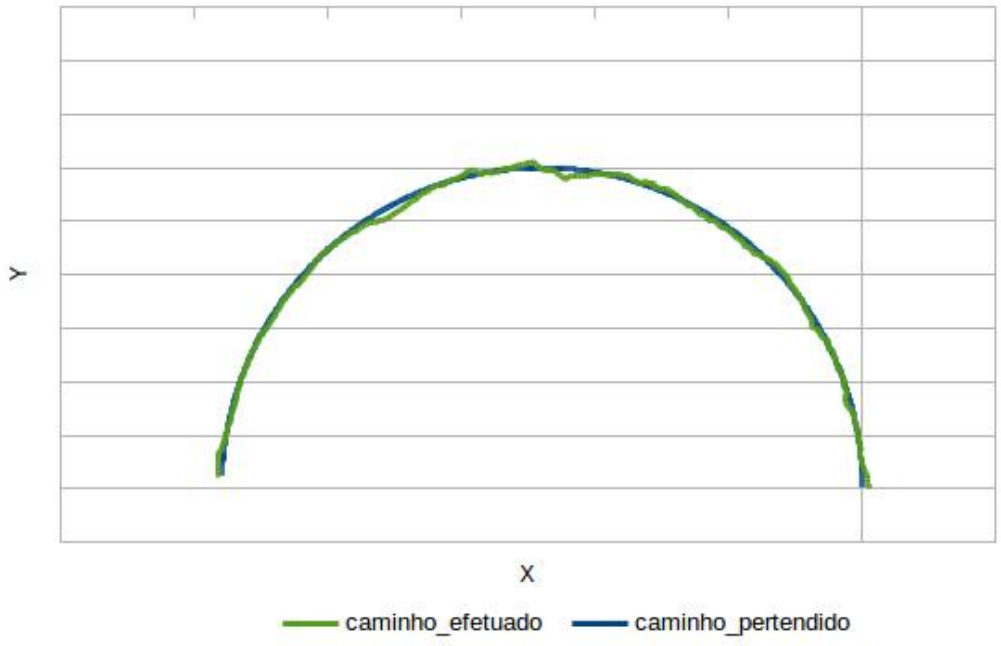


Figura 5.24: Teste 5 - posição x,y

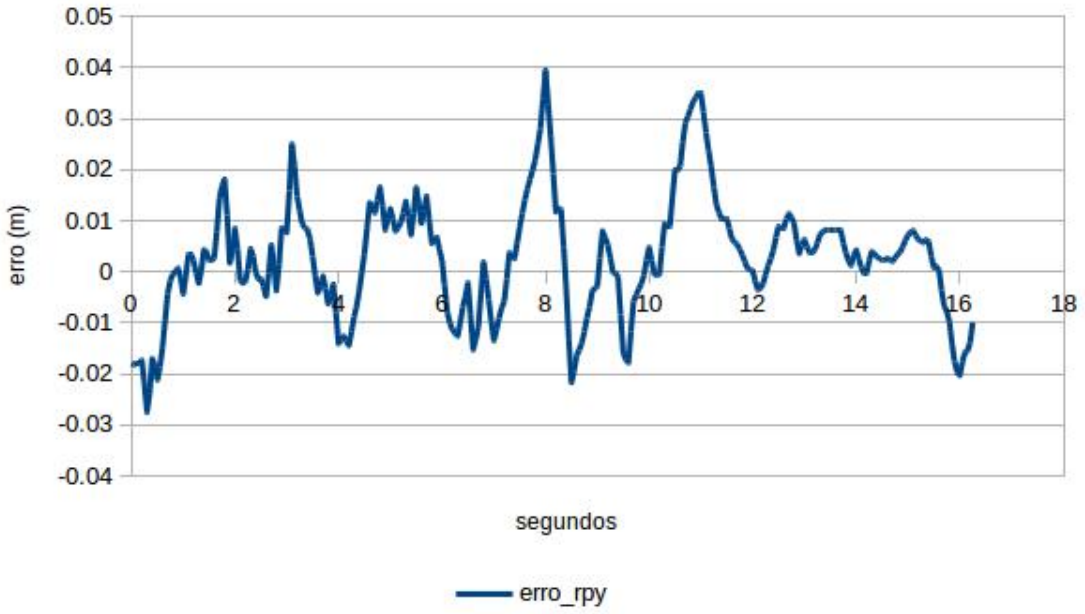


Figura 5.25: Teste 5 - erro rpy

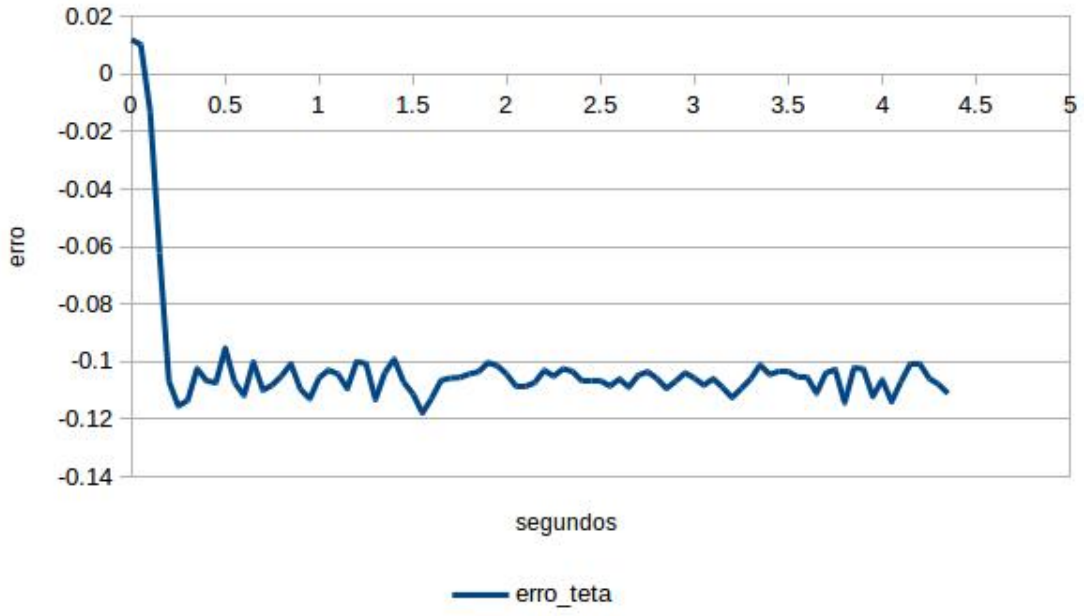


Figura 5.26: Teste 5 - $erro_\theta$

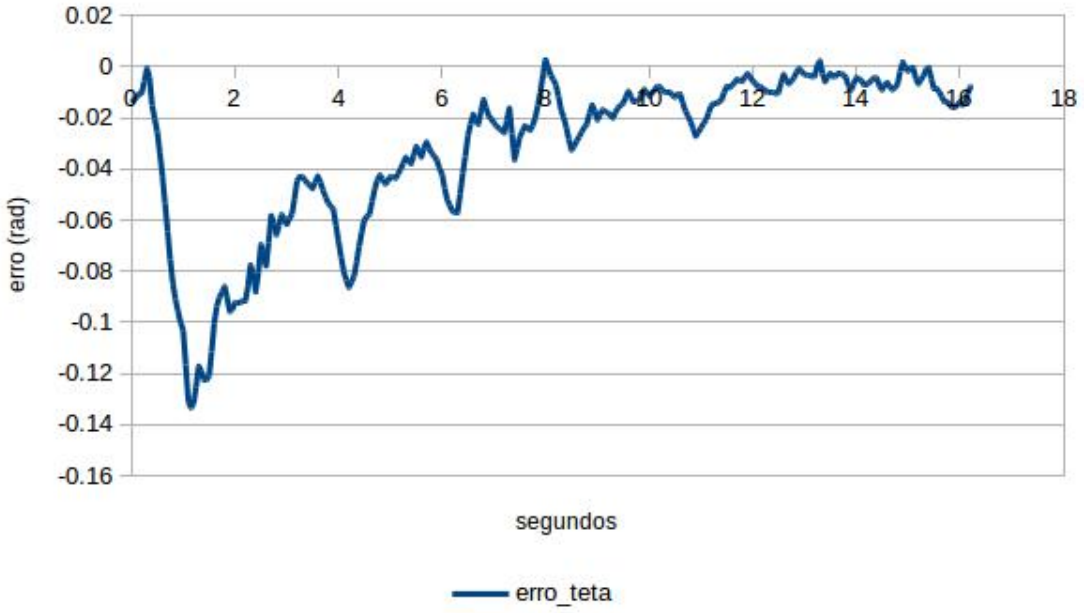


Figura 5.27: Teste 6 - $erro_\theta$

De modo a testar o comportamento do robô quando se desvia do caminho pretendido, foram criados dois arcos com o mesmo centro, mas com raio diferentes e durante o percurso do robô foi alterando aleatoriamente o arco que o robô deveria seguir. Foram efetuados dos testes deste tipo com duas velocidades diferentes. A figura 5.28 representa os arcos pretendidos e o caminho

efetuado pelo robô com uma velocidade de $\approx 0.25m/s$. A figura 5.29 representa o erro ao ponto mais próximo do arco pretendido. Os dois picos visíveis no gráfico representam a alteração do arco pretendido e é possível observar que o robô converge para o arco pretendido quando este é alterado.

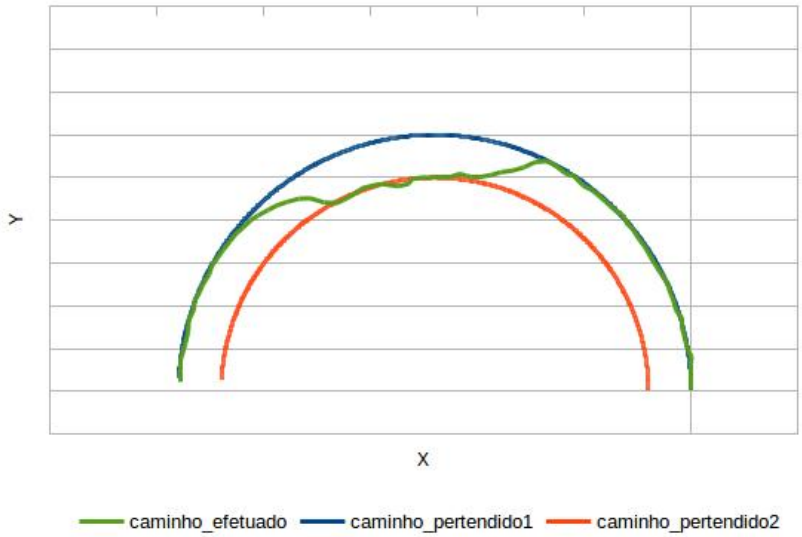


Figura 5.28: Teste 7 - posição x,y

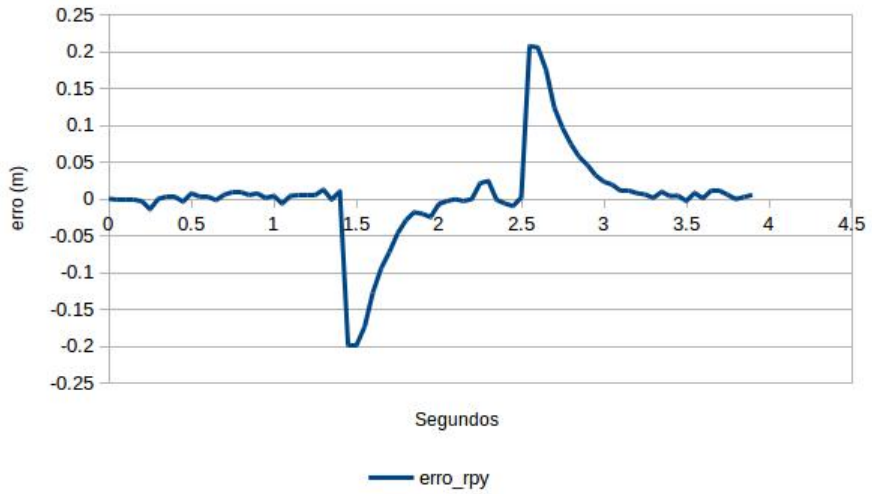


Figura 5.29: Teste 7 - erro rpy

A figura 5.30 representa os arcos pretendidos e o caminho efetuado pelo robô com uma velocidade de $\approx 0.10m/s$. A figura 5.31 representa o $errorpy$ durante o teste efetuado ao robô. Em relação ao teste efetuado anteriormente com uma velocidade superior podemos concluir que a velocidade superior não afeta significativamente o erro de seguimento do arco.

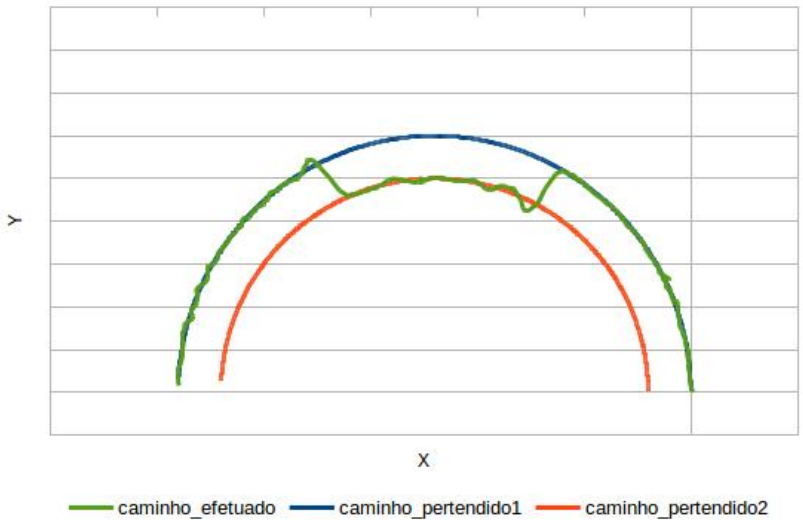


Figura 5.30: Teste 8 - posição x,y

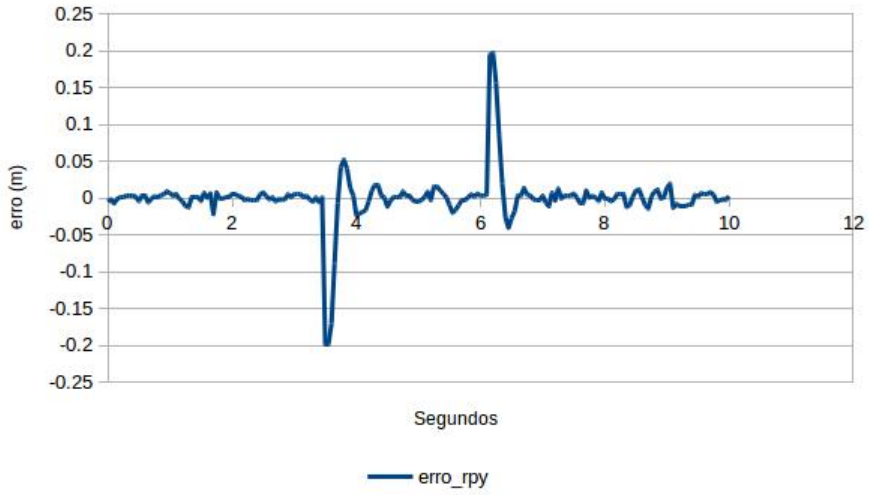


Figura 5.31: Teste 8 - erro rpy

5.4.2.3 Conclusões

Os testes efetuados ao seguidor de caminhos obtiveram resultados satisfatórios em ambiente real. Tendo em conta a precisão do laser utilizado, podemos concluir que apesar de algumas oscilações da localização o robô, o controlador conseguiu obter erros de seguimento dos caminhos pretendidos bastante baixos. A alteração efetuada no controlador do arco relativamente à velocidade de rotação permitiu obter bons resultados quanto à orientação final desejada.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Conclusões

O desenho das rodas Mecanum permite construir uma plataforma verdadeiramente omnidirecional e versátil como demonstrado com este trabalho.

A comunicação com os diversos dispositivos encontra-se muitas vezes associada a protocolos de comunicação e funções dos proprietários, implicando assim a implementação destes protocolos sempre que se pretende comunicar com um novo dispositivo. A aplicação de CANopen como protocolo de comunicação permite-nos obter versatilidade quando à escolha do controlador dos motores. A norma CiA 402 define o perfil e o comportamento funcional dos controladores, sendo assim possível desenvolver um único programa de controlo dos controladores sem preocupações com o fabricante ou funções disponíveis.

O seguidor de caminhos é um elemento fundamental do sistema de navegação de um robô, permitindo que este não se desvie do caminho pretendido efetuando este com o menor erro possível. O facto deste controlador se destinar a plataformas omnidirecionais permitiu uma abordagem simplificada do problema baseada na implementação de um algoritmo que calcula o erro de seguimento do percurso através do ponto mais próximo do caminho pretendido. Os testes a este controlador em simulação obtiveram resultados bastante satisfatórios e com erros muito pequenos.

O protótipo Discovery Q2 permitiu-nos apresentar um robô de rodas Mecanum com capacidade de navegação, isto é, localização e controlo. O trabalho desenvolvido incidiu sobre a abstração de hardware e o cálculo da odometria deste robô. O sistema de localização consistiu na integração de um algoritmo de localização baseado em contornos e a sua fusão com odometria. O controlo é efetuado pelo controlador de caminhos desenvolvido.

6.2 Trabalho Futuro

Como trabalho futuro seria interessante o desenvolvimento de uma interface para o utilizador que permitisse a geração de caminhos, bem como definir uma mensagem em ROS que permitisse

a integração posterior num sistema que geração de caminhos.

Como melhoramentos do controlador de caminhos poderiam ser implementados caminhos mais complexos e também mais específicos quanto à forma como a orientação é controlada. Um outro melhoramento no controlador possível seria na etapa de ajuste das velocidades V e V_n de forma a contemplar a velocidade de rotação, isto é, caso o robô esteja a rodar sobre si próprio durante o movimento linear. Seria pertinente também, no controlador considerar as limitações físicas do robô em termos de velocidade.

Anexo A

Ficheiros de arranque

A.1 Nó Driver Discovery Q2

```
<launch>
  <node pkg="driver_discovery_q2" type="driver_discovery_q2"
  name="driver_motors" output="screen">
    <param name="port" value="/dev/ttyACM0"/>
    <param name="period" value="50"/>
    <param name="wheel_radius" value="0.0508"/>
    <param name="data_time_out" value="1.0"/>
    <param name="enc_pulses_per_revolution" value="3072"/>
    <param name="dist_to_wheel_x" value="0.134"/>
    <param name="dist_to_wheel_y" value="0.134"/>

    <param name="odom_frame_id" value="/odom"/>
    <param name="base_frame_id" value="/base_link"/>
    <param name="laser_x_offset" value="0.1"/>
    <param name="laser_y_offset" value="0.0"/>
    <param name="laser_z_offset" value="0.175"/>
  </node>
</launch>
```

A.2 Nó Omnijoy

```
<launch>
  <node pkg="joy" type="joy_node" name="driver_joy" output="screen">
    <param name="autorepeat_rate" value="10"/>
    <param name="coalesce_interval" value="0.05"/>
```

```
  </node>
  <node pkg="omnijoy" type="omnijoy" name="omnijoy" output="screen">
    <remap from="~/cmd_vel" to="/cmd_vel"/>
    <param name="axis_linear_x" value="0"/>
    <param name="axis_linear_y" value="1"/>
    <param name="axis_angular" value="2"/>
    <param name="axis_deadman" value="4"/>
    <param name="axis_turbo" value="5"/>
    <param name="axis_turbo_up" value="7"/>
    <param name="axis_turbo_down" value="6"/>
    <param name="scale_angular" value="0.15"/>
    <param name="scale_linear" value="0.1"/>
    <param name="turbo_scale_linear" value="0.25"/>
    <param name="turbo_max_scale_linear" value="0.4"/>
    <param name="turbo_scale_angular" value="0.40"/>
  </node>
</launch>
```

A.3 Nó hector_mapping

```
<launch>
  <node pkg="hector_mapping" type="hector_mapping"
  name="hector_mapping" output="screen">
    <remap from="/scan" to="/laser_scan" />
    <param name="base_frame" value="/base_link" />
    <param name="map_frame" value="/map" />
    <param name="odom_frame" value="/odom" />
    <param name="map_resolution" value="0.025"/>
    <param name="map_size" value="2048"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_update_distance_thresh" value="0.2"/>
    <param name="map_update_angle_thresh" value="0.45" />
    <param name="map_pub_period" value="2.0" />
    <param name="map_multi_res_levels" value="3" />
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="laser_min_dist" value="0.1" />
    <param name="laser_max_dist" value="5.0" />
```

```
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />
    <param name="pub_map_odom_transform" value="true"/>
    <param name="output_timing" value="false"/>
    <param name="scan_subscriber_queue_size" value="5"/>
    <param name="pub_map_scanmatch_transform" value="false"/>
    <param name="tf_map_scanmatch_transform_frame_name"
    value="scanmatcher_frame"/>
  </node>
</launch>
```

A.4 Nó hokuyo_node

```
<launch>
  <node pkg="hokuyo_node" type="hokuyo_node"
  name="driver_laser" output="screen">
   <remap from="/scan" to="/laser_scan" />
   <param name="port" type="string" value="/dev/ttyACM1"/>
   <param name="frame_id" type="string" value="laser_link"/>
   <param name="calibrate_time" type="bool" value="true"/>
   <param name="min_ang" type="double" value="-1.75"/>
   <param name="max_ang" type="double" value="1.75"/>
  </node>
</launch>
```

A.5 Simulador

```
<launch>
  <param name="/use_sim_time" value="true"/>

  <node name="discoveryq2_simulator" pkg="stage_ros"
  type="stageros" args="\$(find discoveryq2_simulator_stage)
  /config/discoveryq2.world" respawn="false" output="screen"/>

  <node name="localization_ground_truth" pkg="pose_stamped_to_tf"
  type="pose_stamped_to_tf" respawn="false" output="screen">
    <remap from="/pose_ground_truth_odometry"
    to="/base_pose_ground_truth" />
```

```
12   </node>
13
14   <node name="map_server" pkg="map_server" type="map_server"
15   args="\$(find discoveryq2_simulator_stage)/config/map.yaml">
16     <param name="frame_id" value="map"/>
17   </node>
18 </launch>
```

Anexo B

Código do microcontrolador

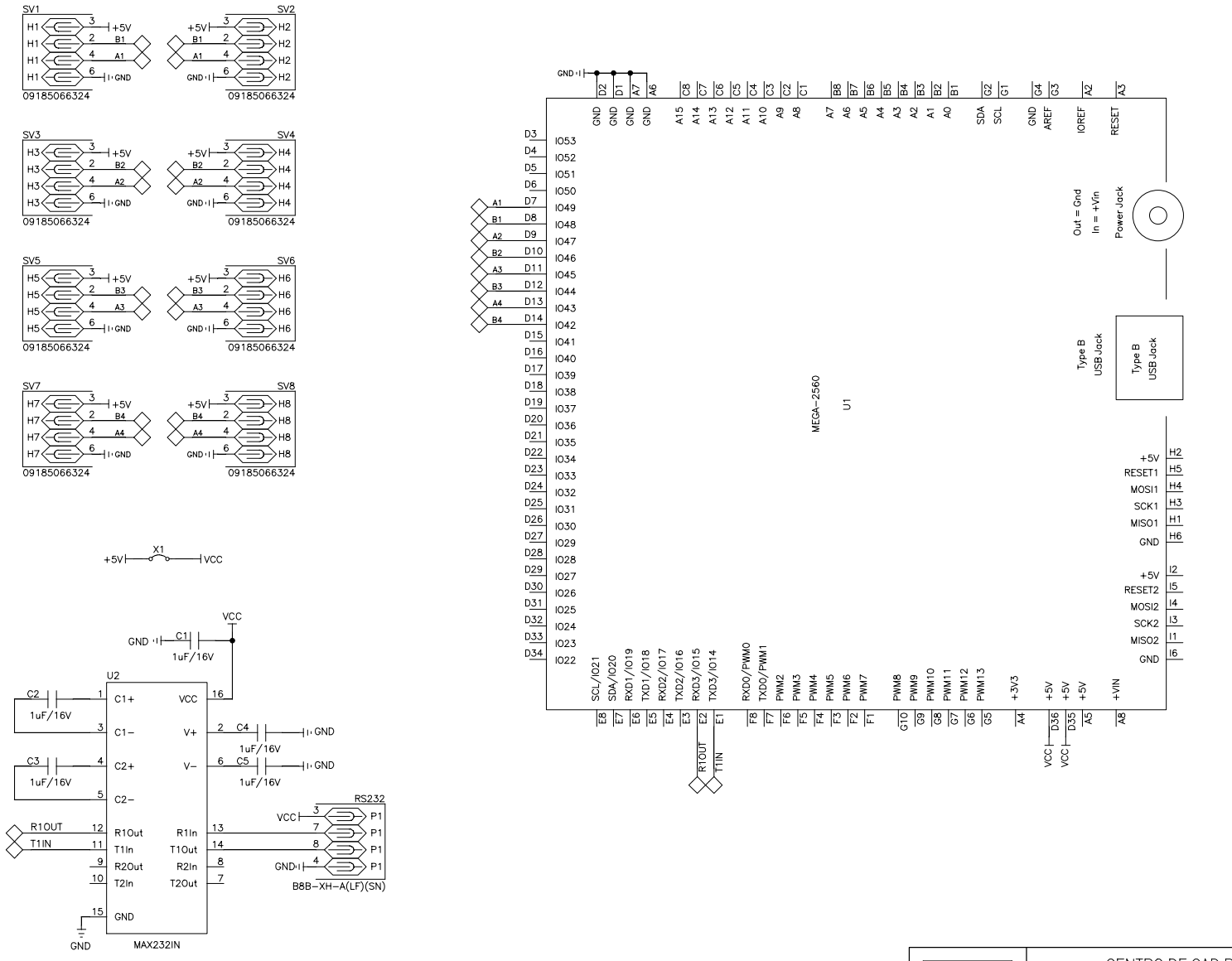
```
// Interface Discovery Q2 Motor Drive
// Code by Fernando SA, June 2016

#include <Arduino.h>
#include <TimerOne.h> //http://www.pjrc.com/teensy/td_libs_TimerOne.html

const int M1_encoderA_pin = 49;  // Motor 1 encoder A input
const int M1_encoderB_pin = 48;  // Motor 1 encoder B input

const int M2_encoderA_pin = 47;  // Motor 2 encoder A input
const int M2_encoderB_pin = 46;  // Motor 2 encoder B input

const int M3_encoderA_pin = 45;  // Motor 3 encoder A input
const int M3_encoderB_pin = 44;  // Motor 3 encoder B input

const int M4_encoderA_pin = 43;  // Motor 4 encoder A input
const int M4_encoderB_pin = 42;  // Motor 4 encoder B input


// Variables
long previousMicros = 0;        // will store last time Odometry was updated
long interval = 50;             // Odometry update period (ms)

byte encoder1_state, encoder2_state, encoder3_state, encoder4_state;
volatile uint16_t encoder1_pos, encoder2_pos, encoder3_pos ,encoder4_pos;
//// outstartchar
//// T - Encoder values
//

void sendOdometry(void)
{
  uint16_t o1, o2 ,o3, o4;
  // Read odometry atomicaly
  cli();
  o1 = encoder1_pos;
  o2 = encoder2_pos;
  o3 = encoder3_pos;
  o4 = encoder4_pos;
  sei();
  // Send it
  Serial.write('T');
  sendint16_t(o1);
  sendint16_t(o2);
  sendint16_t(o3);
  sendint16_t(o4);
}

void timer_interrupt(void)
{
  byte b, new_state;
  static int8_t encoder_table[16] = {0, 1, -1, 0,  -1, 0, 0, 1,  1, 0, 0, -1,  0, -1, 1, 0};
  b = PINL ;

  new_state = (b >> 6) & 0x03; // Put encoder channels in the lowest bits
  encoder1_pos += encoder_table[encoder1_state | new_state];
  encoder1_state = new_state << 2;

  new_state = (b >> 4) & 0x03; // Put encoder channels in the lowest bits
  encoder2_pos += encoder_table[encoder2_state | new_state];
  encoder2_state = new_state << 2;

  new_state = (b >> 2) & 0x03; // Put encoder channels in the lowest bits
  encoder3_pos += encoder_table[encoder3_state | new_state];
  encoder3_state = new_state << 2;

  new_state = (b) & 0x03; // Put encoder channels in the lowest bits
  encoder4_pos += encoder_table[encoder4_state | new_state];
  encoder4_state = new_state << 2;

}


void setup()
{
  previousMicros = micros();

  pinMode(M1_encoderA_pin, INPUT);
  pinMode(M1_encoderB_pin, INPUT);

  pinMode(M2_encoderA_pin, INPUT);
  pinMode(M2_encoderB_pin, INPUT);

  pinMode(M3_encoderA_pin, INPUT);
```

```
  pinMode(M3_encoderB_pin, INPUT);

  pinMode(M4_encoderA_pin, INPUT);
  pinMode(M4_encoderB_pin, INPUT);

  Timer1.attachInterrupt(timer_interrupt);
  Timer1.initialize(200); //uS

  Serial.begin(9600);
  Serial3.begin(9600);
}

void loop()
{

  while(Serial.available() > 0) {
    Serial3.write(Serial.read());
  }

  while(Serial3.available() > 0) {
    Serial.write(Serial3.read());
  }

  unsigned long currentMicros = micros();

  if(currentMicros - previousMicros> interval * 1000) {
    previousMicros = currentMicros;
    sendOdometry();
  }

}
```

```
//  Packed Channels for Serial communication on ATmega168/328
//  Original code by Paulo Costa August 2013
//  This is free software. You can redistribute it and/or modify it under
//  the terms of Creative Commons Attribution 3.0 United States License.
//  To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0
//  or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.


void sendHexNibble(byte b)
{
  if (b < 10) {
    Serial.write('0' + b);
  } else if (b < 16) {
    Serial.write('A' + (b - 10));
  }
}

void sendHexByte(byte b)
{
  sendHexNibble(b >> 4);
  sendHexNibble(b & 0x0F);
}
void sendint16_t(int16_t v)
{
  sendHexByte(v >> 8);
  sendHexByte(v & 0xFF);
}

byte isHexNibble(char c)
{
  if ((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F')) return 1;
  else return 0;
}


byte HexNibbleToByte(char c)
{
  if (c >= '0' && c <= '9') return c - '0';
  else if (c >= 'A' && c <= 'F') return c - 'A' + 10;
  else return 0;
}
```

Anexo C

Esquemático Interface Discovery Q2



CENTRO DE CAD PARA ELECTRONICA – PORTO			
Discovery Q2 Interface 1			
PEDIDO: Fernando Sa			DATA: 2016/04/06
EXECUTADO: Jose Carlos Azevedo			DATA: 2016/04/22
PROJ. No: __.__.__	REQ. No: ___/____	FOLHA: 1 de 1	VERSAO: 1.0

Referências

[1] Dr. Humberto Secchi. Uma Introdução aos Robôs Móveis. Disponível em http://www.obr.org.br/wp-content/uploads/2013/04/Uma_Introducao_aos_Robos_Moveis.pdf, Agosto 2008.

[2] J. Borenstein, H.R. Everett, e L. Feng. *Navigating Mobile Robots: Sensors and Techniques*. A. K. Peters, Ltd., Wellesley, MA, 1996.

[3] H. Yu e A. Dubowsky, S. ands Skwersky. Omni-directional mobility using active split offset castors, Setembro 2000. Proceedings ASME Design Engineering Technical Conferences ,Baltimore.

[4] Victor e Spinu Veaceslav Doroftei, Ioan e Grosu. *Bioinspiration and Robotics Walking and Climbing Robots*, chapter 29, páginas 511–528. I-Tech Education and Publishing, Setembro 2007.

[5] B.E. Ilon. Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base, Abril 8 1975. US Patent 3,876,255. URL: http://www.google.com/patents/US3876255.

[6] Illah R. Siegwart, Roland e Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

[7] Charles P. Muir, Patrick F. e Neuman. Kenematic modeling for feedback control of an omni-directional wheeled mobile robot, 1987. URL: http://www.ri.cmu.edu/pub_files/1991/3/01087767-1.pdf.

[8] Aparna e Bright Glen e Potgieter Johan e Tlale Sylvester Diegel, Olaf e Badve. Improved Mecanum Wheel Design for Omni -directional Robots, Novembro 2002. Proc. 2002 Australasian Conference on Robotics and Automation. URL: http://ftp.imp.fu-berlin.de/pub/Rojas/omniwheel/Diegel-Badve-Bright-Potgieter-Tlale.pdf.

[9] Thomas Koestler, Andreas e Bräunl. Mobile Robot Simulation with Realistic Error Models, Dezembro 2004. 2nd International Conference on Autonomous Robots and Agents. URL: http://www-ist.massey.ac.nz/conferences/icara2004/files/Papers/Paper08_ICARA2004_046_051.pdf.

[10] S.V. e Thakur A.G. e Modak G.S. Wakchaure, K.N. e Bhaskar. Kinematics Modelling of Mecanum Wheeled Mobile Platform. URL: http://www.academia.edu/4557426/KINEMATICS_MODELLING_OF_MECANUM_WHEELED_MOBILE_PLATFORM.

[11] Bing e Ghaeminezhad Nurallah Taheri, Hamid e Qiao. Kinematic Model of a Four Mecanum Wheeled Mobile Robot, Março 2015. International Journal of Computer Applications. URL: http://research.ijcaonline.org/volume113/number3/pxc3901586.pdf.

[12] CAN in Automation. Can knowledge, 2016. URL: http://www.can-cia.org/can-knowledge/.

[13] National Instruments Corporation. The basics of canopen, Agosto 2013. URL: http://www.ni.com/white-paper/14162/en/#toc1.

[14] Márcio e Almeida Sérgio Fernandes, Gustavo e Correia. Sistema de instrumentação distribuído multi-sensorial, Julho 2001. URL: https://web.fe.up.pt/~ee96112/arquivo/relatorio.pdf.

[15] WEG Equipamentos Elétricos. Manual do usuário canopen, Outubro 2010. URL: http://ecatalog.weg.net/files/wegnet/WEG-plc300-comunicacao-canopen-10000849433-manual-portugues-br.pdf.

[16] CAN in Automation. Cia® 402 series: Canopen device profile for drives and motion control, 2016. URL: http://www.can-cia.org/can-knowledge/canopen/cia402/.

[17] Florian Weisshardt. ros_canopen, Visto em 2016. URL: http://wiki.ros.org/ros_canopen?distro=indigo.