Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

David Miguel da Silva Ferreira

Robô Rececionista

Relatório de Projeto

Orientado por:

Doutor Gabriel Pereira Pires, Instituto Politécnico de Tomar

Doutora Ana Cristina Barata Pires Lopes, Instituto Politécnico de Tomar

Projeto apresentado ao Instituto Politécnico de Tomar para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrotécnica

À Diana.

Resumo

Esta dissertação descreve a implementação de um robô guia capaz de efetuar navegação autónoma dentro de um ambiente interior. O seu objetivo principal será navegar autonomamente dentro de um edifício previamente conhecido pelo robô, guiando uma pessoa até um destino escolhido pela mesma através de uma interface tátil que oferece vários destinos pré-definidos.

O projeto firmou-se no estudo e documentação das diversas partes constituintes de um robô existente no mercado, o TurtleBot, e a modificação do mesmo de forma a servir o propósito pretendido, mantendo uma arquitetura modular e utilizando a plataforma de *middleware* ROS (*Robot Operating System*).

O robô desenvolvido é um conjunto de subsistemas, nomeadamente base robótica, sistema de visão e perceção, sistema de navegação e interface homem-máquina. Pode ser facilmente estendido e modificado para servir outros propósitos, nomeadamente para a integração com sistemas de domótica num conceito de vida assistida por ambientes inteligentes (AAL – *ambient assisted living*) dentro da missão a que a unidade de I&D VITA.IPT se propõe. O robô é capaz de navegar autonomamente num ambiente interior, interagindo com o utilizador, apresentando ao mesmo, por exemplo, uma lista de pontos objetivos pré-definidos e *feedback* do estado da navegação.

O robô foi testado em dois cenários reais diferentes com obstáculos dinâmicos. Os resultados experimentais de navegação validaram todos os seus níveis de funcionalidade, mostrando que a plataforma poderá servir de base a várias aplicações em diversos contextos.

Palavras-chave: Robótica móvel, SLAM, Gmapping, localização Monte Carlo, ROS

Abstract

This dissertation describes the implementation of a guide robot capable of autonomous navigation inside an indoor environment. Its main goal will be navigating autonomously inside of a building previously known by the robot, guiding a person to a destination chosen by him/her through a tactile interface that offers several pre-defined destinations.

The project has established itself in the study and documentation of the various constituent parts of an existing commercial robot, TurtleBot, and its modification so that it serves the intended purpose, maintaining a modular architecture and using the middleware platform ROS (Robot Operating System).

The developed robot is an agglomerate of subsystems, namely robotic base, vision and perception system, navigation system and human-machine interface. It can be easily extended and modified to serve other purposes, such as integration with home automation systems in a concept of ambient assisted living (AAL) within the mission that the R&D unit VITA.IPT proposes itself. The robot is capable of navigating autonomously in an indoors environment, interacting with the user, presenting the user with, for example, a list of pre-defined destinations and providing feedback of the navigation status.

The robot was tested in two distinct real scenarios with dynamic obstacles. The navigation experimental results validated all its levels of functionality, showing its potential applicability in several contexts.

Keywords: Mobile robotics, SLAM, Gmapping, Monte Carlo localization, ROS

Agradecimentos

É com muita satisfação que expresso aqui o meu sentido obrigado a todos aqueles que tornaram este projeto realidade.

Antes de mais, um grande obrigado aos meus orientadores Gabriel Pires e Ana Lopes por todo o apoio, motivação e disponibilidade demonstrada.

Gostaria ainda de agradecer ao Jorge Perdigão do ISR-UC pela ajuda preciosa na resolução dos problemas iniciais relacionados com o funcionamento do ROS.

Aos meus colegas Luís Caetano, José Batista e Ângelo Alves pelo encorajamento nas horas de desespero.

A todos os docentes do M2E e ao eng. Pedro Neves do LEE-LAB.IPT pelo apoio dado em dentro e fora das salas de aula.

E sobretudo aos meus pais pelo esforço feito para me proporcionarem todas as oportunidades com que sonhei, e à minha irmã pela amizade, apoio e incentivos que sempre me deu.

Este trabalho teve financiamento do projeto AMS-HMI2012 – *Assisted Mobility by Shared-Control and Advanced Human-Machine Interfaces*, com referência RECI/EEI-AUT/0181/2012, ISR/UC/IPT/APCC, 2013-2015, financiado pela Fundação para Ciência e Tecnologia e programa COMPETE.

Trabalho realizado no âmbito das atividades da Unidade VITA.IPT – Vida Assistida por Ambientes Inteligentes.

Índice

Dedicatória III

Resumo V

Abstract VII

Agradecimentos IX

Índice de Figuras XV

Lista de Abreviaturas e Siglas XIX

1. Introdução 1

1.1. Motivação e contexto 1

1.2. Objetivos 2

1.3. Contribuições chave 3

1.4. Organização da dissertação 3

2. Estado da Arte e conceitos de base 7

2.1. Robôs guia 7

2.1.1. Minerva 7

2.1.2. CompanionAble 8

2.1.3. MOnarCH 9

2.2. Sistemas middleware para robôs 10

2.2.1. *ROS – Robot Operating System* 10

2.2.2. Player 16

2.3. Mapeamento e localização 16

2.3.1. Gmapping 17

2.3.2. Hector 17

2.3.3. Adaptive Monte Carlo Locatization 18

2.4. Navegação de robôs móveis 18

2.4.1. Planeadores globais 18

2.4.2. Planeadores locais 19

2.4.3. Mapas de custo 19

3. Arquitetura do sistema 21

3.1. Descrição da plataforma robótica 22

3.1.1. Base robótica Kobuki 23

3.1.2. *Laser range-finder* Hokuyo UTM-30LX 29

3.1.3. Ecrã tátil 30

3.1.4. *Netbook* ASUS 31

3.1.5. Estrutura do robô 32

3.2. Integração em ROS 37

3.2.1. *Overview* dos pacotes do TurtleBot 37

3.2.2. Modificação dos pacotes do TurtleBot para o robô desenvolvido 43

3.3. Controlador de alto nível do robô 45

3.4. Interface gráfica 46

4. SLAM 49

4.1. GMapping 52

4.2. Localização de Monte Carlo 54

4.3. Modelo cinemático, odometria e laser 55

4.4. Implementação em ROS e parâmetros de configuração 58

5. Sistema de navegação 61

5.1. Arquitetura de alto nível do sistema navegação do robô 62

5.2. Arquitetura do pacote `move_base` 63

5.3. Planeador global utilizado: Dijkstra 65

5.4. Planeador local utilizado: DWA 66

5.5. Mapas de custo 67

5.6. Implementação em ROS e parâmetros de configuração 68

6. Interface gráfica e controlador de alto nível 71
7. Testes experimentais e resultados 75
7.1. Testes de SLAM 80
7.2. Testes de navegação 89
7.3. Testes adicionais 92
8. Conclusões e trabalho futuro 95
9. Referências Bibliográficas 97
10. Anexos 101
10.1. Anexo 1 – Enunciado do Projeto 101
10.2. Anexo 2 – Mini-guia de utilização do robô 102

Índice de Figuras

Figura 1 – Robô Minerva 8
Figura 2 – Robô CompanionAble 9
Figura 3 – Robô MOnarCH 10
Figura 4 – Símbolo ROS 11
Figura 5 – Robô Atlas 4 da "Team HKU" do *DARPA Robotics Challenge* 12
Figura 6 – Grafo de computação do ROS 15
Figura 7 – Funcionamento das comunicações no ROS 15
Figura 8 – Arquitetura global do sistema 21
Figura 9 – TurtleBot 22
Figura 10 – Robô rececionista: vista do utilizador e vista lateral 23
Figura 11 – Bases robóticas: Kobuki à esquerda, iClebo POP à direita 24
Figura 12 – Vista de baixo do Kobuki: rodas, tampa da bateria, conector de carregamento e *switch* ON/OFF 24
Figura 13 – Vista de cima da base Kobuki: conectores DB25, USB e de alimentação, LEDs, botões e bumper 25
Figura 14 – *Pinout* do conector DB25 26
Figura 15 – Bateria 4S2P ligada dentro do compartimento da bateria do Kobuki 27
Figura 16 – Placa de alimentação 28
Figura 17 – Esquemático da placa de alimentação 29
Figura 18 – Hokuyo UTM-30LX: o *laser range-finder* utilizado no robô 29
Figura 19 – Ecrã de toque FDT LP104X0114-FNR utilizado no robô 31
Figura 20 – *Netbook* ASUS F200MA-BING-KX387B 31
Figura 21 – Vista de cima da placa base da estrutura 32
Figura 22 – Placa base da estrutura no fim da maquinação 33
Figura 23 – Vista de topo dos apoios do *netbook* 34
Figura 24 – Placas unidas com varão roscado M4 com o tubo de PVC e o LRF 34
Figura 25 – Placas unidas com varão roscado M4 com o tubo de PVC, o LRF e os apoios para o netbook 35
Figura 26 – Modelo 3D da caixa do ecrã tátil 36

Figura 27 – Aspeto final do robô depois de totalmente montado 36
Figura 28 – Lista dos pacotes existentes do TurtleBot através de uma ligação de terminal remoto..... 38
Figura 29 – Lista de pacotes do TurtleBot 39
Figura 30 – Nó de teleoperação utilizando o teclado do TurtleBot 42
Figura 31 – Visualização do mapa do piso 1 dos edifícios G a L e mapas de custo do robô utilizando o RViz 43
Figura 32 – Descrição das juntas relativas ao LRF no URDF do robô 44
Figura 33 – Pasta `laser/` do pacote `turtlebot_navigation` 45
Figura 34 – Algoritmo de controlo de alto nível 46
Figura 35 – Ecrã principal da interface gráfica 47
Figura 36 – Ecrã exibido durante a navegação do robô mostrando o botão de paragem de emergência 48
Figura 37 – Ecrã mostrado ao utilizador durante o carregamento do robô 48
Figura 38 – Exemplo de um mapa métrico discreto (esquerda) e de um mapa topológico (direita) 50
Figura 39 – Exemplo de um mapa aumentado com informação semântica 51
Figura 40 – Módulos fundamentais de um robô móvel, evidenciando o SLAM e suas dependências 52
Figura 41 – Interpretação geometrica de SLAM: são efetuadas as medições z da distância e orientação das referências m relativamente às posições x 54
Figura 42 – Esquema de um robô diferencial 55
Figura 43 – Algoritmo do modelo de campo de verosimilhança [27] 57
Figura 44 – Ficheiro com os parâmetros do gmapping 59
Figura 45 – Parâmetros do módulo amcl 60
Figura 46 – Arquitetura de navegação do robô e as camadas do mesmo 62
Figura 47 – Visão de alto nível do pacote `move_base` 63
Figura 48 – Comportamentos de recuperação do robô por defeito do pacote `move_base` 64
Figura 49 – Relação entre o custo de célula com a footprint de um robô 68
Figura 50 – Ficheiros de configuração do pacote `turtlebot_navigation` 69
Figura 51 – Implementação do controlador e da interface gráfica e troca de informação entre si 72

Figura 52 – Pormenor do código desenvolvido para enviar os comandos do utilizador para o controlador 73
Figura 53 – Planta do piso térreo do edifício 76
Figura 54 – Planta do primeiro píso do edifício 77
Figura 55 – Pormenor dos átrios dos blocos H a J 78
Figura 56 – Pormenor da planta do piso térreo do bloco I 79
Figura 57 – Robô com a estrutura original do TurtleBot e o LRF montado na plataforma mais elevada 81
Figura 58 – Mapa resultante do primeiro mapeamento do edifício I com os problemas assinalados 82
Figura 59 – Mapa do edifício I corrigido 83
Figura 60 – Mapa do piso térreo do edifício 85
Figura 61 – Mapa do primeiro piso do edifício 87
Figura 62 – Pormenor do mapa na zona de cadeiras no átrio do edifício I e mapa de custos com insuflação 91
Figura 63 – Configuração do robô durante a demonstração na competição de robótica GreenT 92
Figura 64 – Visualização do mapa criado durante a competição de robótica "GreenT" no RViz 93

Lista de Abreviaturas e Siglas

AAL – *Ambient assisted living*

AP – *Wireless Access Point* (ponto de acesso sem fios)

API – *Application Programming Interface* (Interface de programação de aplicações)

CNC – Controlo numérico computorizado

DWA – *Dynamic window approach* (abordagem de janela dinâmica)

GIL – *Global interpreter lock*

IC – *Integrated Circuit* (Circuito integrado)

IEEE – *Institute of Electrical and Electronics Engineers*

ISO – *International Organization for Standardization*

LRF – *Laser Rangefinder*

P2P – *Peer-to-peer*

PCB – *Printed circuit board* (placa de circuito impresso)

PGM – *Portable greymap format*

ROS – *Robot Operating System*

RPC – *Remote procedure call* (chamada de procedimento remoto)

TCP/IP – *Transmission Control Protocol/Internet Protocol*

USB – *Universal Serial Bus*

YAML – *YAML Ain't Markup Language*

1. Introdução

Este capítulo apresenta os pontos de vista e motivações que levaram a este trabalho, assim como os seus objetivos principais e contribuições-chave.

1.1. Motivação e contexto

Durante os últimos anos, diversos estudos realizados na Europa e outros países do mundo ocidental têm mostrado uma tendência de envelhecimento e desertificação do território. Este aumento no número de população idosa irá inevitavelmente pressionar a área de cuidados de idosos para inovar e encontrar novas soluções robustas para acompanhar estes indivíduos, seja através de uma assistência a tempo inteiro ou através de tecnologias para manter o maior grau de autonomia possível dos mesmos. Isto leva-nos para um mundo onde robôs e sistemas inteligentes terão um papel cada vez mais importante em integrar e assistir indivíduos com dificuldades, sejam elas motoras, cognitivas ou outras, independentemente da idade dos mesmos. Por outro lado, vemos que existe cada vez mais tecnologia envolvida no nosso quotidiano. São exemplos disso os veículos com tecnologias de condução assistida que evitam acidentes através de análise de ângulos mortos, deteção de peões e outros obstáculos dinâmicos que ponham em risco a segurança dos ocupantes de veículos. Os automóveis incorporam ainda sistemas que aumentam o nível de conforto e segurança dos ocupantes através de modos de pilotagem automática recorrendo a toda uma panóplia de sensores como sistemas de radar e sistemas de comunicação inter-veículos para um planeamento de navegação evitando congestionamentos, vias com maiores índices de acidente ou zonas de mau tempo.

Os robôs de acompanhamento e interação social começam a ter uma expressão real e podem vir a desempenhar um papel fundamental na terapêutica de doentes, seja através da assistência na locomoção, lembrando da toma de medicação ou interagindo com o utilizador tendo em vista a não degeneração cognitiva do mesmo. É ainda interessante a ideia de que estes robôs se encontrem integrado com sistemas de domótica num conceito

de *ambient assisted living* [1], tendo em vista o aumento de autonomia de idosos na sua residência e os benefícios a nível psicológico que daí advêm.

Atualmente, devido à grande complexidade de um sistema deste género, a maior parte dos projetos de robótica, em especial na área da robótica móvel, são implementados utilizando plataformas de *software* e topologias de *hardware* já provadas, sendo nos últimos anos utilizado no âmbito de robôs móveis o sistema de *middleware* ROS (Robot Operating System) graças à grande quantidade de componentes já desenvolvidos, testados e documentados existentes. Dentro do IPT, as últimas plataformas robóticas modulares desenvolvidas encontravam-se demasiado focadas numa modularidade de baixo nível, negligenciando o nível elevado de abstração necessário para o desenvolvimento de projetos mais complexos na área da robótica.

1.2. Objetivos

Este projeto tem por objetivo desenvolver um "robô rececionista", desenhado para operar em ambientes interiores. Pretende-se que o mesmo seja desenvolvido utilizando um sistema modular com capacidades de perceção e navegação autónoma em ambientes interiores, devendo o mesmo receber um ponto objetivo do utilizador e procurar a melhor rota para o atingir, evitando obstáculos. A definição do ponto objetivo pelo utilizador deverá ser feita através de um interface tátil, onde deverão ser apresentados botões com destinos pré-definidos e feito o feedback para o utilizador do progresso da tarefa de navegação, permitindo sempre que o utilizador faça uma paragem de emergência.

A razão pela qual se pretende que o sistema seja modular prende-se com o objetivo de que este sistema sirva de base a várias aplicações, nomeadamente para esta aplicação específica de robô rececionista em que o robô deverá simplesmente guiar o utilizador até ao seu ponto objetivo e para o desenvolvimento futuro de um robô de acompanhamento num contexto AAL, onde poderá servir de assistente a um doente como descrito no ponto 1.1. .

Existe ainda o objetivo que este trabalho sirva como referência para futuros projetos na área da robótica móvel no IPT, pelo que se pretende que a presente tese se apresente

como sendo de fácil leitura e compreensão, servindo a mesma aos alunos da licenciatura em Engenharia Eletrotécnica e de Computadores e dos cursos associados como um guia para iniciação à robótica móvel e plataforma ROS.

1.3. Contribuições chave

A contribuição chave desta dissertação é o desenvolvimento e implementação de um robô de navegação autónoma funcional que seja capaz de navegar autonomamente até um ponto objetivo designado pelo utilizador, sendo ainda capaz de se obstáculos dinâmicos. Para implementar este robô, foram analisados trabalhos de outros autores, e para obter um sistema funcional foi necessário implementar e modificar módulos de *software*, tendo sempre o cuidado de os analisar e documentar para referência em projetos futuros.

Sendo a robótica móvel uma área em constante desenvolvimento, é imperativo que dentro do IPT seja criado *know-how* na área da robótica móvel, não só criando uma plataforma de desenvolvimento modular assente em *software* e *hardware* testado, mas também criando documentação que permita o desenvolvimento futuro de projetos mais ambiciosos que possam ajudar o IPT nos seus objetivos de providenciar formação em tecnologias reconhecidas pela industria e de criação de valor nas empresas da região através da investigação e desenvolvimento de produtos destinados ao mercado, sendo este projeto a minha contribuição para esta missão do IPT.

1.4. Organização da dissertação

A estrutura desta dissertação encontra-se organizada da seguinte maneira:

- **Introdução** – Descrição introdutória do projeto desenvolvido, fazendo um enquadramento do mesmo e explicando a visão global e a motivação que levou à sua criação;

- **Estado da Arte e conceitos de base** – Introdução que se destina a documentar o que está a ser feito atualmente no campo da robótica móvel, mais especificamente no âmbito deste projeto, os robôs guia, assim como munir o leitor de conceitos base necessários para a compreensão do trabalho desenvolvido;

- **Arquitetura do sistema** – Neste capítulo é apresentada a arquitetura geral do robô desenvolvido e é descrita superficialmente cada uma das partes constituintes do mesmo;

- **SLAM** – Neste capítulo é apresentado o problema de localização e mapeamento simultâneos, o método utilizado para sua solução e a sua explicação e contextualização;

- **Sistema de navegação** – Neste capítulo é apresentada e explicada a arquitetura do sistema de navegação e dos seus componentes, assim como a sua implementação;

- **Interface gráfica e controlador de alto nível** – Neste capítulo é apresentada a interface gráfica desenvolvida, assim como o controlador de alto nível do robô, é explicada a lógica de cada uma das partes e detalhada a sua relação e implementação;

- **Testes experimentais e resultados** – Neste capítulo são contextualizados e descritos os testes experimentais realizados, as ilações tiradas dos mesmos e a possíveis metodologias para resolução de problemas encontrados;

- **Conclusões e trabalho futuro** – Apreciação final do projeto e apresentação de possível trabalho futuro;

- **Referências bibliográficas** – Secção onde estão referenciadas todas as fontes utilizadas como material de apoio no desenvolvimento do projeto;

- **Anexos** – Secção onde todos os documentos que complementam o relatório se encontram, nomeadamente um manual.

2. Estado da Arte e conceitos de base

Um robô móvel é uma máquina automática que é capaz de movimentar de forma autónoma, podendo navegar dentro de um determinado espaço controlado, ou comando por teleoperação por um utilizador. Os robôs móveis são cada vez mais comuns em ambientes comerciais e industriais, onde são utilizados por exemplo para o transporte de produtos das linhas de produção para armazenagem. Isto torna-os uma área interessante para a investigação, uma vez que têm inúmeras aplicações, como é exemplo o robô guia desenvolvido neste projeto.

2.1. Robôs guia

Ao longo das últimas duas décadas foram-se feitos diversos avanços na área da robótica, tendo sido desenvolvidos vários robôs que guiam pessoas em diversos contextos, nomeadamente nas áreas do entretenimento e de assistência a pessoas Nos próximos subcapítulos irão ser mostrados alguns exemplos de robôs guia que refletem aplicações nos contexto mencionados acima, através dos quais conseguimos ter perceção da evolução desta área da robótica.

2.1.1. Minerva

O robô Minerva é um robô móvel desenhado para educar e entreter pessoas em espaços públicos pela Universidade de Carnegie Mellon [2]. O robô teve como propósito guiar pessoas através de um museu (Museu Nacional de História Americana do Smithsonian) nos Estados Unidos da América, o que aconteceu durante o verão de 1998. Este robô navegava em ambientes dinâmicos reais não alterados, no meio de uma multidão [3].

Figura 1 – Robô Minerva

2.1.2. CompanionAble

O robô CompanionAble é um robô enquadrado num sistema de *ambient assisted living* (AAL) desenvolvido por um consórcio europeu liderado pela Universidade de Reading do Reino Unido. Este projeto financiado em 7.8 milhões de Euro pela União Europeia iniciado em 2008 teve como objetivo desenvolver um robô de companhia para auxiliar os idosos a viverem em casa de forma (semi-)independente durante mais tempo do que o possível tradicionalmente, providenciando estimulação cognitiva e gestão de tratamentos do idoso, colaborando com um sistema domótico inteligente [4]. Este robô de companhia trabalha de forma colaborativa com uma casa inteligente para disponibilizar planeamento e gestão inteligente de tarefas do dia-a-dia através de listas de tarefas, uma agenda com lembretes conscientes do contexto para consultas médias e medicação, dando os bons dias ao utilizador e cumprimenta-lo quando volta a casa, lembrando o utilizador de coisas importantes antes de sair de casa, treino cognitivo, análise de emoções, álbuns de música e fotografias, lembretes de segurança e prevenção de situações perigosas, controlo inteligente da residência, deteção de quedas, entre outras [5].

Figura 2 – Robô CompanionAble

2.1.3. MOnarCH

O projeto MOnarCH é um projeto liderado pelo IST-ID focado na robótica social e sistemas de sensores interligados em rede para interação com crianças, através de atividades educativas e de entretenimento na enfermaria pediátrica do Instituto Português de Oncologia de Lisboa (IPOL) [6]. Apesar de este projeto estar focado nas interações sociais entre seres humanos e robôs, os robôs desenvolvidos no âmbito deste projeto são robôs móveis e podem ser utilizados como robôs guia, tendo todo o *hardware* necessário para realizar navegação autónoma [7]. Um dos objetivos é combater o isolamento das crianças dentro do IPOL e estimular a interação de grupo, assim como servir de auxiliar nas salas de aula. Uma das formas de estimular a interação de grupo será feita através da projeção de jogos no chão para duas ou mais crianças jogarem com o robô.

Figura 3 – Robô MOnarCH

2.2. Sistemas middleware para robôs

Os sistemas de *middleware* de um robô é uma camada de abstração que reside entre o sistema operativo e as aplicações de *software* [8]. É desenhada para gerir a heterogeneidade de *hardware*, simplificar o desenho de software e reduzir a dificuldade de desenvolvimento e os custos associados ao mesmo, possibilitando ao programador preocupar-se com a lógica e a algoritmia como um módulo independente da plataforma onde vai ser posto em funcionamento, criando assim modularidade num robô. Existem vários sistemas de *middleware* para robôs utilizados na área da robótica móvel, e serão apresentados alguns nas secções seguintes.

2.2.1. *ROS – Robot Operating System*

O *Robot Operating System* (ROS) é uma plataforma que agrega *frameworks* para o desenvolvimento de *software* para robôs, oferecendo serviços que permitem abstração de software, controlo de baixo nível de dispositivos, implementação de funcionalidades de utilização comum, comunicação entre processos e gestão de pacotes [9]. Adicionalmente,

disponibiliza ainda ferramentas e bibliotecas para obter, compilar, escrever e correr código em múltiplos computadores.

Podemos descrever o ROS como um *meta-operating system* que corre em cima de outro (atualmente apenas são suportados sistemas baseados em Unix), tipicamente distribuições GNU/Linux como o Ubuntu, através de processos nesse sistema operativo. Os processos podem correr em várias máquinas simultaneamente, e comunicam num género de rede *peer-to-peer* (P2P) centralizada, utilizando diversos estilos de comunicação como comunicações do tipo RPC através de serviços, transmissão de dados através de tópicos de forma assíncrona e armazenamento de dados num servidor de parâmetros. Devido a esta versatilidade, o ROS é uma plataforma *realtime*, sendo possível no entanto integrá-la com plataformas de tempo real.

Figura 4 – Símbolo ROS

O ROS pretende ser uma plataforma de *software* que fomenta a reutilização de código no desenvolvimento e investigação na área da robótica, sendo efetivamente uma *framework* de processos distribuídos (nós) que são desenhados individualmente e de forma a abstraírem-se o mais possível dos outros, podendo esses processos ser agrupados em pacotes que podem ser facilmente partilhados e distribuídos. Esta ideologia transparece também na comunidade ligada ao ROS, permitindo decisões independentes sobre o desenvolvimento mas ainda assim sendo possível a conjugação de diversos módulos com as ferramentas de infraestrutura do ROS.

De forma a suportar este objetivo principal do ROS de partilha e colaboração, existem objetivos secundários. Um dos objetivos perseguidos é a leveza da *framework*, sendo implementada como um complemento do *software* existente, tentando assim evitar o desenvolvimento de bibliotecas específicas para o ROS, e por outro lado fomentar o desenvolvimento de bibliotecas que não dependem do ROS. Desta forma, é possível criar interfaces limpas e funcionais que poderão ser utilizadas para integrar essas bibliotecas no ROS, como é feito por exemplo no pacote `gmapping`. Ter este objetivo em mente permite que possamos ter independência da linguagem de programação utilizada, que é outro objetivo do ROS. Pode-se assim desenvolver módulos que têm requisitos temporais e computacionais exigentes usando linguagens de baixo nível como C++, e ao mesmo tempo desenvolver *software* de nível mais alto em linguagens que permitam um nível de produtividade do programador mais elevado como Python. O último objetivo secundário passa pela escalabilidade, sendo o ROS apropriado para sistemas de grande complexidade e dimensão, como os robôs envolvidos no *DARPA Robotics Challenge* [10] (Figura 5).

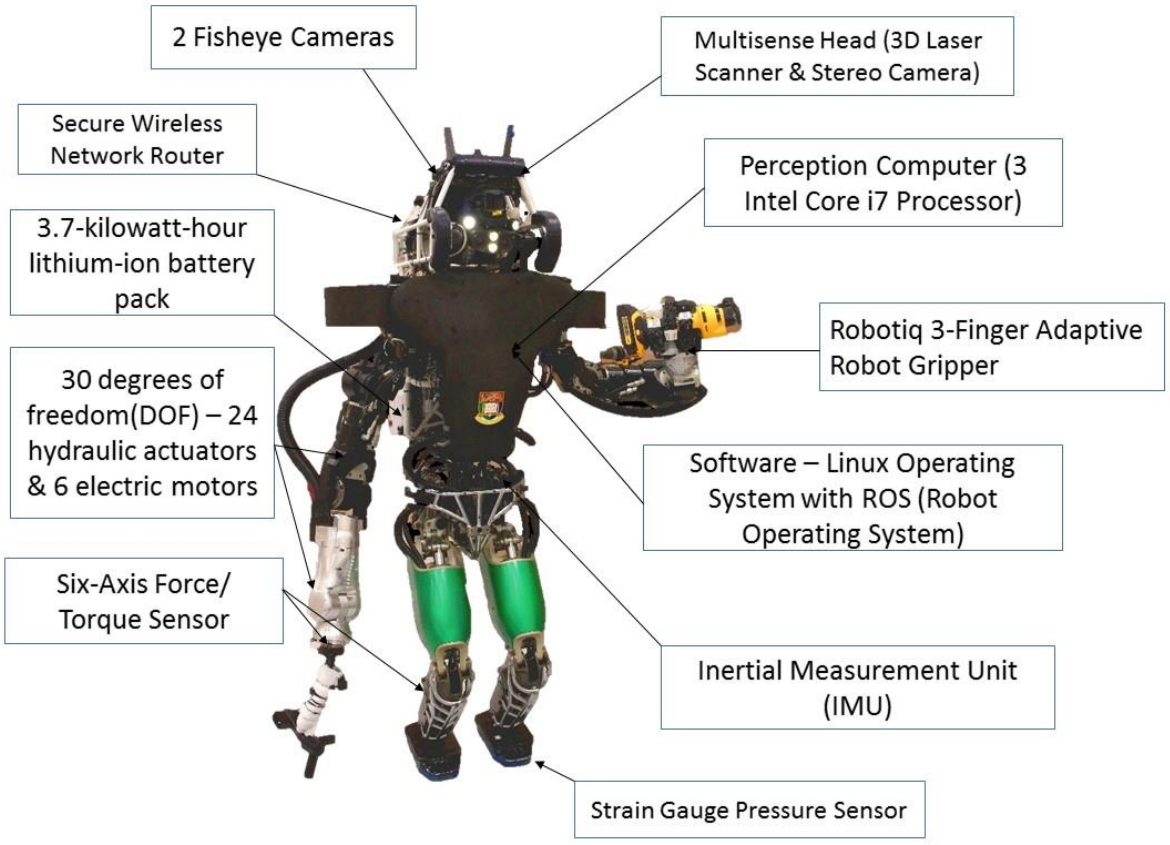

Figura 5 – Robô Atlas 4 da "Team HKU" do *DARPA Robotics Challenge*

2.2.1.1 Conceitos de base do ROS

O ROS tem três níveis de conceitos: o nível do sistema de ficheiros, o do grafo de computação e o da comunidade ROS [11]. Neste subcapítulo vamos apenas falar dos dois primeiros.

Os conceitos do nível do sistema de ficheiros abrangem principalmente os recursos que se podem encontrar no disco rígido do(s) computador(es), como:

- Pacotes: os pacotes são a unidade principal para organizar *software* dentro do ROS. Um pacote pode conter processos de tempo de execução do ROS (nós), uma biblioteca dependente do ROS, *datasets*, ficheiros de configuração ou qualquer outro recurso que se possa organizar de forma lógica e útil, como a descrição em URDF do robô, ou todas as configurações e *software* específico para a navegação do robô em desenvolvimento;
- Meta-pacotes: meta-pacotes são pacotes específicos que representam um grupo de outros pacotes. Apesar de ser possível utilizá-los, regra geral são utilizados apenas como *placeholders* para o sistema de *building* do ROS;
- Manifestos de pacotes: Os ficheiros `package.xml` são manifestos que fornecem meta-dados sobre um pacote, incluindo o seu nome, versão, descrição, licença, dependências e outra informação. O formato deste ficheiro está definido no REP-127 [12];
- Repositórios: os repositórios são conjuntos de pacotes que partilham um sistema de controlo de versões, e pode conter apenas um pacote;
- Tipos de mensagem: são ficheiros com a extensão `msg` presentes na subdiretoria `msg` de um pacote que definem a estrutura de dados das mensagens enviadas no ROS.
- Tipos de serviço: são ficheiros com a extensão `srv` presentes na subdiretoria `srv` de um pacote que definem as estruturas de dados do pedido e da resposta para os serviços no ROS.

O grafo de computação é a rede *peer-to-peer* de processos ROS que processam dados em conjunto. Os conceitos básicos do grafo de computação do ROS são nós, mestre, servidor de parâmetros, mensagens, serviços, tópicos e *bags*, disponibilizando todos eles dados para o grafo de maneiras diferentes:

- Nós: nós são processos que efetuam computação. O ROS é desenhado para ser modular a uma escala pequena (o sistema de controlo de um robô compreende diversos nós). Por exemplo, um nó controla um *laser range-finder* (LRF), outro controla os motores das rodas, outro efetua a

localização, outro efetua o planeamento, etc. Um nó ROS é escrito usando uma biblioteca de cliente ROS, como o roscpp (C++) ou o rospy (Python);

- Mestre: o mestre ROS disponibiliza registo e *lookup* de nomes para o resto do grafo de computação. Sem o mestre, os nós não seriam capazes de se encontrarem, trocar mensagens ou invocar serviços;
- Servidor de parâmetros: o servidor de parâmetros permite que dados sejam guardados por chave numa localização central, e neste momento faz parte do mestre;
- Mensagens: os nós comunicam entre si passando mensagens. Uma mensagem é simplesmente uma estrutura de dados composta por campos de tipo definido. São suportados tipos primitivos (inteiros, números de virgula flutuante, valores booleanos, etc.), assim como *arrays* de tipos primitivos e estruturas à semelhança das estruturas em C;
- Tópicos: as mensagens são encaminhadas usando um sistema de transporte com semântica de publicação/subscrição. Um nó envia uma mensagem publicando-a num determinado tópico (o tópico é o nome utilizado para identificar o conteúdo da mensagem). Um nó que esteja interessado nessa informação irá subscrever esse tópico. Podem existir mais múltiplos *publishers* e *subscribers* para o mesmo tópico, e um nó pode publicar ou subscrever mais do que um tópico. No geral, os *publishers* e *subscribers* não estão cientes da existência do outro, sendo a ideia separar a produção de informação do seu consumo. Podemos ver os tópicos como barramentos de informação com estrutura definida e qualquer nó pode ligar-se ao barramento para enviar ou receber mensagens desde que elas tenham a estrutura correta;
- Serviços: apesar do modelo de publicação/subscrição ser um paradigma de comunicação muito flexível, o seu modelo de comunicação de N para N participantes unidirecional não é apropriado para interações pedido/resposta, que são muitas vezes necessárias num sistema distribuído. Este tipo de interações é por isso feito através de serviços, que são definidos por um par de estruturas de mensagens, um para o pedido e outro para a resposta. Um nó disponibiliza um serviço sob um nome e o cliente utiliza o serviço enviando uma mensagem com o pedido e esperando a resposta. Regra geral, as bibliotecas de cliente ROS apresentam esta interação ao programador como se fosse uma *remote procedure call* (RPC);
- *Bags*: Os *bags* são um formato para guardar e reproduzir mensagens de dados ROS, sendo um mecanismo importante para guardar dados, como dados de sensores, que são difíceis de adquirir mas necessários para desenvolver e testar algoritmos.

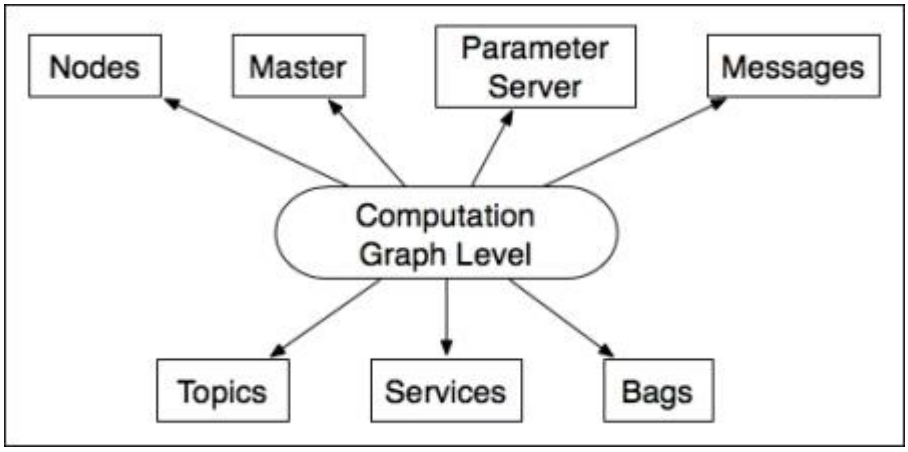

Figura 6 – Grafo de computação do ROS

Resumindo, o mestre ROS serve como um *nameservice* no grafo de computação, armazenando informação de registo de tópicos e serviços para os nós ROS. Os nós comunicam com o mestre para reportar a sua informação de registo. Quando comunicam com o mestre, podem receber informação sobre outros nós e ligar-se aos mesmos como e quando for apropriado. O mestre irá ainda fazer *callbacks* para os nós envolvidos quando as informações de registo se alteram, permitindo assim aos nós criar ligações de forma dinâmica à medida que novos nós começam a correr. Os nós ligam-se a outros nós diretamente: o mestre apenas disponibilizada informação de *lookup*, da mesma maneira que um servidor de DNS. Nós que subscrevam um tópico irão pedir ligações dos nós que publicam esse tópico e irão estabelecer uma ligação usando um protocolo de ligação acordado, tipicamente o TCPROS, que utiliza *sockets* TCP/IP.

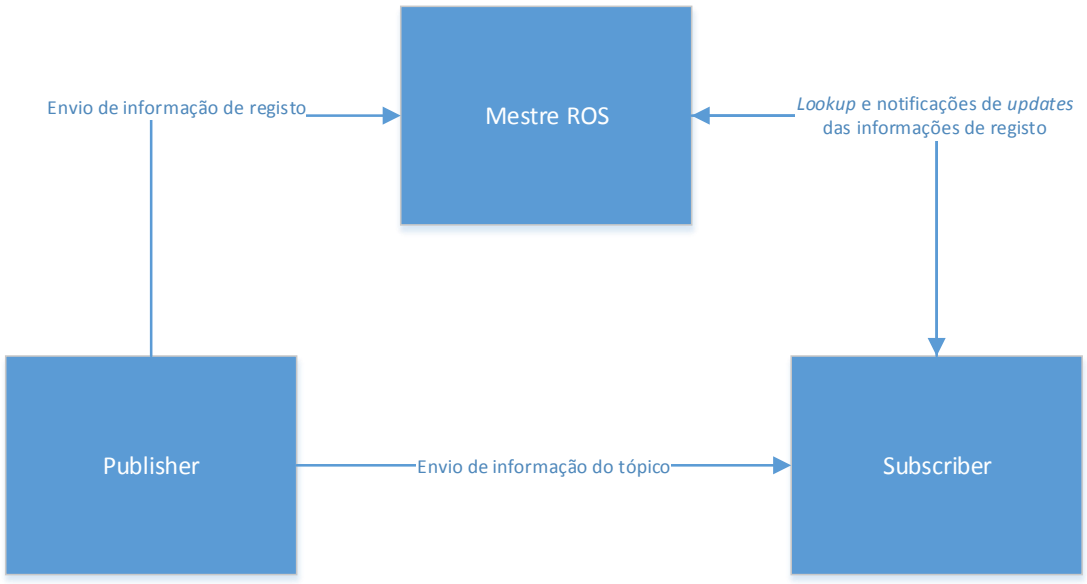

Figura 7 – Funcionamento das comunicações no ROS

Esta arquitetura permite uma operação separada de cada sistema, onde nomes são utilizados como os meios através dos quais sistemas maiores e mais complexos podem ser construídos, tendo por isso um papel muito importante no ROS, uma vez que os nós, tópicos, serviços e parâmetros todos têm nomes. Todas as bibliotecas de cliente do ROS suporta o mapeamento de nomes, o que permite um programa já compilado ser reconfigurado para operar numa topologia de grafo de computação diferente durante a sua execução. Por exemplo, podemos ter um nó que publica a informação gerada por um LRF num determinado tópico que é utilizado pelo sistema de navegação. Se este LRF por algum motivo avariar e tivermos montado uma Kinect no robô, podemos redirecionar a informação dada pelo nó do LRF para um tópico diferente, e configurar o nó da Kinect para mapear o espaço à sua frente e fundir essa informação de forma a termos uma informação no formato da do LRF, publicá-la no tópico utilizado pelo sistema de navegação, sem nunca ter de reiniciar todos os serviços do robô ou reescrever código.

A versão de ROS utilizada neste projeto é a Hydro.

2.2.2. Player

O "The Player Project" permite criar *software* gratuito para investigação e desenvolvimento na área da robótica e sistemas sensoriais [13]. Entre o *software* desenvolvido por este projeto, temos o Player, uma interface de controlo para dispositivos robóticos, que disponibiliza uma interface de rede para uma panóplia de robôs e sensores. Este segue uma lógica de cliente-servidor que permite que sejam desenvolvidos programas de controlo para robôs em praticamente qualquer linguagem de programação com uma ligação de rede para o robô, sendo por isso um dos sistemas *middleware* mais utilizados na área da robótica.

2.3. Mapeamento e localização

O objetivo de um robô autónomo é ser capaz de construir ou usar um mapa ou planta e localizar-se nele. Para atingir esse objetivo, é preciso resolver o problema de

simultaneous location and mapping (SLAM) [14]. Os métodos mais populares para resolver este problema passam pela utilização de filtros de partículas ou o filtro de Kalman estendido.

2.3.1. Gmapping

O Gmapping é um filtro de partículas Rao-Blackwell para criar mapas utilizando dados de um LRF [15]. Este algoritmo utiliza um filtro de partículas em que cada partícula tem um mapa individual, sendo depois utilizadas técnicas adaptativas para reduzir o número de partículas no filtro para aprendizagem de mapas. Para além de utilizar a informação relativa ao movimento do robô, é ainda utilizada a mais recente observação, diminuindo o grau de incerteza sobre a pose do robô no preditor do filtro.

Para utilizarmos este algoritmo no ROS, existe um pacote com o mesmo nome que faz um *wrapping* do mesmo. Este pacote requer que o robô disponibilize a informação de odometria e que o mesmo esteja equipado por um LRF fixo montado na horizontal.

2.3.2. Hector

O Hector SLAM, desenvolvido pela equipa Hector da TU Darmstadt, é um algoritmo flexível e escalável de SLAM que pode ser utilizado sem ter informação de odometria, fazendo fusão de informação de sensores inerciais e de sistemas de LIDAR com frequências de atualização elevadas utilizando um filtro de Kalman estendido [16]. Apesar de não ter nenhuma função de *loop closure* explícita, o *feedback* disponível sobre este algoritmo é bastante positivo quando utilizado com acelerómetros precisos e bem calibrados.

2.3.3. Adaptive Monte Carlo Locatization

Quando temos o problema de mapeamento resolvido, podemos tomar uma diferente abordagem ao problema de localização, utilizando algoritmos que se focam apenas nisso mesmo, como é o caso do algoritmo de Monte Carlo, implementado no módulo ROS `amcl` [17].

O módulo `amcl` é um sistema de localização probabilística para robôs que se movimentam num ambiente 2D. Este sistema implementa o algoritmo de localização adaptativo de Monte Carlo que utiliza um filtro de partículas para seguir a posição de um robô num mapa já conhecido.

2.4. Navegação de robôs móveis

Depois de se localizar num mapa, um robô móvel tem como objetivo deslocar-se da sua posição atual até um ponto de destino. Para tal, é necessário planear uma trajetória livre de colisões, válida, atingível e que preferencialmente minimize o tempo de viagem entre as duas posições. A este problema chama-se planeamento de trajetória, e regra geral a solução passa pela divisão do mesmo em duas tarefas: um planeamento de trajetória global, que deverá gerar uma trajetória de alto nível não-refinado entre a posição inicial e o objetivo; e um planeamento de trajetória local que deverá gerar uma trajetória de baixo nível que refina a trajetória global junto do robô e evita obstáculos detetados.

2.4.1. Planeadores globais

As soluções de planeamento global utilizadas são regra geral baseadas em algoritmos de procura em árvores e grafos como o A* e o Dijkstra [18]. Estes algoritmos são muito eficientes a nível computacional e oferecem a solução ótima teórica para um dado mapa de custos.

Neste projeto, o planeamento global é feito utilizando o pacote `navfn` do ROS, e o mesmo implementa o algoritmo de Dijkstra, existindo ainda o pacote `global_planner` que implementa tanto o algoritmo A* como o de Dijkstra [19].

2.4.2. Planeadores locais

O planeamento local é um sistema reativo, baseado em informação sensorial, tendo como objetivo principal evitar colisões com obstáculos. As soluções mais comuns são metodologias de janela dinâmica (*dynamic window approach* – DWA) ou *trajectory rollout*, sendo o DWA mais eficiente [20].

No ROS, o planeamento local é feito utilizando o pacote `base_local_planner` que implementa as duas metodologias mencionadas acima.

2.4.3. Mapas de custo

Para efetuar planeamento de trajetórias, os algoritmos utilizados utilizam mapas de custo para calcular o custo de cada trajetória e assim escolher quais deverá ser utilizada. No caso da navegação 2D, os mapas de custo são representados como um mapa de células cujo valor é o custo de atravessar essa célula [21].

3. Arquitetura do sistema

O robô desenvolvido encontra-se divido em 4 partes conforme ilustra o diagrama da Figura 8: a plataforma robótica, o ROS que vai ligar a plataforma robótica ao controlador de alto nível do robô, o controlador de alto nível do robô e a interface gráfica para o utilizador.

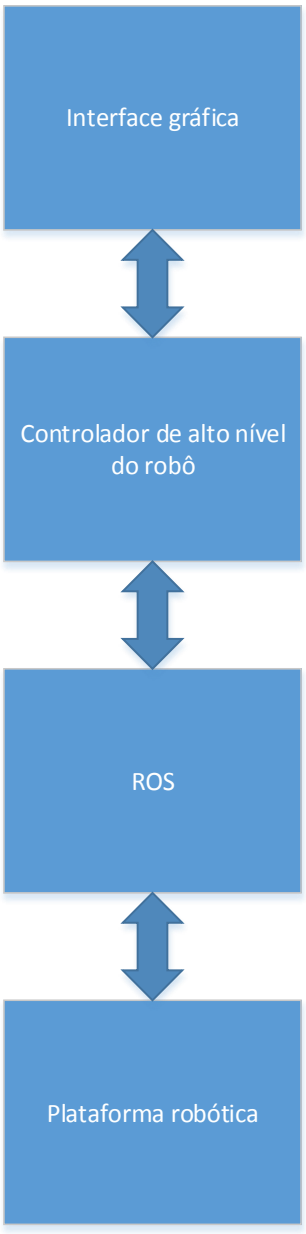

Figura 8 – Arquitetura global do sistema

3.1. Descrição da plataforma robótica

A plataforma robótica desenvolvida baseia-se no TurtleBot [22], uma base robótica *open-source* composta por uma base robótica Kobuki [23] (fabricada pela Yujin Robot), uma Kinect [24] da Microsoft como sistema de visão e uma estrutura de níveis feita de MDF e alumínio maquinados para se poder colocar um portátil ou outros sensores no robô (Figura 9). Esta plataforma foi adquirida com o objetivo de servir de base a robôs móveis baseados em ROS no IPT, sendo uma das plataformas com mais suporte na comunidade ROS.

Figura 9 – TurtleBot

Como se pretendia colocar um ecrã tátil no robô para interação com o utilizador, foi decidido reformular a estrutura do mesmo, guardando toda a estrutura original do TurtleBot para futuros projetos.

Depois de concluído, o robô desenvolvido ficou com o aspeto da Figura 10, sendo o mesmo composto pela base robótica Kobuki, um LRF Hokuyo UTM-30LX, a estrutura feita à medida, um ecrã tátil e um *netbook* Asus.

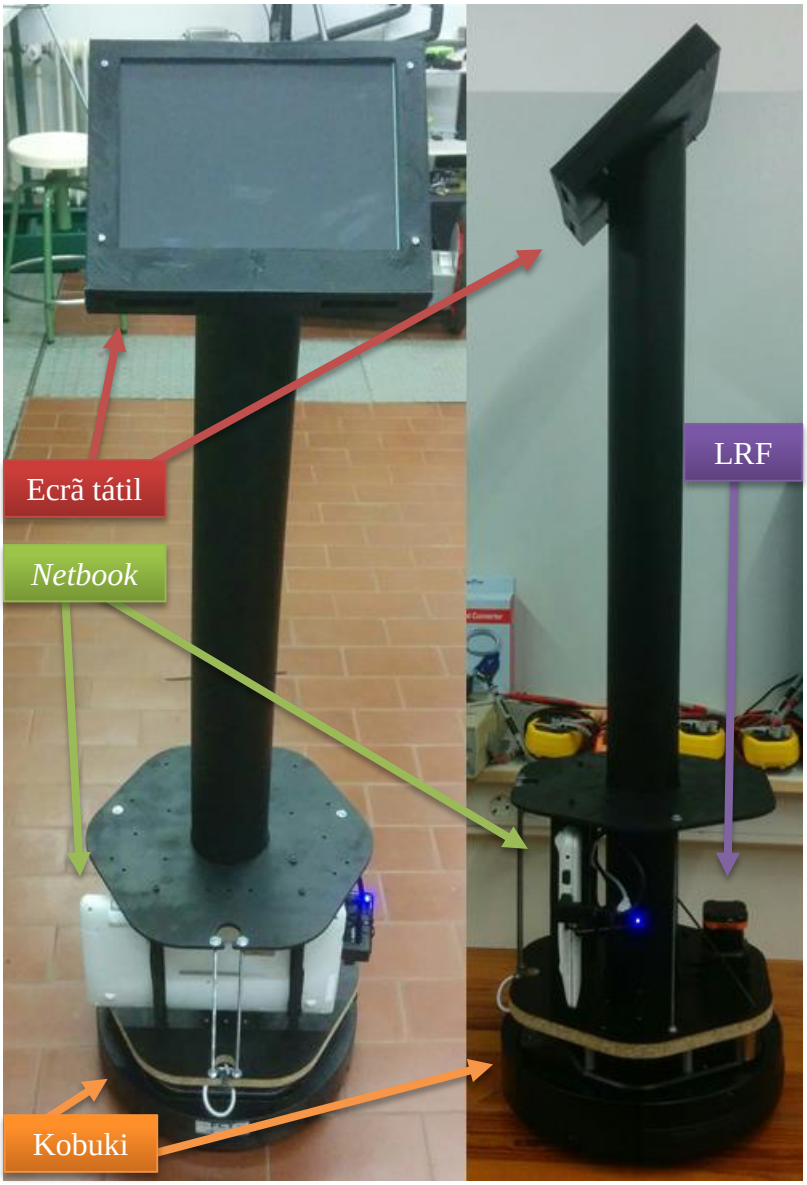

Figura 10 – Robô rececionista: vista do utilizador e vista lateral

3.1.1. Base robótica Kobuki

A base robótica motorizada utilizada é uma base Kobuki desenvolvida pela Yujin Robot e é comercializada como uma base para desenvolvimento robótico em ambientes *indoor*, partilhando parte da eletrónica do aspirador robótico iClebo POP, semelhante ao Roomba.

Figura 11 – Bases robóticas: Kobuki à esquerda, iClebo POP à direita

A base Kobuki está desenhada para ter uma velocidade translacional máxima de 65cm/s e uma velocidade rotacional máxima de π rad/s e transportar até 5kg em piso rígido (4kg em carpete). Esta base tem uma configuração diferencial, tendo duas rodas num eixo comum, sendo cada roda controlada independentemente e a direção do deslocamento controlada com base na diferença de velocidade entre as duas rodas, permitindo a rotação da base sobre si mesma mas tornando impossível efetuar movimentos de translação ao longo do eixo das mesmas. Para se obter estabilidade, são ainda utilizadas 2 rodas livres, colocadas num eixo perpendicular ao das rodas motrizes.

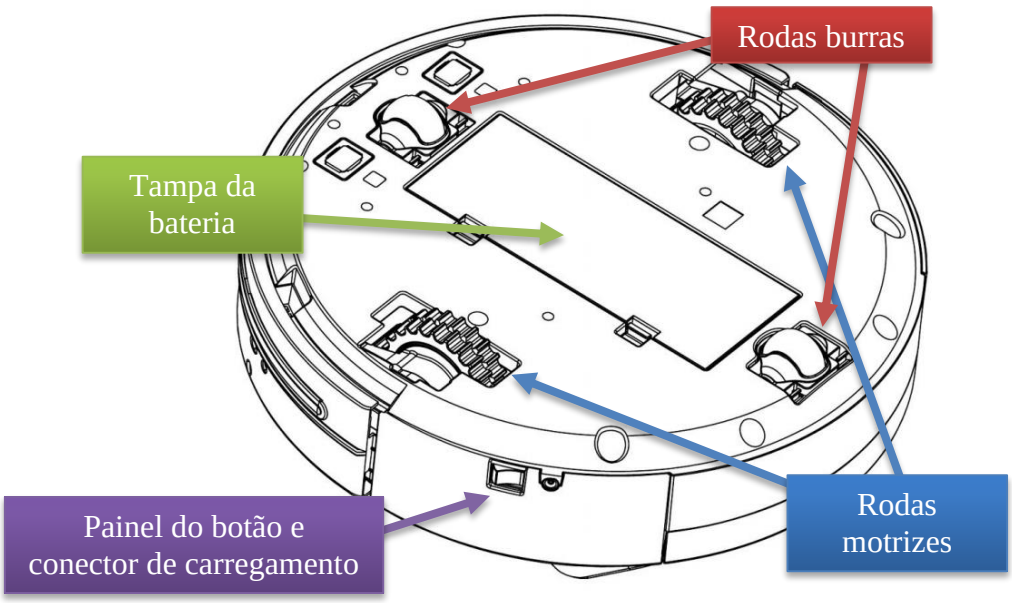

Figura 12 – Vista de baixo do Kobuki: rodas, tampa da bateria, conector de carregamento e *switch* ON/OFF

Em termos sensoriais, a base inclui três *bumpers* e três sensores de precipício (esquerda, centro e direita), dois sensores de queda nas rodas motrizes, um *encoder* em cada motor (52 pulsos por revolução do *encoder*, 2578.33 pulsos por revolução da roda, 11.7 pulsos por milímetro percorrido), um giroscópio de 1 eixo (110 graus por segundo), 3 botões de toque (não são suportados cliques simultâneos) e 3 sensores infravermelhos integrados no *bumper* para *docking*. Em termos de atuadores, para além dos 2 motores, temos dois LEDs programáveis com 2 cores (verde e laranja) e um *beeper* com sequências pré-programadas.

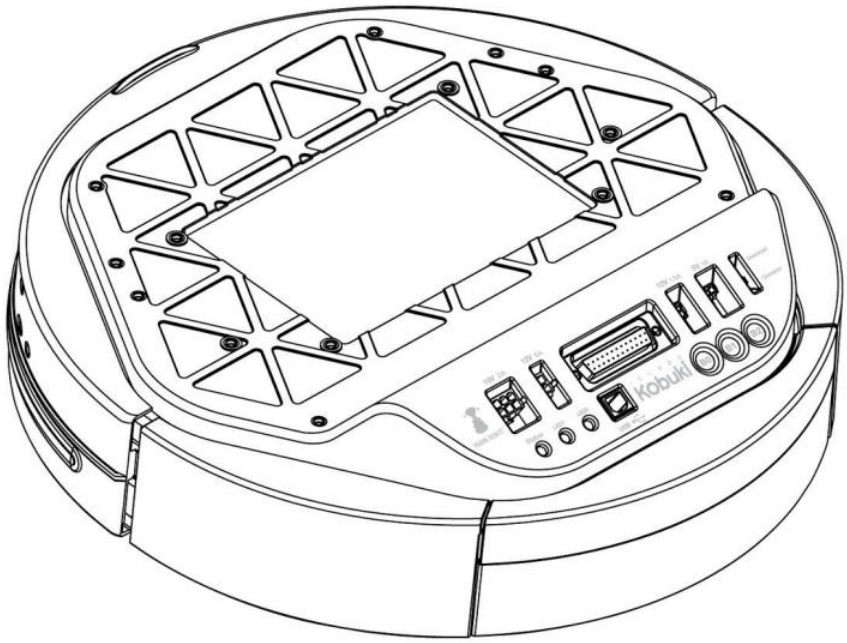

Figura 13 – Vista de cima da base Kobuki: conectores DB25, USB e de alimentação, LEDs, botões e bumper

É ainda possível expandir as capacidades da base através do conector DB25 que expõe barramentos de alimentação 3.3V/1A e 5V/1A, 4 entradas analógicas, 4 entradas digitais e 4 saídas digitais. A comunicação com outros dispositivos pode ser feita através de USB ou dos pinos TX e RX no conector DB25.

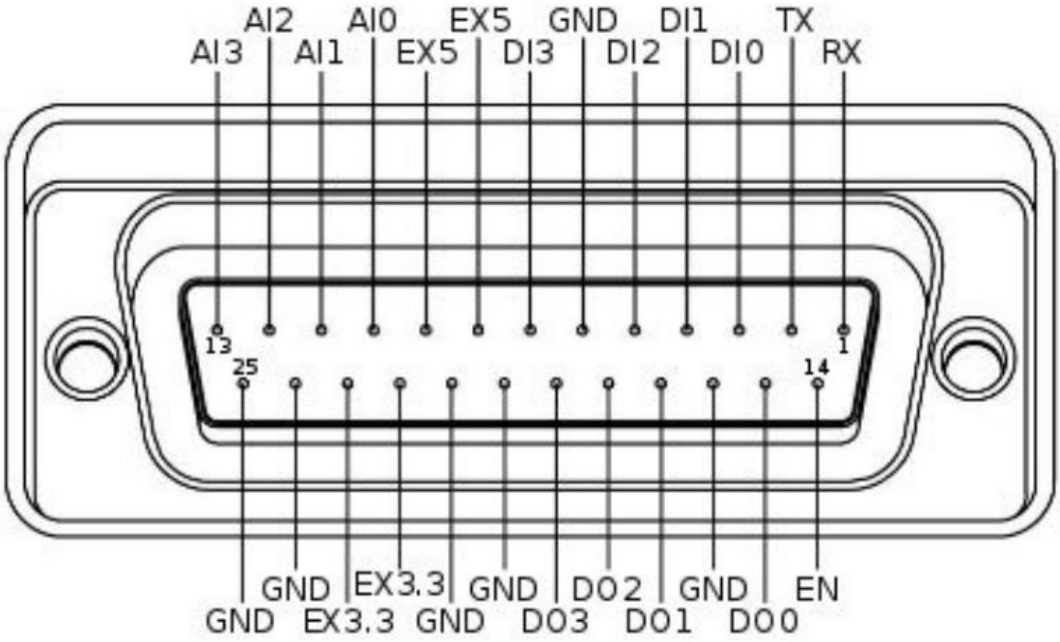

Figura 14 – *Pinout* do conector DB25

A base é fornecida de série com uma bateria de lítio de 2200mAh (modelo pequeno – 4S1P) e opcionalmente com uma de 4400mAh (modelo grande – 4S2P) de 14.8V, esperando-se ter um tempo de operação de 3 ou 7 horas e um tempo de carregamento de 1.5 a 2.6 horas, conforme seja utilizada a bateria pequena ou grande, respetivamente. As baterias são feitas usando células de iões de lítio de 3.7V, sendo a pequena feita com 1 série de 4 módulos e a grande com 2 séries de 4 módulos em paralelo. Estas baterias são utilizadas para alimentar a base robótica e 3 barramentos de alimentação para o exterior: um de 12V 5A, outro de 12V 1.5A e ainda um de 5V 1A. Existe ainda um rail de 19V 2.1A para o exterior que serve para carregar um *netbook*, mas apenas se encontra alimentado quando a base está a ser carregada. O carregador da base robótica é um conversor 100-240VAC 50/60Hz para 19VDC, tendo como correntes máximas 1.5A de entrada e 3.16A de saída, sendo o mesmo ligado à base robótica utilizando um *jack*, encontrando-se o positivo ao centro.

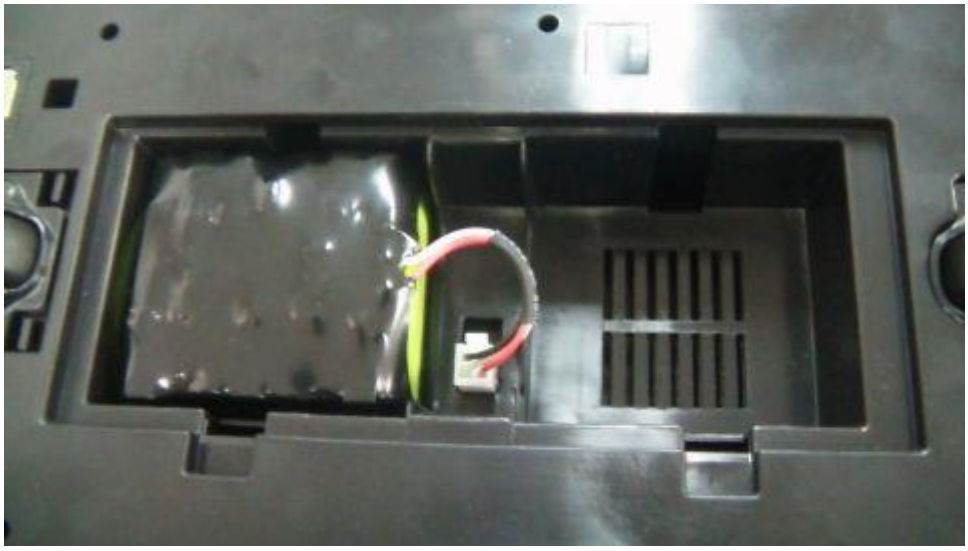

Figura 15 – Bateria 4S2P ligada dentro do compartimento da bateria do Kobuki

3.1.1.1 Placa de alimentação

Para facilitar o trabalho experimental e a ligação dos diversos componentes à base robótica, foi desenhada uma placa de alimentação de periféricos. Esta placa faz a ligação entre cada um dos *rails* e os periféricos a alimentar: o LRF e o ecrã tátil. Ao longo dos testes realizados foi necessário utilizar um ponto de acesso sem-fios (AP) para efetuar teleoperação e monitorização remota, tendo sido usado um *router* Asus WL-500g Premium v2 com *firmware* DD-WRT, cuja tensão de alimentação é 5VDC e o consumo medido não excede 1A, e como tal foi ligado ao barramento de alimentação 5V/1A. Esta placa foi colocada entre a placa base da estrutura e a base robótica, utilizando espaçadores com fita de dupla-face.

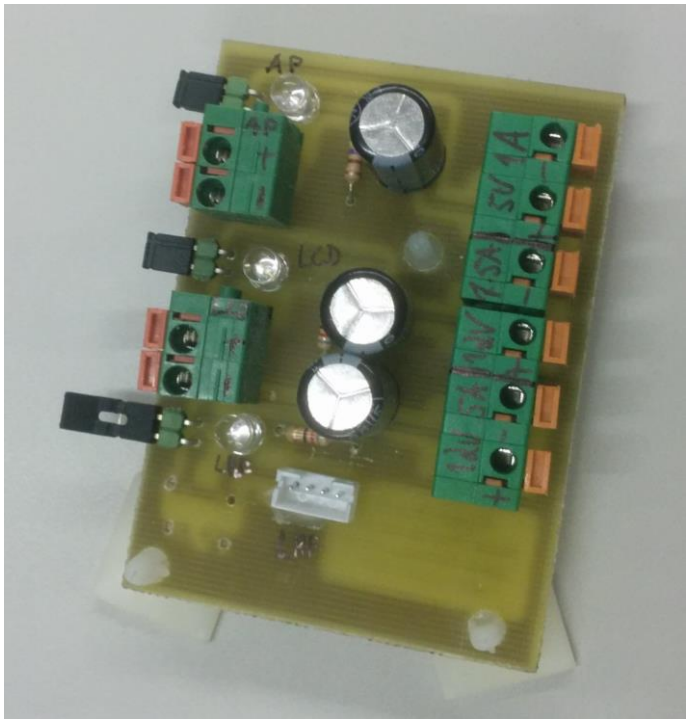

Figura 16 – Placa de alimentação

Esta placa tem bornes de mola para ligação dos barramentos de alimentação à mesma, assim como para ligação do LCD e do AP uma vez que os mesmos apenas têm um recetáculo para ligação de um *jack*. Como o LRF possui cabo de alimentação com uma ficha específica, foi reservado o espaço para a colocação futura de bornes de mola iguais aos usados, para além do conector específico para o LRF utilizado. A alimentação a cada dispositivo pode ser desligada removendo o respetivo *jumper*, servindo os LEDs existentes para mostrar se a respetiva saída se encontra alimentada ou não. Foram ainda colocados condensadores para amortecer picos de corrente e evitar que alguma proteção de sobrecarga existente na base fosse despoletada intempestivamente durante o funcionamento do robô. De notar que condensadores com capacidades elevadas irão evidentemente levar a correntes transitórias elevadas no momento em que o robô é colocado em serviço, pelo que é imperativo verificar que essas correntes não despoletam a atuação de proteções, o que não aconteceu com os condensadores utilizados.

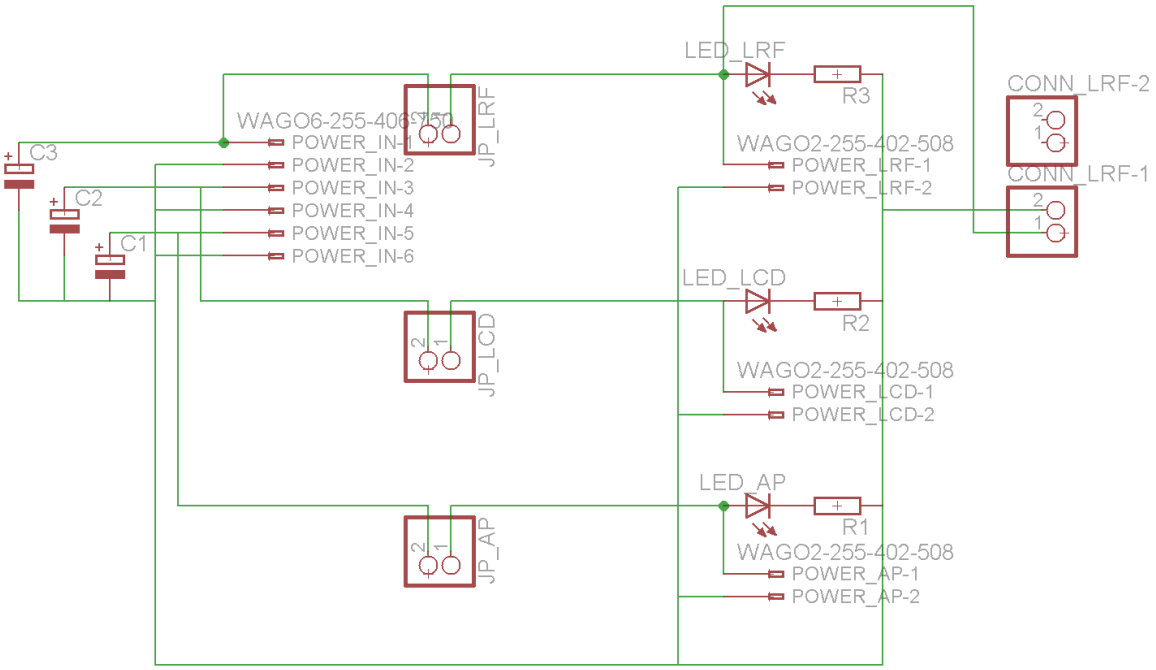

Figura 17 – Esquemático da placa de alimentação

3.1.2. *Laser range-finder* Hokuyo UTM-30LX

Apesar do TurtleBot ser fornecido com a Kinect para visão do robô, a mesma não produz informação tão exata para navegação como um LRF. Por essa razão, foi decidido utilizar um LRF para visão, neste caso o Hokuyo UTM-30LX.

Figura 18 – Hokuyo UTM-30LX: o *laser range-finder* utilizado no robô

O Hokuyo UTM-30LX é um LRF de pequena dimensão capaz de detetar obstáculos a distâncias entre 10cm e 30m com uma resolução de 1mm num arco de 270º com uma resolução de 0.25º, tendo uma velocidade de *scan* de 25ms, utilizando um laser da classe 1 com um comprimento de onda de 905nm. Este LRF é alimentado com 12VDC±10%, tendo um consumo inferior a 8W e 1A como corrente máxima de entrada. O índice de proteção do equipamento é IP64, sendo por isso possível utilizá-lo em ambientes exteriores se necessário. Tendo em conta as características do LRF, este foi ligado ao *rail* de alimentação 12V/5A.

3.1.3. Ecrã tátil

Para a interação homem-computador, optámos por utilizar um ecrã de toque resistivo, sendo assim possível utilizá-lo por exemplo com luvas calçadas, o que não seria possível com um sistema de toque capacitivo. O ecrã de toque escolhido é o LP104X0114-FNR da marca *Flat Display Technology* de 10.4" (resolução nativa 1024x768) *open-frame*, uma vez que apesar de ter controladores proprietários (eGalax) que lhe dão mais funções, é suportado pelos controladores para ecrãs de toque já presentes no *netbook* a utilizar no robô. A imagem é transmitida por HDMI, apesar do ecrã também suportar VGA, e o sistema de toque comunica utilizando USB. É ainda possível ligar duas colunas 2W@4Ω ao ecrã. Uma vez que este ecrã é *open-frame*, foi desenhada uma caixa para o mesmo, estando este trabalho descrito mais à frente, no capítulo referente à estrutura do robô.

Figura 19 – Ecrã de toque FDT LP104X0114-FNR utilizado no robô

Tendo em conta as características do ecrã, este foi ligado ao barramento de alimentação 12V/1.5A.

3.1.4. *Netbook* ASUS

Para controlar o robô e correr ROS, foi utilizado um *netbook* ASUS F200MA-BING-KX387B. Este *netbook* de 11.6" está equipado com um processador Intel Celeron N2830, 2GB de RAM e um disco de 500GB, tendo saída de vídeo HDMI e VGA, três portas USB (uma delas USB 3.0), uma porta Ethernet, Bluetooth e WiFi. Apesar do controlador gráfico ser *onboard*, o mesmo possuí capacidade suficiente para a visualização 3D feita usando o RViz. O sistema operativo utilizando foi o Ubuntu 12.04 de 64bits.

Figura 20 – *Netbook* ASUS F200MA-BING-KX387B

3.1.5. Estrutura do robô

Para a estrutura do robô foi decidido manter uma estrutura semelhante à do TurtleBot e reutilizando tanto material já existente nas oficinas do IPT quanto possível. Foi maquinada na CNC do laboratório VITA.IPT uma placa em contraplacado de 17 mm para tomar o lugar da primeira placa de MDF do TurtleBot, tendo uma furação semelhante à do TurtleBot, mas com um entalhe para encaixar um tubo de PVC 90mm que iria servir de coluna para suporte do ecrã tátil e para passagem da cablagem e furação específica para a fixação do laser.

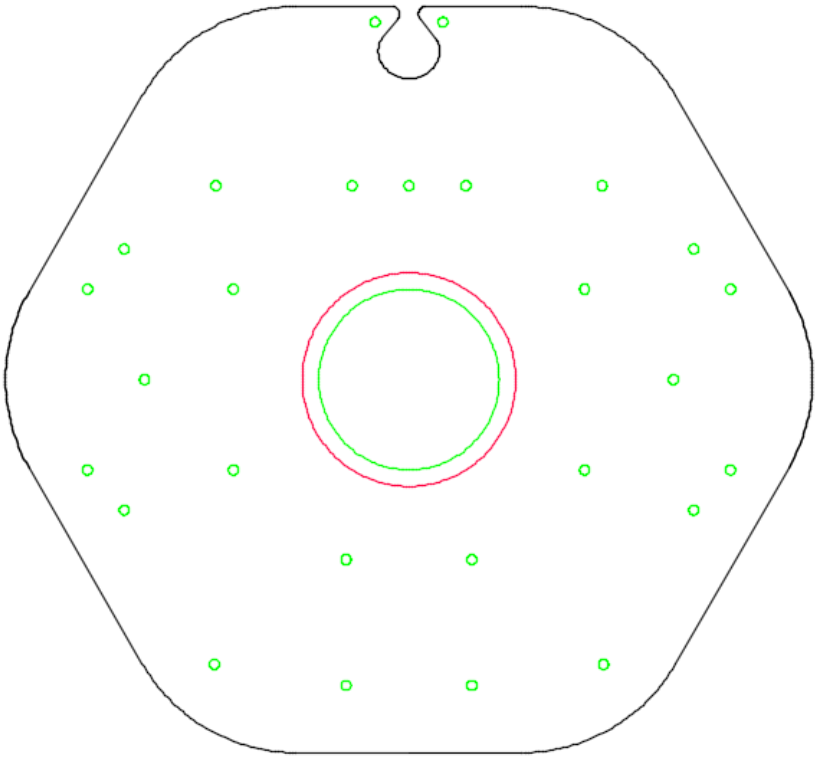

Figura 21 – Vista de cima da placa base da estrutura

Figura 22 – Placa base da estrutura no fim da maquinação

Uma vez que o tubo de PVC que suporta o ecrã tátil tem um comprimento considerável, decidiu-se fazer uma placa mais acima desta para dar maior estabilidade ao robô. Tento em conta que o propósito da mesma é simplesmente amparar o tubo de PVC, fez-se uma placa semelhante à placa base, laminada a partir de 2 placas de platex, sendo a mesma pintada de preto. Entre estas duas placas foi colocado o *netbook*, tendo sido também maquinadas em platex e laminadas duas placas de apoio para o mesmo usando 3 camadas para cada peça (Figura 23). Para garantir que todas as peças ficavam justas, foi utilizada fita de tecido em todas as superfícies que iriam estar em contacto com outras de forma a minimizar folgas.

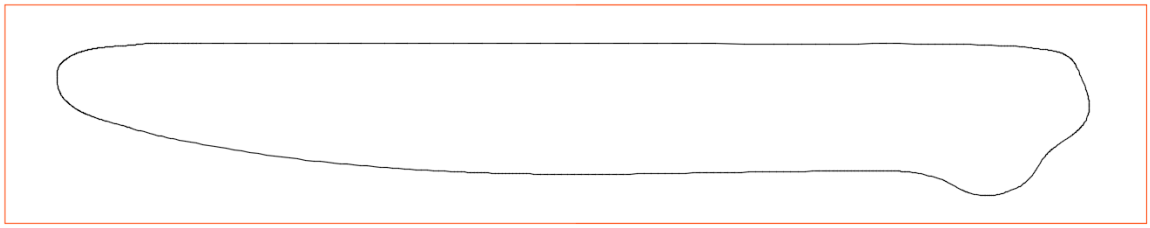

Figura 23 – Vista de topo dos apoios do *netbook*

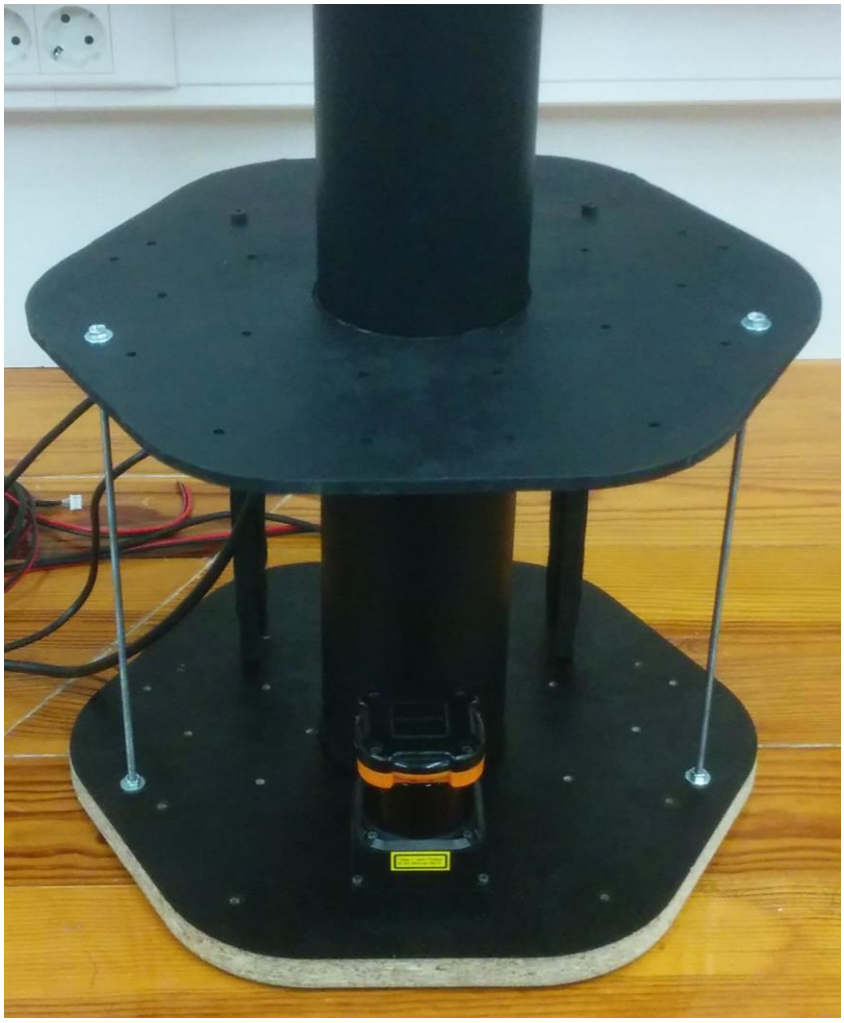

Figura 24 – Placas unidas com varão roscado M4 com o tubo de PVC e o LRF

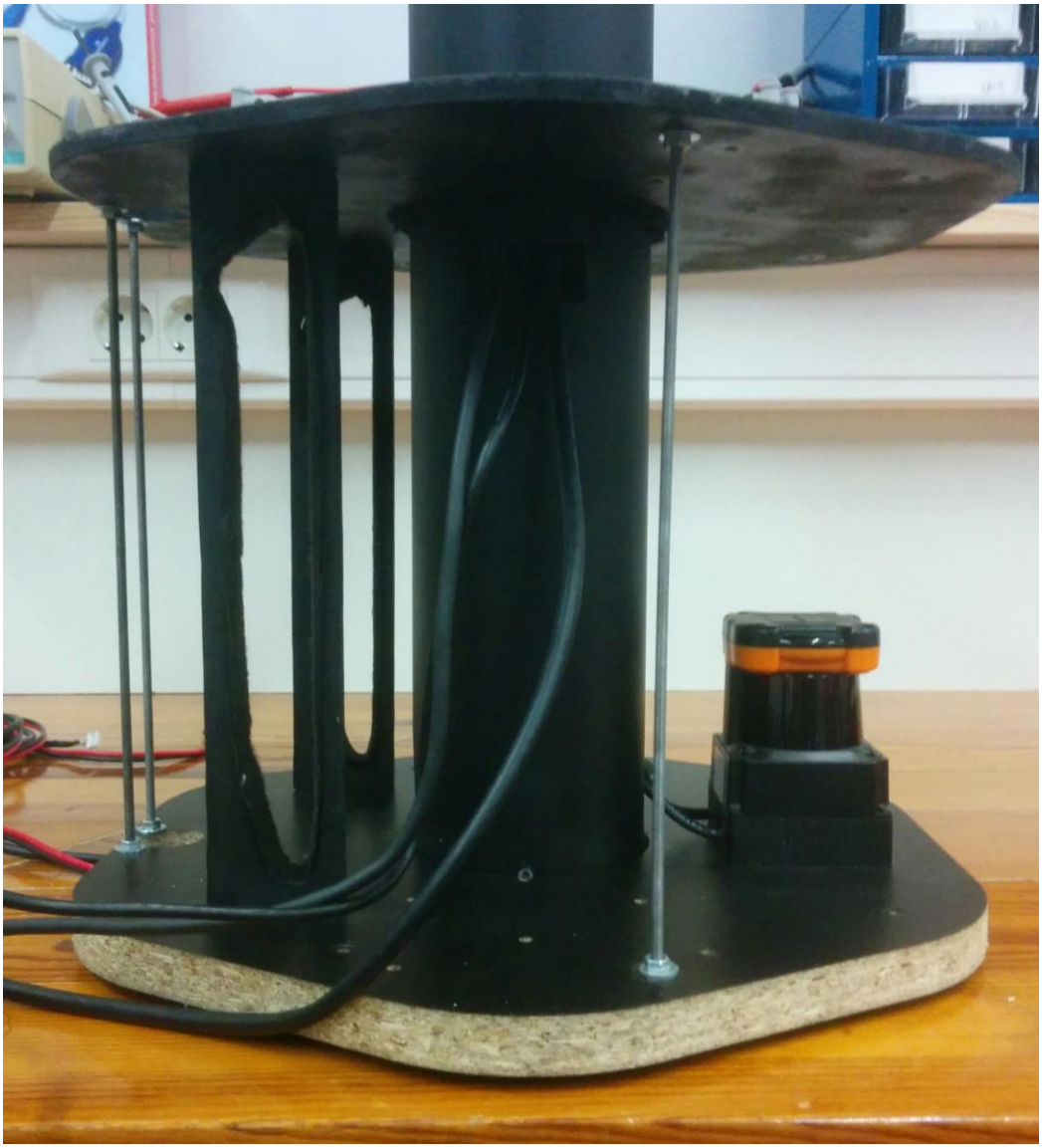

Figura 25 – Placas unidas com varão roscado M4 com o tubo de PVC, o LRF e os apoios para o netbook

Para fixar o ecrã ao robô, foi desenhado em 3D uma caixa para o mesmo (Figura 26). Esta caixa foi impressa em 3D pelo aluno da LEEC Rui Costa e Silva em 4 peças, por limitações na área de impressão da impressora 3D. Depois de montada, esta caixa é colada no topo do tubo de PVC, que foi cortado a um ângulo de 40º com o plano horizontal, ficando o robô com o aspeto que podemos ver na Figura 27.

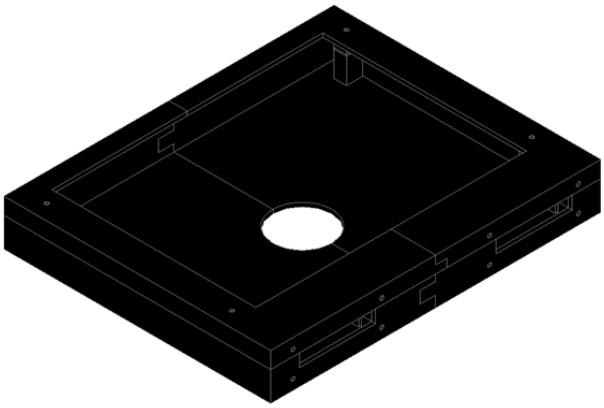

Figura 26 – Modelo 3D da caixa do ecrã tátil

Figura 27 – Aspeto final do robô depois de totalmente montado

3.2. Integração em ROS

Uma vantagem na utilização do *hardware* escolhido é o facto de tanto a base robótica como o LRF já terem pacotes ROS implementados que diminuem assim a quantidade de trabalho a desenvolver. Aliado ao facto de já existirem pacotes de SLAM e navegação implementados no ROS, isto significa que para começar a trabalhar é "apenas" necessário juntar os nós necessários para o funcionamento do robô, configurando e satisfazendo os requisitos de cada nó, mapeando tópicos e lançando nós. Sendo este robô baseado no TurtleBot, tanto em termos de *hardware* como *software*, e tendo sido usados os seus pacotes como base para o desenvolvimento deste robô, é importante que seja feita uma análise do TurtleBot e dos seus pacotes, de forma a se compreender o funcionamento do TurtleBot e as modificações necessárias para o funcionamento de um robô baseado no mesmo.

3.2.1. *Overview* dos pacotes do TurtleBot

Para permitir o funcionamento do TurtleBot, os seus criadores desenvolveram uma coleção de pacotes. Seguindo a convenção informal de identificação de pacotes existente no universo ROS, podemos ver os pacotes específicos para o TurtleBot numa instalação de ROS a funcionar procurando pelos pacotes cujo nome comece por `turtlebot`. Na instalação utilizada, podemos ver a lista dos pacotes existentes escrevendo `roscd turtlebot` na consola e clicando na tecla de tabulação para ver que pacotes existem começados por `turtlebot`, como podemos ver na Figura 28.

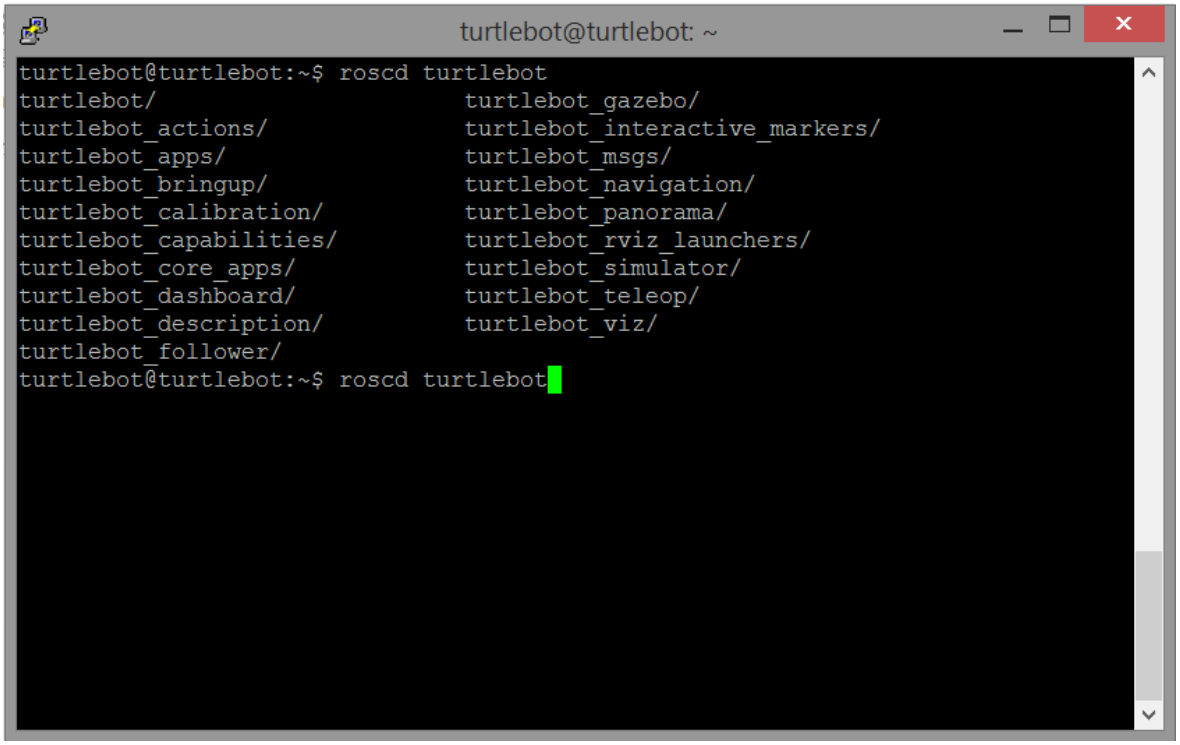

Figura 28 – Lista dos pacotes existentes do TurtleBot através de uma ligação de terminal remoto

Depois de analisados os pacotes, podemos então agrupar os pacotes existentes de acordo com a Figura 29.

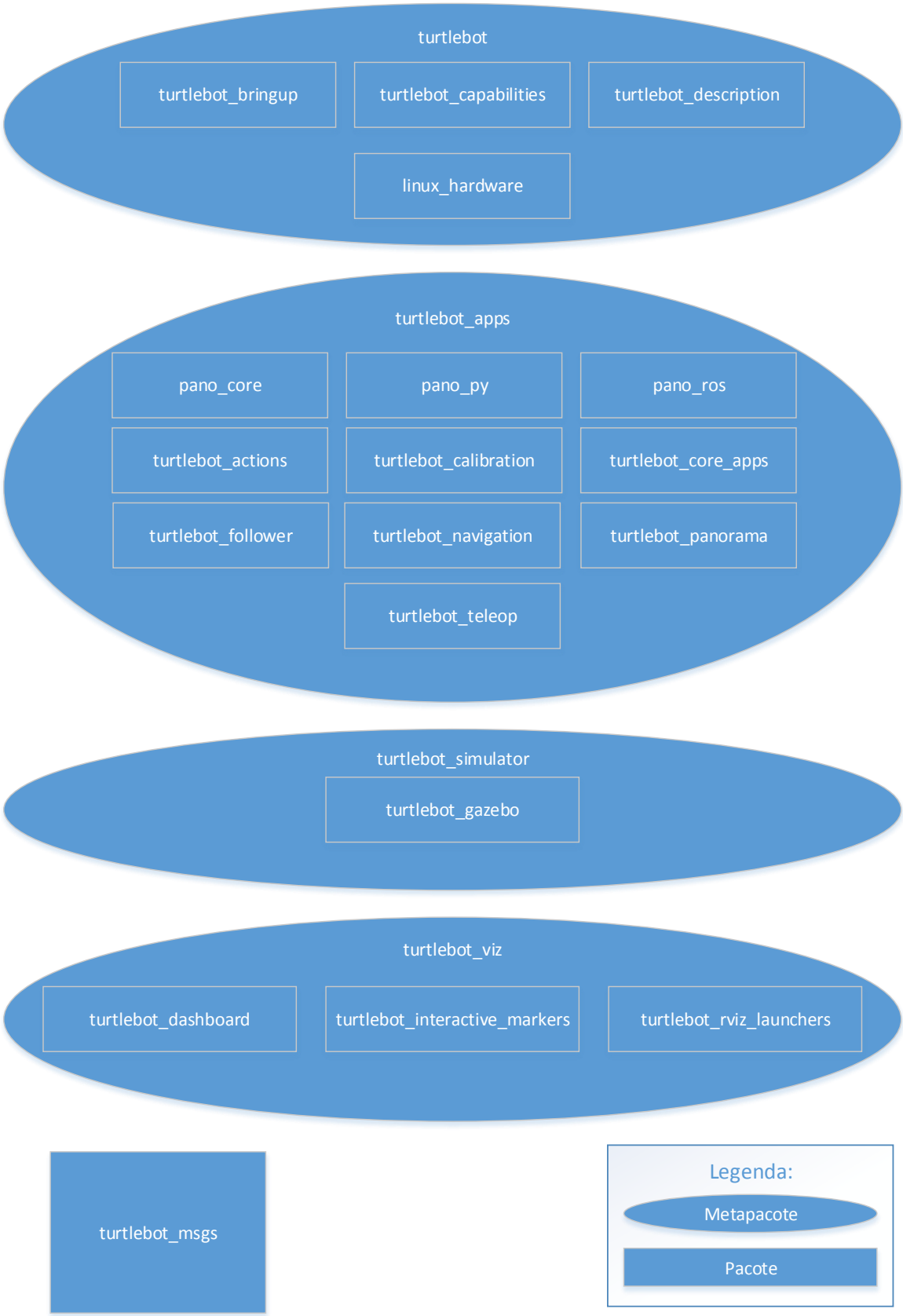

Figura 29 – Lista de pacotes do TurtleBot

Analisando a Figura 29, podemos ver que os pacotes do TurtleBot estão divididos em quatro meta-pacotes: `turtlebot`, que contém os pacotes essenciais para correr e usar o TurtleBot; `turtlebot_apps`, que implementa funções para o robô como a aplicação de teleoperação; o `turtlebot_viz`, que implementa funções de visualização do robô; e o `turtlebot_simulator`, que implementa funções necessárias para simulação do robô. Fora destes meta-pacotes temos ainda o pacote `turtlebot_msgs` que define as mensagens específicas deste robô que serão utilizadas no ROS. Os primeiros três meta-pacotes são os mais relevantes para este projeto e serão descritos em mais detalhe mais adiante.

Podemos ainda ver que depois da análise dos pacotes da Figura 29, dentro dos meta-pacotes do TurtleBot temos pacotes que não começam por `turtlebot_`. Estes pacotes, utilizados no TurtleBot e que foram desenvolvidos pelos criadores do mesmo, implementam funcionalidades necessárias para outros pacotes existentes. No caso do pacote `linux_hardware`, este implementa a leitura de informação da bateria do *netbook* e disponibilização da mesma no ROS, sendo depois utilizada pelo pacote `turtlebot_dashboard`. É de notar que este pacote em versões seguintes do ROS passou a fazer parte do meta-pacote de controladores do ROS dentro do pacote `linux_peripheral_interfaces`. No caso dos pacotes cujo nome começa por `pano_`, estes disponibilizam funções utilizadas pelo pacote `turtlebot_panorama`.

3.2.1.1 Meta-pacote `turtlebot` e seus pacotes

Como referido anteriormente, o meta-pacote `turtlebot` contém todos os controladores e informação necessária para correr e utilizar o TurtleBot. Dos pacotes dentro do mesmo, já foi explicado anteriormente o propósito e a funcionalidade do pacote `linux_hardware`, ficando apenas por explicar os pacotes `turtlebot_bringup` e `turtlebot_description`, uma vez que o pacote `turtlebot_capabilities` é apenas um *placeholder* para versões futuras do ROS.

O pacote `turtlebot_description` disponibiliza um modelo 3D completo do TurtleBot para simulação e visualização. Os ficheiros neste pacote são interpretados e utilizados por uma variedade de componentes, nomeadamente para o cálculo de

posicionamento de cada junta do robô. No ROS, todos os robôs são descritos através de juntas, utilizando para isso o formato URDF (adicionar referencia). Deste pacote interessa-nos principalmente o ficheiro `urdf/turtlebot_library.urdf.xacro`, onde iremos especificar uma junta para o LRF do nosso robô, como iremos falar mais à frente.

O pacote `turtlebot_bringup` disponibiliza os *scripts* para utilizar com o comando `roslaunch` para correr as funcionalidades base do TurtleBot. Deste pacote estamos interessados principalmente no ficheiro `launch/minimal.launch`, utilizado para iniciar a base do robô.

3.2.1.2 Meta-pacote `turtlebot_apps` e seus pacotes

Como falado anteriormente, o meta-pacote `turtlebot_apps` disponibiliza demonstrações e exemplos de aplicações para correr no TurtleBot para ajudar na iniciação e utilização do ROS e do TurtleBot. Destes pacotes, iremos apenas falar dos pacotes `turtlebot_navigation` e do pacote `turtlebot_teleop`, uma vez que os restantes têm pouco interesse no âmbito deste projeto.

O pacote `turtlebot_teleop` disponibiliza a funcionalidade de teleoperação utilizando *joysticks* ou o teclado. Deste pacote estamos interessados no ficheiro `launch/keyboard_teleop.launch`, utilizado para iniciar o nó de teleoperação utilizando o teclado (Figura 30).

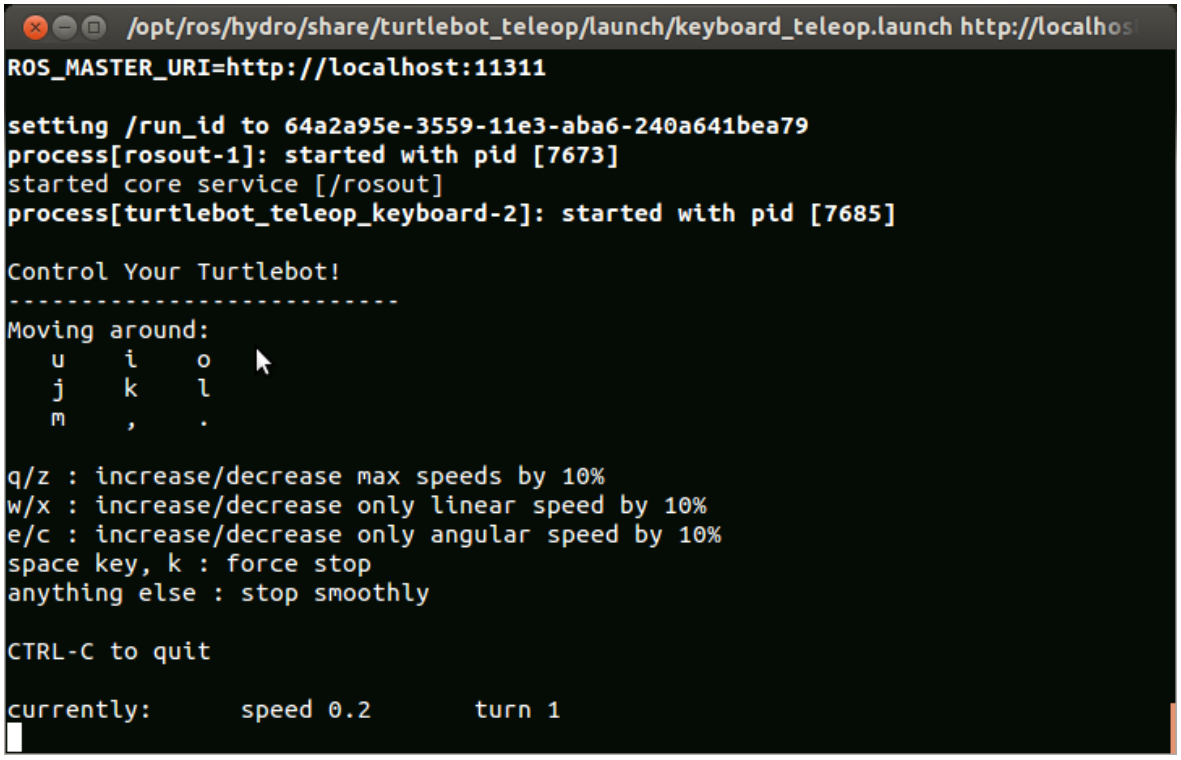

Figura 30 – Nó de teleoperação utilizando o teclado do TurtleBot

O pacote `turtlebot_navigation` disponibiliza demonstrações de como criar mapas e efetuar localização utilizando os pacotes `gmapping` e `amcl` respetivamente, enquanto é utilizada a *stack* de navegação do ROS. Este é o pacote com mais interesse neste projeto, uma vez que engloba toda a parte de navegação do robô.

3.2.1.3 Meta-pacote `turtlebot_viz` e seus pacotes

O meta-pacote `turtlebot_viz` disponibiliza todos os pacotes relacionados com a visualização e monitorização do robô. Neste meta-pacote podemos encontrar o pacote `turtlebot_dashboard` que implementa um *dashboard* onde podemos ver o estado das entradas e saídas da base robótica e outros detalhes do robô, o pacote `turtlebot_interactive_markers` que permite um comando interativo do TurtleBot utilizando marcadores interativos no RViz e o pacote `turtlebot_rviz_launchers` que contém ficheiros para visualização do robô e informação relativa ao mesmo utilizando o RViz (Figura 31). RViz é o nome da ferramenta de visualização 3D do ROS.

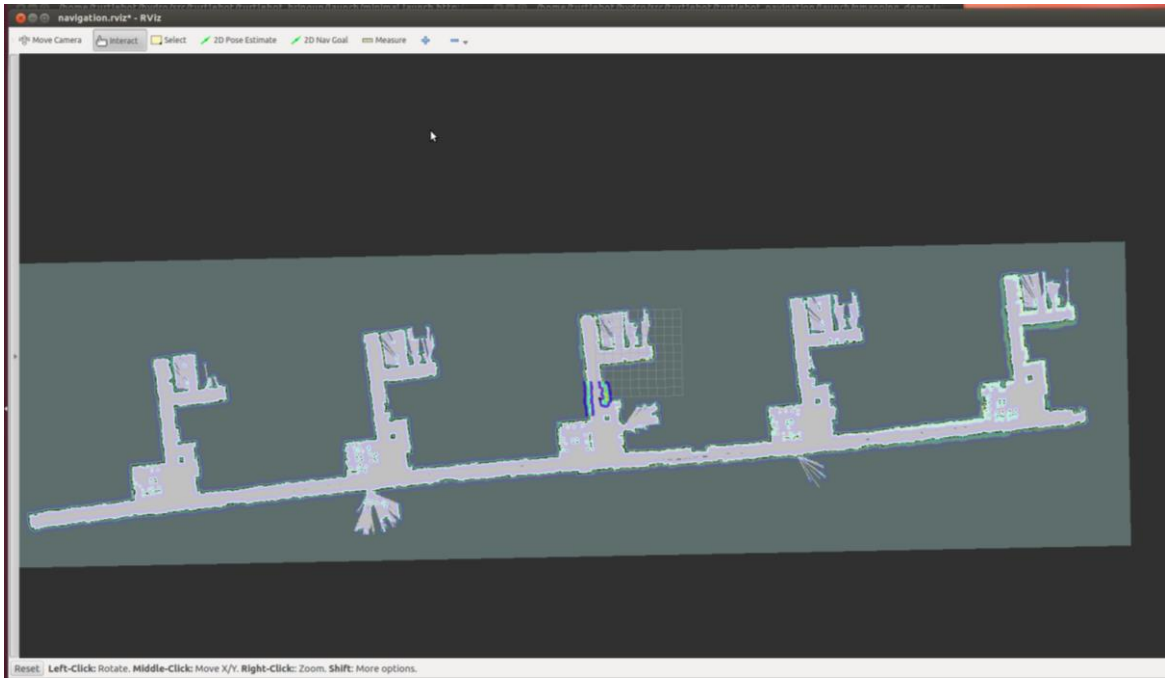

Figura 31 – Visualização do mapa do piso 1 dos edifícios G a L e mapas de custo do robô utilizando o RViz

3.2.2. Modificação dos pacotes do TurtleBot para o robô desenvolvido

A diferença principal entre o TurtleBot e o robô rececionista é a estrutura física do mesmo e a utilização de um LRF em vez de uma Kinect, pelo que faz todo o sentido usar como base os pacotes do TurtleBot e modificá-los para utilização com este robô. As alterações necessárias são as seguintes: alteração da descrição do robô e do pacote de navegação para refletir a troca da Kinect pelo LRF.

Durante os testes iniciais foram descobertos alguns problemas com a instalação dos pacotes do TurtleBot: o pacote `linux_hardware` não reportava a percentagem da bateria do *netbook* por exemplo. Foi corrigido este problema (cuja solução passou pela atualização do pacote por uma versão mais recente ainda não presente nos repositórios do Ubuntu e ajustes respetivos no pacote de *bringup*) e outros pequenos problemas encontrados e testadas as soluções utilizadas para confirmar que estava a ser utilizada uma base sólida e 100% funcional.

3.2.2.1 Alteração da descrição do robô

Apesar da estrutura física do robô, em termos funcionais o único dado relevante a colocar na descrição do robô para o funcionamento do robô é a posição do LRF, uma vez que a Kinect pode ser desativada no pacote de navegação. Como tal, foi alterado o ficheiro `urdf/turtlebot_library.urdf.xacro` apenas para incluir uma junta fixa com o nome `laser` relativa à junta base do robô, chamada `base_link` e que está posicionada no centro da projeção do robô no chão, e outra junta chamada `base_laser_link` utilizada apenas para visualizar a posição da junta laser no modelo do robô utilizando o RViz.

```
−<joint name="laser" type="fixed">
    <origin xyz="0.105 0.00 0.155" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="base_laser_link"/>
  </joint>
−<link name="base_laser_link">
  −<visual>
    −<geometry>
        <box size="0.00 0.05 0.06"/>
      </geometry>
      <material name="Green"/>
    </visual>
  −<inertial>
      <mass value="0.000001"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="0.0001" ixy="0.0" ixz="0.0" iyy="0.0001" iyz="0.0" izz="0.0001"/>
    </inertial>
  </link>
```

Figura 32 – Descrição das juntas relativas ao LRF no URDF do robô

3.2.2.2 Alteração do pacote de navegação

O TurtleBot está desenhado para utilizar uma Kinect para fazer mapas e localização do robô. Para manter a possibilidade de utilizar a Kinect para navegação no futuro, foi decidido ter ficheiros de *launch* e configuração independentes para utilização do LRF para navegação, tendo sido os mesmos colocados dentro da pasta `laser/` do pacote `turtlebot_navigation` com a Kinect desativada.

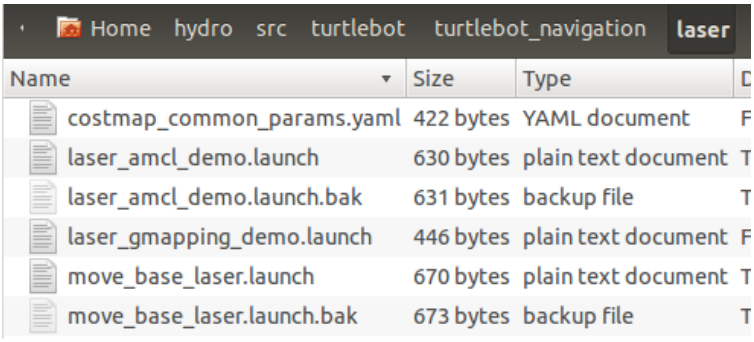

Figura 33 – Pasta `laser/` do pacote `turtlebot_navigation`

3.3. Controlador de alto nível do robô

Para fazer a ponte entre a interface gráfica do utilizador e a plataforma ROS e controlar o comportamento do robô, foi desenhado um controlador de alto nível em Python. Este controlador é um nó de ROS que recebe informação relativamente ao robô e atualiza a interface gráfica, assim como recebe comandos (objetivos de navegação, paragem de emergência, etc.) da interface gráfica e faz com que os mesmos se reflitam no estado do robô. O funcionamento do mesmo pode ser demonstrado através da Figura 34.

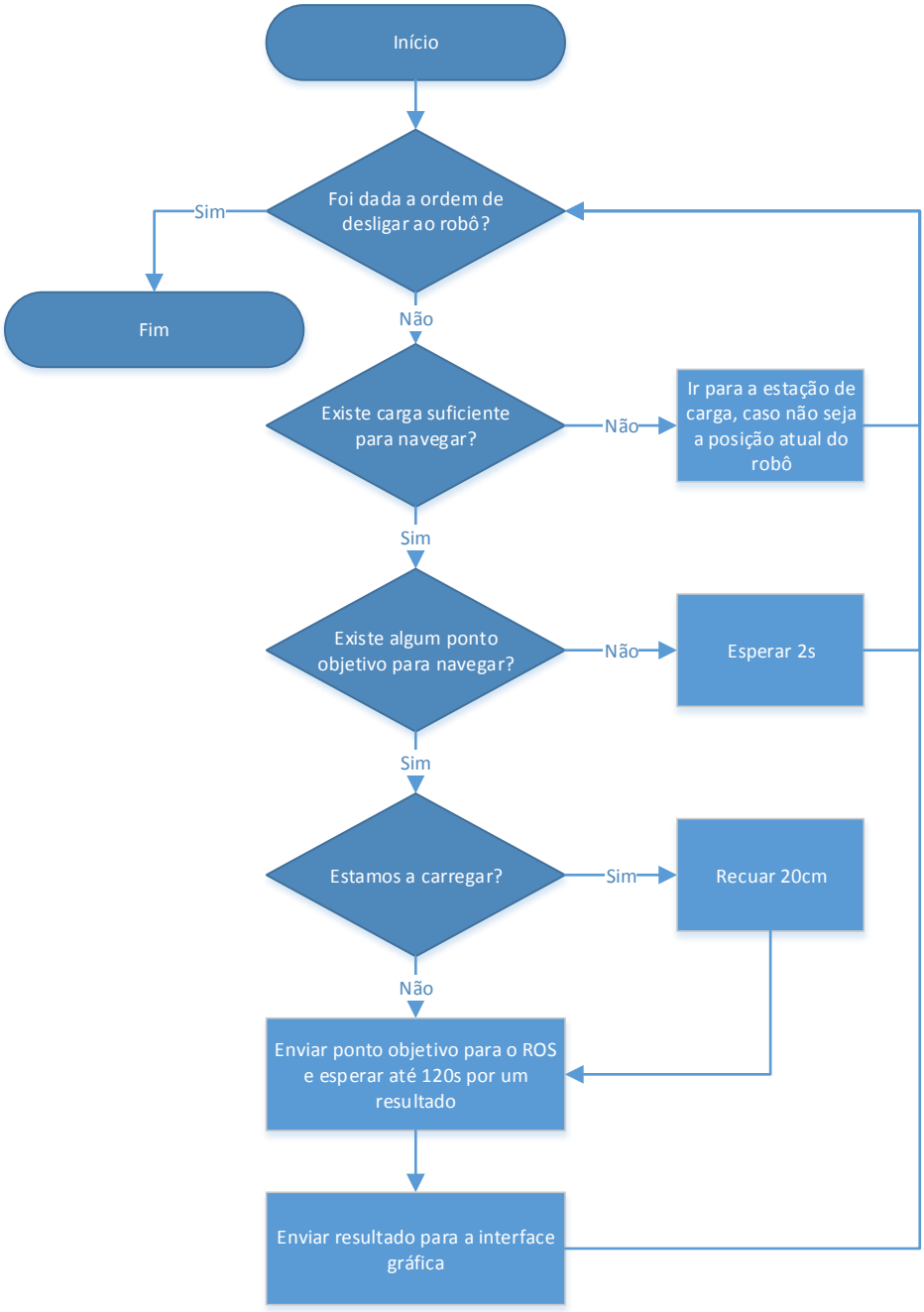

Figura 34 – Algoritmo de controlo de alto nível

3.4. Interface gráfica

Para o utilizador conseguir interagir com o robô, foi desenhada uma interface gráfica em Python utilizando a *toolkit* wxPython (baseada na *toolkit* C++ wxWidgets) para

ser utilizada através de um ecrã tátil, sendo por isso composta principalmente por poucos e grandes botões que servem para ordenar o robô para se deslocar para um ponto objetivo pré-definido e para parar a navegação. Esta interface gráfica está desenhada para funcionar com o mesmo aspeto e comportamentos em Windows, Linux e Mac.

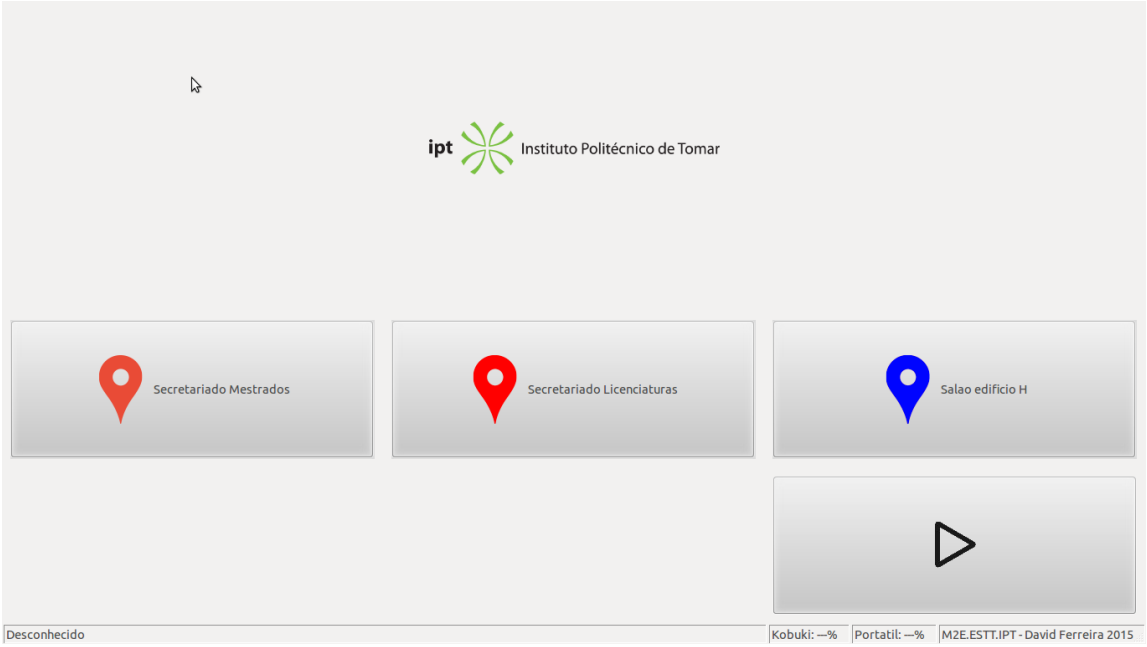

Figura 35 – Ecrã principal da interface gráfica

Na Figura 35 podemos ver os botões para nos deslocarmos para um de três pontos objetivos e um botão para nos deslocarmos para a segunda página de objetivos. Em baixo, na Figura 36, podemos ver o ecrã exibido depois de se clicar no botão que comanda o robô para seguir para o objetivo "Secretariado Licenciaturas", onde podemos clicar no botão STOP para parar a navegação. Os objetivos são guardados num ficheiro em formato JSON, onde têm um ID e associado a esse ID um nome legível e a pose do robô nesse objetivo.

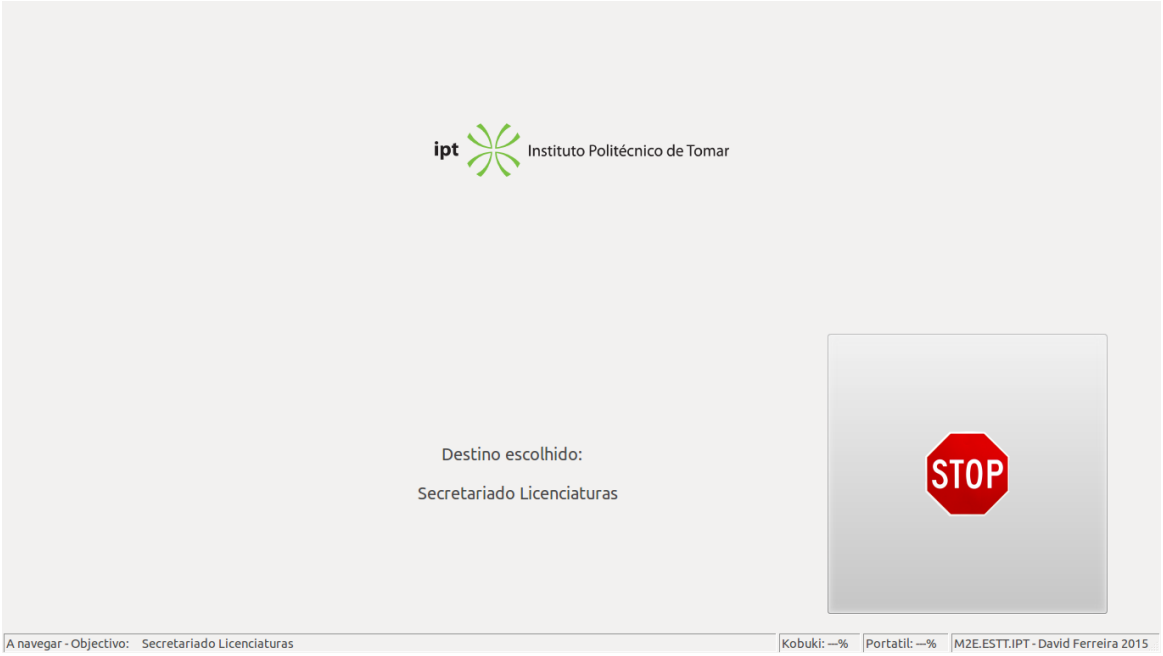

Figura 36 – Ecrã exibido durante a navegação do robô mostrando o botão de paragem de emergência

Quando o robô se encontra a carregar, é mostrado o ecrã que se pode ver na Figura 37. Se existir carga suficiente no robô, ao clicar no ícone da bateria será mostrado o ecrã principal com os objetivos de navegação existentes, caso contrário nada acontecerá.

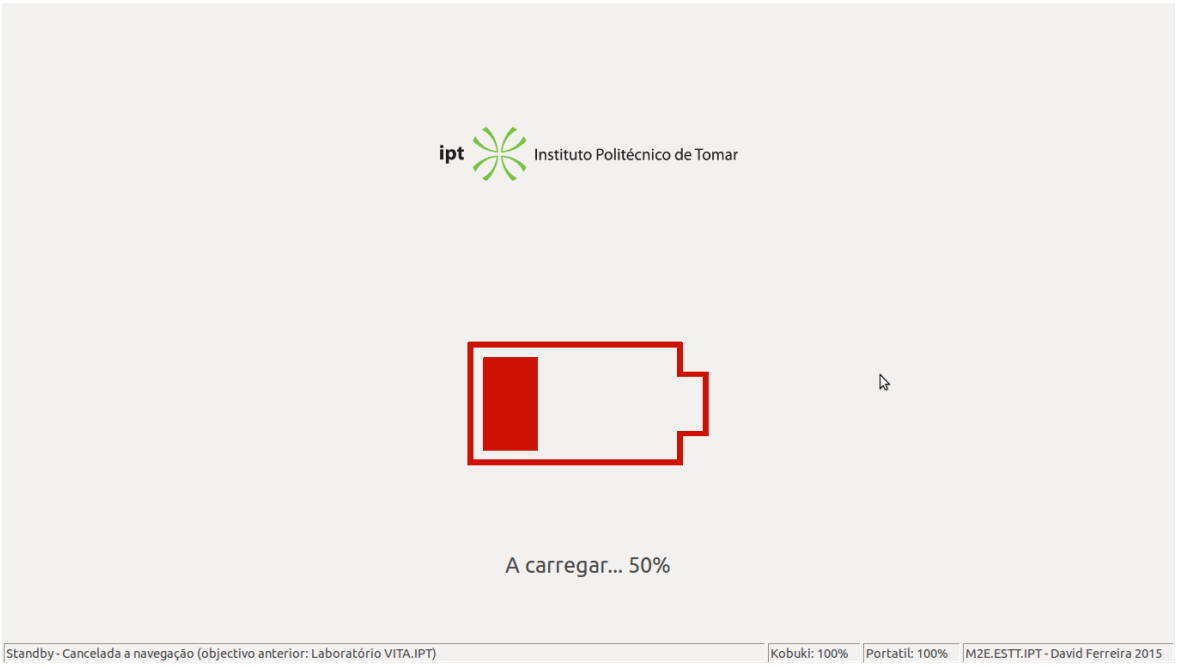

Figura 37 – Ecrã mostrado ao utilizador durante o carregamento do robô

4. SLAM

Durante as últimas duas décadas a comunidade de robótica tem mostrado um grande empenho para encontrar soluções genéricas e robustas para o problema de localização e mapeamento simultâneos (*Simultaneous Location And Mapping* – SLAM). Têm sido utilizadas diversas abordagens a este problema, regra geral com bons resultados para robôs, ambientes e requisitos específicos [25] [26]. Dado ser um problema fundamental da robótica, o SLAM tem sido abordado por muitos grupos de investigação e por esse motivo existem várias soluções de SLAM publicados e reconhecidos na comunidade de investigação. As abordagens ao problema de SLAM dependem do tipo de sensores com que o robô está equipado, o ambiente onde irá operar, requisitos e outras restrições. Efetuar SLAM com um robô não é uma solução óbvia porque para podermos localizar o robô precisamos de um mapa *a priori*, sendo necessário, por outro lado, fazer a localização do robô para mapear o ambiente. As soluções para o problema de SLAM apoiam-se fortemente em hipóteses probabilísticas: tanto a estimação de mapas como a localização de robôs são realizados através da aquisição de dados sensoriais que fazem uso uso de métodos probabilísticos que têm a capacidade de modelar ruído e representar a incerteza associada com os dados sensoriais e processos de estimação.

Para melhor compreender o problema de SLAM, é necessário dividi-lo em duas partes distintas:

- **Localização:** A localização de robôs móveis é o problema de determinar a pose (posição e orientação relativo ao referencial do mundo) de um robô no ambiente que o rodeia utilizando a informação sensorial disponível e é geralmente chamado "estimação de pose", podendo ainda ser visto como um problema de estabelecer uma correspondência entre o sistema de coordenadas do mapa e o sistema de coordenadas local do robô [27];

Os métodos de localização podem ser classificados em dois grupos principais: localização local e localização global. A localização local assume que a pose inicial do robô é conhecida e o robô apenas tem de compensar pequenos erros de odometria que ocorrem durante a navegação do mesmo. Estes métodos tipicamente não conseguem recuperar quando deixam de conseguir acompanhar a pose do robô. Na localização global, o robô é colocado algures no seu ambiente e não é conhecida a sua pose inicial no sistema

de coordenadas do mapa, sendo por isso mais complexo do que localização local. Quando um robô é capaz de se localizar globalmente, isto implica que o mesmo se consegue localizar num ambiente desconhecido e é capaz de resolver o problema denominado por sequestro do robô [27]. Neste problema o robô é literalmente pegado em braços e colocado noutro local, devendo então ser capaz de se re-localizar.

- **Mapeamento:** O mapeamento consiste no problema de representar o ambiente que rodeia o robô de forma a permitir que se efetue a localização do robô nesse ambiente.

Existem dois tipos principais de mapas que podem ser gerados e mantidos por robôs móveis: métricos e topológicos.

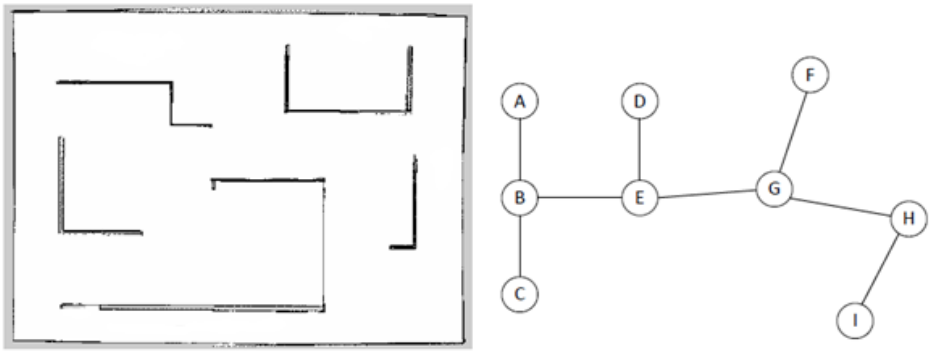

Figura 38 – Exemplo de um mapa métrico discreto (esquerda) e de um mapa topológico (direita)

Os mapas métricos são mais simples e fáceis de compreender por humanos, uma vez que representam o ambiente diretamente da informação sensorial, podendo os mesmos ser contínuos ou discretos, sendo os mapas contínuos uma interpolação do mundo real através de medições discretas. Mapas discretos, como os mapas de grelhas de ocupação, são representados por uma matriz de células, em que cada célula tem associado um valor que indica o tipo de ambiente naquele preciso lugar, podendo por exemplo ser espaço livre, um obstáculo, espaço desconhecido ou mesmo o valor da probabilidade daquela área estar ocupada. Regra geral, as técnicas de mapeamento mais utilizadas limitam-se a usar 3 estados para cada célula, representando espaço livre, ocupado ou desconhecido. A área representada por cada célula pode ser ajustada e ser tão pequena quanto desejado, estando apenas limitada pelo tipo, qualidade e quantidade de medições o quão fiel o mapa representa o mundo real, pelo que regra geral é necessário atingir um equilíbrio entre a resolução do mapa e o custo computacional de o gerar e manipular.

Os mapas topológicos, por outro lado, descrevem a conectividade de diferentes locais, fazendo uso de grafos para representar os ambientes. Nestes mapas, temos locais, ou nós, que são ligados utilizando arcos que têm anotada informação sobre como navegar entre os dois locais. Os nós num mapa topológico representam locais, características ou outras situações destintas. Estes mapas são muito mais fáceis de guardar e processar por um computador do que os mapas métricos, aumentando por isso o seu desempenho em ambientes grandes e complexos. Porém são mais difíceis de manter em termos de consistência, especialmente se estivermos num ambiente com poucos traços únicos que possamos utilizar para o representar (corredores longos e muito uniformes por exemplo).

Atualmente começa a ganhar relevância o mapeamento semântico, que consiste na criação de mapas que não só representam a ocupação do ambiente, mas também outras propriedades do ambiente, nomeadamente a identificação de espaços como portas ou corredores, entre outras propriedades que sejam relevantes para a aplicação do robô [28].

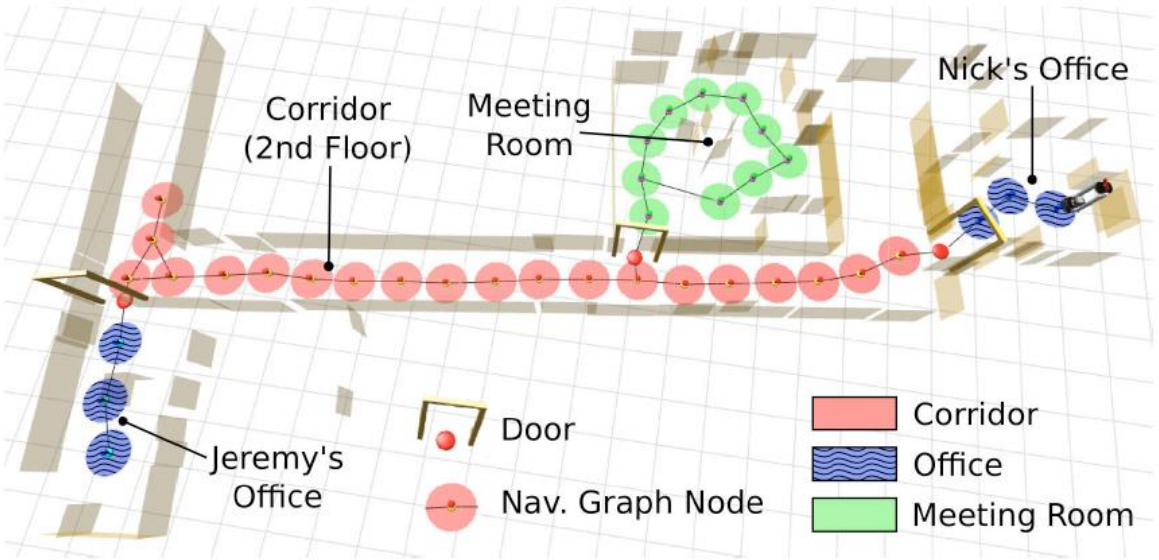

Figura 39 – Exemplo de um mapa aumentado com informação semântica

Na Figura 40 podemos ver os módulos fundamentais de um robô móvel, estando o SLAM integrado no módulo de processamento de dados onde trabalha sobre os dados disponibilizados pelo módulo de aquisição de dados (odometria e medições de LRF). A

consequente determinação da localização e mapeamento permite a implementação de métodos de tomada de decisão, nomeadamente de navegação, no módulo de decisão.

Neste projeto, utilizou-se o Gmapping para resolver o problema de SLAM.

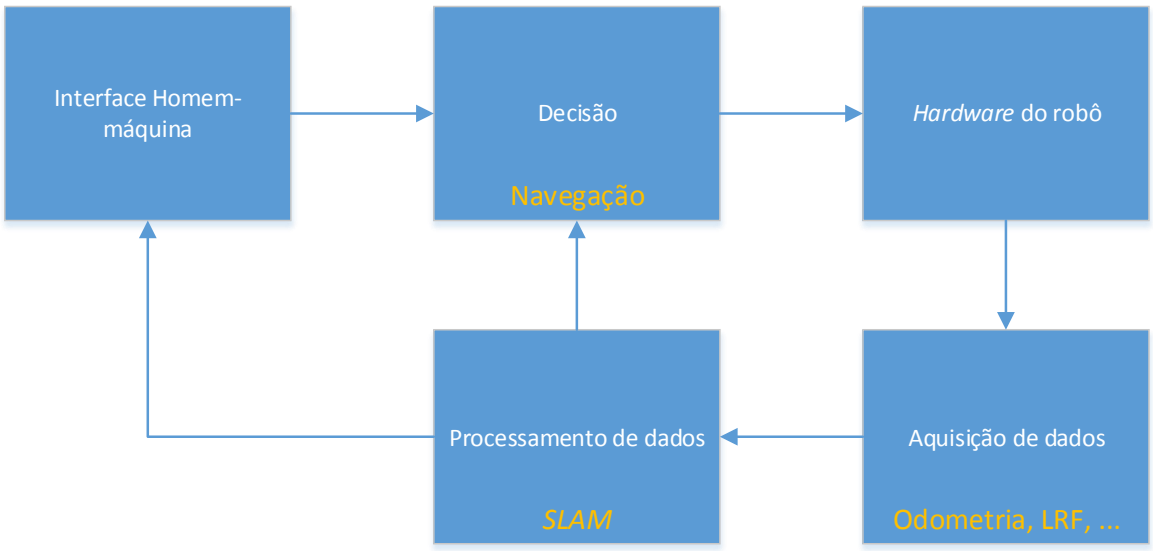

Figura 40 – Módulos fundamentais de um robô móvel, evidenciando o SLAM e suas dependências

4.1. GMapping

O GMapping [15] implementa o método FastSLAM com uma versão bastante eficiente do filtro de partículas Rao-Blackwell [29] para aprender mapas de grelha utilizando dados de um LRF e é um projeto desenvolvido em C++ pela comunidade OpenSLAM que foi integrado num pacote ROS. Esta implementação toma em conta não só o movimento do robô mas também a última observação do ambiente, reduzindo a incerteza relativamente à pose do robô na etapa de predição do filtro.

O método FastSLAM utiliza um filtro de partículas Rao-Blackwell e vários esquemas para reamostragem de partículas. Este método guarda várias localizações e hipóteses de mapa como partículas individuais e atribui pesos a essas partículas baseados no grau de coincidência dessas partículas com as observações efetuadas (*matching*).A

realização do SLAM com base em filtros de partículas evita a linearização e complexidade computacional do SLAM baseado no filtro de Kalman extendido [30].

Num filtro de partículas, uma determinada partícula está associada a um mapa de grelha de ocupação e um mapa topológico, sendo ambos atualizados enquanto o robô explora o ambiente que o rodeia. No mapa topológico, os nós representam as posições visitadas pelo robô e os arcos representam a trajetória correspondente a essa partícula. Para construir o mapa topológico, um nó inicial é considerado como a pose inicial do robô. A cada t segundos um novo nó é adicionado ao mapa topológico se a distância entre a pose atual e todos os outros nós do mapa topológico excede um valor pré-determinado ou se nenhum dos nós são visíveis a partir da pose atual. Sempre que um nó é criado, é também adicionado um arco deste nó para o último nó visitado. A cada partícula é então atribuído um peso baseado numa função de relevância que atribui um peso ω, normalizando para a soma unitária, a cada partícula. As partículas são então reamostradas com substituição, onde a probabilidade de seleção para remoção é proporcional ao peso ω, sendo as restantes partículas depois atualizadas, permitindo que o filtro de partículas armazene diversas hipóteses e as utilize como for necessário. Na Figura 41 podemos ver a interpretação geométrica de SLAM através de uma série de medições z_i da distância e orientação das referências m_i relativamente à posição x_i, podendo depois ser determinada a posição das referências utilizando o espaço geométrico [31].

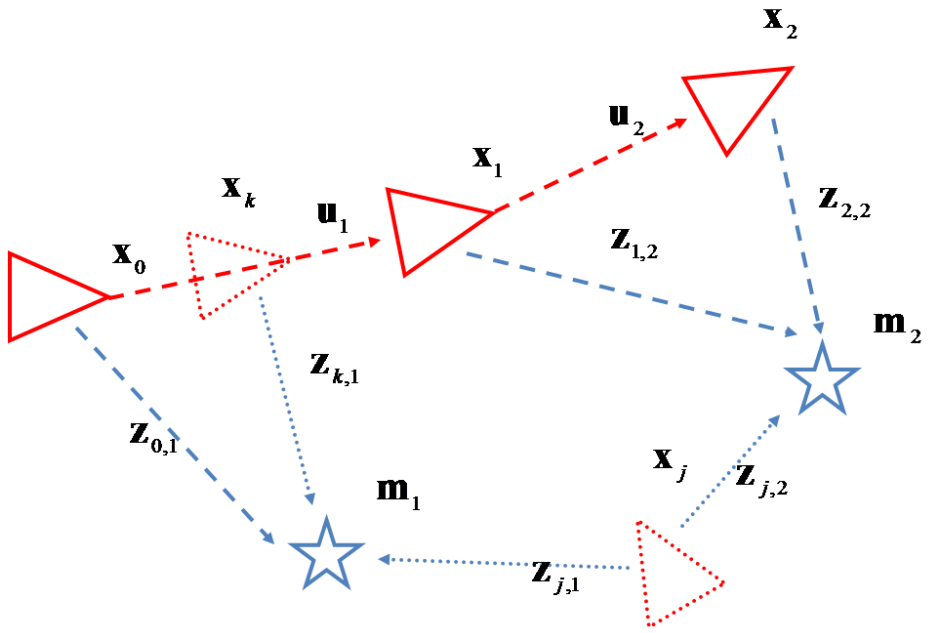

Figura 41 – Interpretação geometrica de SLAM: são efetuadas as medições z da distância e orientação das referências m relativamente às posições x

Este método de SLAM escala favoravelmente para um número elevado de pontos de referência, funcionando bem com ambientes diversificados, mas é computacionalmente pesado se o número de partículas for elevado e tem uma estimativa demasiado otimista da pose do robô.

4.2. Localização de Monte Carlo

O módulo `amcl` do ROS, utilizado neste robô, implementa uma abordagem adaptativa da localização de Monte Carlo. Ao contrário de outros métodos que descrevem a função de densidade da probabilidade para representar a incerteza, este método representa a incerteza através de um conjunto de amostras que são tiradas de forma aleatória da função referida [32]. Para atualizar esta representação de densidade ao longo do tempo, são utilizados os métodos de Monte Carlo que foram inventados nos anos 70. Ao utilizar uma representação baseada em amostras, é possível obter um método de localização que tem diversas vantagens relativamente a métodos anteriores, nomeadamente: reduz de forma considerável a quantidade de memória necessária

relativamente à localização de Markov (a localização de Markov mantém uma distribuição probabilística sobre o espaço de todas as hipóteses da localização do robô no mundo), e pode integrar medições a uma frequência consideravelmente mais elevada; é mais precisa do que a localização de Markov que se encontra sujeito a tamanho de célula fixa, uma vez que o estado representado nas amostras não é discretizado; e é de fácil implementação.

4.3. Modelo cinemático, odometria e laser

Este robô é um robô circular diferencial, onde existem 2 rodas motrizes ao longo do mesmo eixo, que rodam e são controladas independentemente.

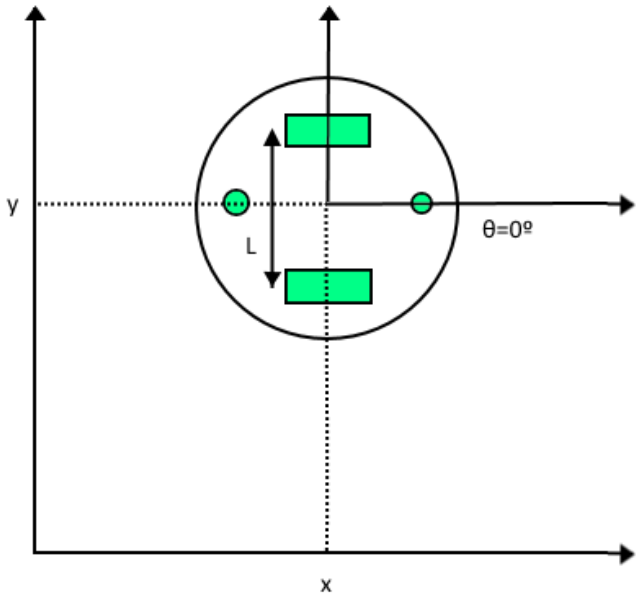

Figura 42 – Esquema de um robô diferencial

Cada roda (identificadas com o sufixo l e r para a roda esquerda e direita, respetivamente) tem um determinado raio r e está afastada a outra roda L m, medidos ao centro da roda, estando no ponto médio entre as duas rodas o centro robô, relativamente ao qual se caracterizam os movimentos do robô no mundo. O modelo cinemático deste tipo de robô é bastante conhecido, e os movimentos do robô podem ser descritos relativamente ao

robô ou ao mundo [33]. Nas equações abaixo podemos ver o modelo cinemático do robô relativamente ao mesmo e ao mundo, respetivamente:

$$\begin{bmatrix} v_x(t) \\ v_y(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{L} & \frac{r}{L} \end{bmatrix} \begin{bmatrix} \omega_l(t) \\ \omega_r(t) \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos\theta(t) & 0 \\ \sin\theta(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}$$

Nas equações acima, encontram-se representadas por $\omega_l(t)$ e $\omega_r(t)$ as velocidades angulares da roda esquerda e direita respetivamente, por $v_x(t)$, $v_y(t)$ e $\dot{\theta}(t)$ as velocidades ao longo dos eixos x e y e orientação relativa ao eixo z no referencial do robô, por $\dot{x}(t)$, $\dot{y}(t)$ e $\dot{\theta}(t)$ as velocidades ao longo dos eixos x e y e orientação relativa ao eixo z no referencial do mundo e por $v(t)$ e $\omega(t)$ as velocidades linear e angular no centro de massa do robô.

Como podemos ver nas equações acima, a velocidade do robô pode ser calculada a partir da medição da velocidade de rotação rodas motrizes e do seu raio. No entanto, podem existir irregularidades nas rodas e nos codificadores óticos, o que afeta a correção das medidas, sendo necessário por vezes tomar em conta estes problemas, efetuando uma correção das medidas. Para isso é necessário efetuar a calibração da odometria através de um processo, que no caso do TurtleBot, se baseia na utilização de uma parede direita e de um LRF para calcular a diferença entre a orientação do robô calculada através dos dados do LRF e da odometria. Este processo é realizado utilizando o pacote `turtlebot_calibration` do ROS, onde o robô é previamente alinhado com a parede. Faz-se uma primeira leitura do LRF seguindo-se uma rotação completa do robô (assumindo que os dados de odometria se encontram corretos), ao que se sucede uma segunda leitura do LRF. Utilizando as duas leituras do LRF é calculado o ângulo real da rotação robô, sendo este procedimento repetido várias vezes para diversas velocidades, uma vez que o erro poderá variar com a velocidade dos motores, e eventualmente

diferentes pisos, de forma a que seja devolvido um fator de correção que minimize os erros de odometria para a generalidade das situações.

O modelo do laser utilizado foi o denominado por campo de verosimilhança (*likelihood field*). Ao contrário de outros modelos, tais como o modelo de raio, este modelo não tem uma explicação física plausível, utilizando um algoritmo *ad hoc* que não computa uma probabilidade condicional relativa a um modelo físico do sensor, mas que no entanto funciona bem e produz resultados mais refinados mesmo em espaços congestionados, em especial quando comparado com o modelo de raio e tem uma computação tipicamente mais eficiente [27]. Sucintamente, este algoritmo determina a probabilidade da medida efetuada da seguinte forma (ver algoritmo na Figura 43): na linha 4 verifica-se se a leitura do sensor z_t^k corresponde ao seu valor máximo, sendo esta ignorada caso tal se verifique. Nas linhas 5 a 8 determina-se a distância $dist$ ao obstáculo mais próximo no mapa para uma dada posição do robô e determina-se a probabilidade q da distância medida. Em particular, na linha 8 determina-se esta probabilidade tendo por base a distância esperada. O cálculo da probabilidade da distância medida utiliza uma distribuição Gaussiana centrada em zero, $prob(dist^2, \sigma_{hit}^2)$.

1: **Algorithm likelihood_field_range_finder_model**(z_t, x_t, m)**:**

2: $q = 1$

3: *for all* k *do*

4: *if* $z_t^k \neq z_{\max}$

5: $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$

6: $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$

7: $dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \;\middle|\; \langle x', y' \rangle \text{ occupied in } m \right\}$

8: $q = q \cdot \left(z_{\text{hit}} \cdot \text{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$

9: *return* q

Figura 43 – Algoritmo do modelo de campo de verosimilhança [27]

4.4. Implementação em ROS e parâmetros de configuração

Para fazer mapeamento utilizando o gmapping neste robô, temos de executar o comando `roslaunch turtlebot_navigation gmapping_demo.launch`. Este ficheiro coloca em funcionamento o pacote `move_base` e os controladores de segurança do robô, para além do gmapping. A configuração do gmapping encontra-se no ficheiro `launch/includes/gmapping.launch.xml` do pacote `turtlebot_navigation`, e os valores utilizados para os parâmetros do gmapping são essencialmente os por defeito, excetuando-se o parâmetro `maxRange` que faz o corte das leituras do laser aos 8 metros e os parâmetros `linearUpdate` e `angularUpdate` que quantifica o movimento necessário antes de processar um novo scan para 0,5m de movimento linear e 0,436rad de movimento angular. A Figura 44 mostra a totalidade dos parâmetros do gmapping utilizados e os mesmos encontram-se explicados na ROS Wiki [34].

```
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="base_footprint"/>
    <param name="odom_frame" value="odom"/>
    <param name="map_update_interval" value="5.0"/>
    <param name="maxUrange" value="6.0"/>
    <param name="maxRange" value="8.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.5"/>
    <param name="angularUpdate" value="0.436"/>
    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="80"/>

    <param name="xmin" value="-1.0"/>
    <param name="ymin" value="-1.0"/>
    <param name="xmax" value="1.0"/>
    <param name="ymax" value="1.0"/>

    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>
/launch>
```

Figura 44 – Ficheiro com os parâmetros do gmapping

No fim de executar o comando já referido, é necessário navegar o robô pelo ambiente, utilizando tipicamente teleoperação, e no fim guardar o mapa utilizando o comando `rosrun map_server map_saver -f /caminho/para/o/mapa`, sendo criados os ficheiros mapa.pgm e mapa.yaml. O ficheiro PGM é um ficheiro de imagem em escala de cinzentos em que cada pixel tem o valor relativo ao estado de ocupação de cada célula do mapa. O ficheiro YAML tem os dados relativos ao mapa, tais como o nome do ficheiro com o mapa (o ficheiro PGM), a resolução do mapa em metros por pixel, a pose

do ponto de origem do mapa como (x,y,θ), o valor a partir do qual a célula se deve considerar como ocupada, o valor máximo até ao qual a célula se pode considerar livre e se a lógica do mapa deverá ser invertida (se o branco passa a ser significar célula ocupada e o preto uma célula livre, ou vice-versa).

O módulo amcl, utilizando na navegação do robô, é configurado no ficheiro `launch/includes/amcl.launch.xml` do pacote `turtlebot_navigation`. Os detalhes de configuração foram alterados para funcionar com os tópicos ROS utilizados, sendo os valores dos restantes parâmetros os definidos por defeito (Figura 45).

```
<param name="odom_model_type"           value="diff"/>
<param name="odom_alpha5"               value="0.1"/>
<param name="gui_publish_rate"          value="10.0"/>
<param name="laser_max_beams"            value="60"/>
<param name="laser_max_range"           value="12.0"/>
<param name="min_particles"             value="0"/>
<param name="max_particles"             value="10000"/>
<param name="kld_err"                   value="0.05"/>
<param name="kld_z"                     value="0.99"/>
<param name="odom_alpha1"               value="0.2"/>
<param name="odom_alpha2"               value="0.2"/>
<!-- translation std dev, m -->
<param name="odom_alpha3"               value="0.2"/>
<param name="odom_alpha4"               value="0.2"/>
<param name="laser_z_hit"               value="0.5"/>
<param name="laser_z_short"             value="0.05"/>
<param name="laser_z_max"               value="0.05"/>
<param name="laser_z_rand"              value="0.5"/>
<param name="laser_sigma_hit"           value="0.2"/>
<param name="laser_lambda_short"        value="0.1"/>
<param name="laser_model_type"          value="likelihood_field"/>
<!-- <param name="laser_model_type" value="beam"/> -->
<param name="laser_likelihood_max_dist" value="2.0"/>
<param name="update_min_d"              value="0.25"/>
<param name="update_min_a"              value="0.2"/>
<param name="odom_frame_id"             value="odom"/>
<param name="base_frame_id"             value="base_footprint"/>
<param name="resample_interval"         value="1"/>
<!-- Increase tolerance because the computer can get quite busy -->
<param name="transform_tolerance"       value="1.0"/>
<param name="recovery_alpha_slow"       value="0.0"/>
<param name="recovery_alpha_fast"       value="0.0"/>
<param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
<param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
<param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
<remap from="scan"                      to="$(arg scan_topic)"/>
```

Figura 45 – Parâmetros do módulo amcl

5. Sistema de navegação

Uma das funcionalidades que se espera de um robô móvel é que o mesmo seja capaz de se movimentar e, no caso deste projeto, que a sua movimentação seja autónoma para atingir os pontos objetivo dados pelo utilizador. Para atingir um ponto objetivo, há a necessidade de resolver o problema de planeamento de trajetória. O problema de planeamento de trajetória é definido como, dado um determinado robô móvel, um mapa, uma posição inicial e um ponto objetivo, a tarefa de planear uma trajetória livre de colisões que é válida, atingível e preferencialmente que seja geometricamente ótima que o dado robô consiga seguir desde a posição inicial para o ponto objetivo. Este problema costuma ser divido em duas tarefas: planeamento global e planeamento local. O planeamento global deve gerar uma trajetória aproximada, de alto nível, entre a posição inicial e o objetivo. Por outro lado, o planeamento local deve produzir uma trajetória de baixo nível que resolve um segmento da trajetória global e evita obstáculos que sejam detetados.

As soluções do problema de planeamento global são geralmente baseados em algoritmos de procura em grafo e árvores como o A*, Dijkstra, entre outros. Estes algoritmos são muito eficientes a nível computacional e encontram a solução ótima teórica para um dado mapa de custo. Existem também implementações utilizando D*, um algoritmo baseado no A* mas que permite custos dinâmicos na árvore de procura, permitindo melhor execução em ambientes dinâmicos ou desconhecidos [35].

O planeamento local é essencialmente um sistema reativo baseado em informação sensorial que tem como objetivo evitar colisões do robô como obstáculos. As soluções mais comuns passam por, como referido no capítulo 2.4.2. , metodologias de janela dinâmica (*dynamic window approach* – DWA) ou *trajectory rollout*, sendo o DWA mais eficiente. Para evitar esforços computacionais muito elevados, têm sido propostos métodos aleatórios como o *rapidly-exploring randomizing tree* [36].

No entanto, o planeamento de trajetória por si só não consegue levar o robô da sua posição inicial para o ponto objetivo: é necessário existir um sistema que consiga gerir todo o fluxo de informação necessária para navegar, nomeadamente a receção do ponto objetivo do utilizador, informações de odometria, leituras de LRF e mapa do ambiente e interligar todos os intervenientes no processo de navegação, nomeadamente os planeadores

global e local. Para realizar essa função neste projeto, foi utilizado o pacote `move_base` do ROS, como irá ser explicado mais à frente, sendo o mesmo o núcleo do sistema de navegação do robô.

5.1. Arquitetura de alto nível do sistema navegação do robô

O sistema de navegação é composto não só pelo pacote `move_base` e os seus componentes, mas também pelo robô em si como base motorizada e pela interface de interação do utilizador. Na Figura 46 podemos ver o relacionamento entre os diversos componentes do sistema de navegação do robô e as camadas do mesmo. O controlador do robô é composto pelo pacote `move_base` e pelo controlador de alto nível desenvolvido em Python.

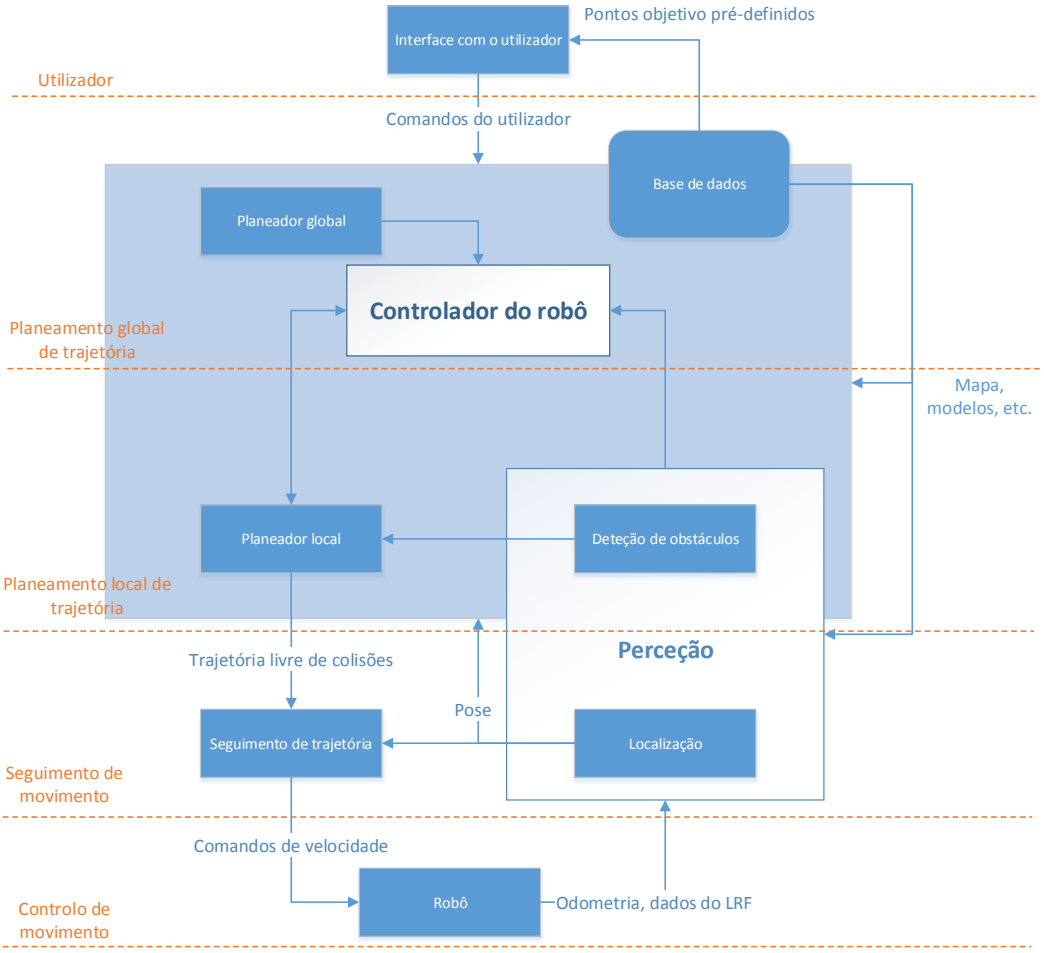

Figura 46 – Arquitetura de navegação do robô e as camadas do mesmo

Podemos ainda descrever resumidamente cada um dos módulos presentes na figura anterior e especificar as suas entradas e saídas:

- Interface com o utilizador: apresenta uma interface para o utilizador interagir com o robô e tem como saída um ponto objetivo para o robô atingir;
- Planeador global: efetua o planeamento global do robô. Tem como entradas o mapa de custos global e as poses inicial e objetivo do robô e como saída a trajetória que o robô deverá seguir para atingir a sua pose objetivo;
- Planeador local: efetua o planeamento local do robô. Tem como entradas o mapa de custos local e a pose atual do robô, e como saída um conjunto de velocidades linear e angular necessário para atingir a pose objetivo sem existirem colisões;
- Seguidor de trajetória: segue a trajetória do robô, comparando-a com a esperada e dando os comandos de velocidade ao robô de forma ao mesmo seguir a trajetória calculada pelo planeador. Tem como entradas a trajetória computada pelo planeador local e a pose atual do robô e como saída um comando de velocidade para o robô;
- Módulo de perceção: utiliza os dados de odometria e do LRF do robô para localizar o robô e detetar obstáculos. Tem como entrada a odometria do robô e as leituras do LRF, e como saída a pose do robô.

5.2. Arquitetura do pacote `move_base`

O pacote `move_base` disponibiliza um nó ROS que serve de interface para configurar, correr e interagir com a *stack* de navegação num robô. Na Figura 47 podemos ter uma visão de alto-nível da arquitetura do pacote, onde os nós em azul são nós que têm de ser configurados e desenvolvidos para cada robô onde este pacote é utilizado [37].

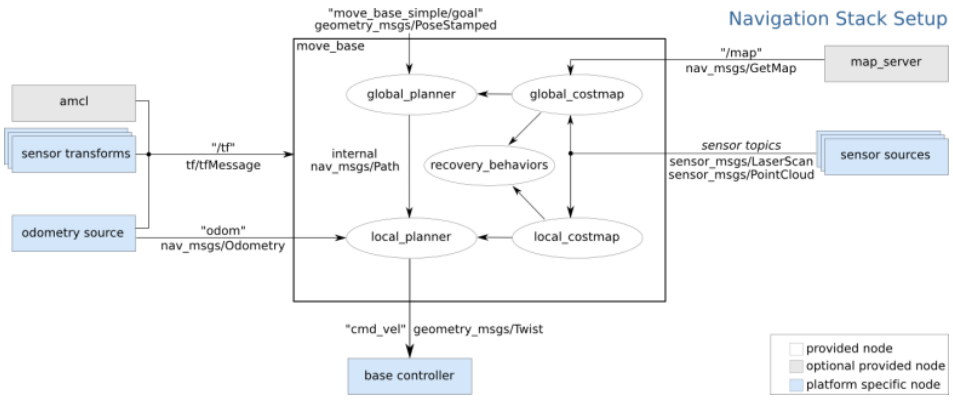

Figura 47 – Visão de alto nível do pacote `move_base`

Correr este nó num robô que se encontra configurado corretamente resulta num robô que irá tentar atingir uma pose objetivo dentro da sua base dentro de uma tolerância especificada pelo utilizador. Se não existirem obstáculos dinâmicos, o robô irá eventualmente atingir a pose objetivo ou irá sinalizar a falha ao utilizador. Este módulo tem ainda a capacidade de ter comportamentos de recuperação quando o robô considera que está preso. Por defeito, os comportamentos de recuperação são os que se podem ver na Figura 48.

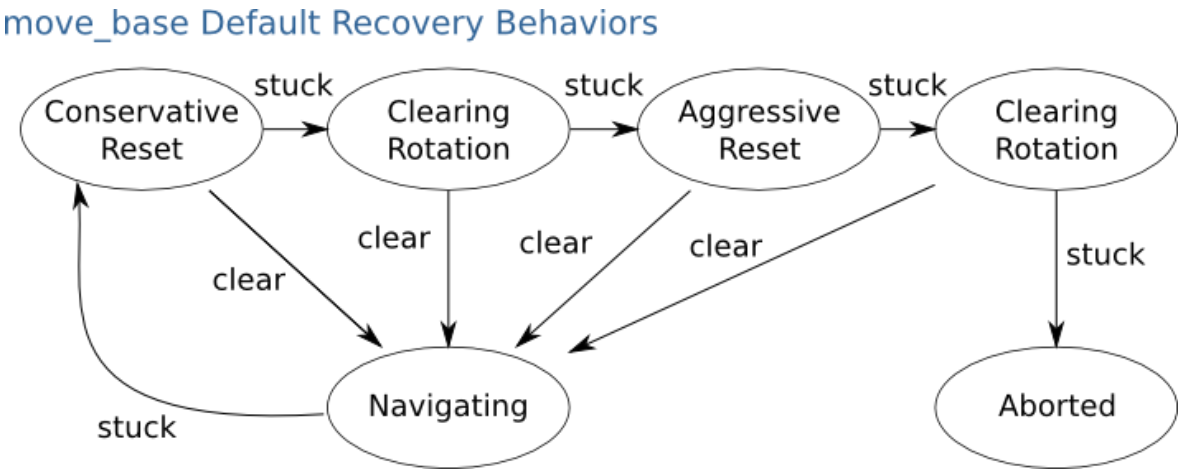

Figura 48 – Comportamentos de recuperação do robô por defeito do pacote `move_base`

Quando o robô se encontra preso, o mesmo começa por limpar os obstáculos dentro de uma região especificada pelo utilizador à volta do robô. Se ainda assim o robô se encontrar preso, o robô irá efetuar uma rotação sobre si mesmo para atualizar o seu mapa de custos local, na esperança de o obstáculo que o levou a ficar preso tenha deixado de existir. Se a obstrução se manter, é efetuada uma limpeza mais agressiva do mapa de custos, e se a mesma não resolver a situação, volta a rodar sobre si mesmo e atualizar o mapa à sua volta, e aborta a navegação caso ainda assim fique preso. Estes comportamentos podem ser configurados ou mesmos desativados.

Este pacote utiliza outros pacotes ROS dentro do meta-pacote `navigation`, nomeadamente: `costmap_2d`, que disponibiliza funcionalidades relacionadas com mapas de custo; `nav_core`, que disponibilizada as interfaces de planeador global, local e de comportamento de recuperação; `base_local_planner`, que disponibiliza um planeador local; `nav_fn`, que disponibiliza uma função de navegação interpolada rápida, computada

com o algoritmo de Dijkstra, ou seja o planeador global; `clear_costmap_recovery` e `rotate_recovery`, que disponibilizam os mecanismos de recuperação do robô.

5.3. Planeador global utilizado: Dijkstra

Para realizar o planeamento global neste projeto foi utilizado o módulo de ROS `navfn`. Este módulo implementa o algoritmo de Dijkstra para resolução do problema de planeamento global. O algoritmo de Dijkstra é um algoritmo de procura em grafo que resolve o problema da trajetória mais cura através da computação da árvore de trajetória mais curta. Entre outras aplicações, é utilizado para o planeamento de trajetória em robôs se existir e estiver disponível um mapa do mundo (seja em forma de grelha ou grafo).

O algoritmo assume os seguintes prossupostos:

- Existe um nó inicial A que define onde começa a árvore de procura e o custo do caminho até ao nó N é a distância entre os dois nós;
- Existe um valor ternário para cada nó que define se esse nó já foi visitado, ainda não foi visitado ou se é o nó atual no processo de procura;
- É definida uma estrutura que permite a listagem de nós com determinadas características num conjunto;
- Existe uma função que retorna o conjunto de nós imediatamente ligados a um nó.

A partir destes prossupostos podemos então descrever o algoritmo da seguinte maneira:

1. Definir a distância de A como zero, e a dos restantes como infinito (distância ainda não conhecida);
2. Marcar todos os nós como não visitados e marcar o nó A como o atual. Criar uma listagem de todos os nós não visitados;
3. Obter, para o nó atual, um conjunto de nós adjacentes que ainda não foram visitados. Calcular as distâncias para os nós neste conjunto. Comparar a

distância do nó atual com a distância de cada um dos nós do conjunto e detetar a menor diferença;

4. Marcar o nó com a menor diferença como o nó atual e mudar o seu estado de visitado. Se o nó de menor diferença já tiver sido visitado, termina-se o algoritmo uma vez que não existem soluções possíveis. Se o nó selecionado é o nó N, terminar o algoritmo uma vez que uma solução foi encontrada. Caso contrário, continuar o algoritmo indo para o ponto 3.

5.4. Planeador local utilizado: DWA

A abordagem de janela dinâmica para evitar obstáculos é um método que resolve esse problema através de uma amostragem discreta do espaço de controlo (velocidade linear e angular). Começamos com um espaço de controlo U, em que cada par velocidade linear-velocidade angular deverá encontrar-se dentro das velocidades mínimas e máximas permitidas do robô. Da mesma maneira, para um determinado período T do ciclo de controlo e velocidades atuais do robô, existe um espaço de controlo U_R que contém as velocidades atingíveis dentro do período T tendo conta as acelerações lineares e angulares mínimas e máximas permitidas ($a_{v_{min}}, a_{v_{max}}, a_{\omega_{min}}, a_{\omega_{max}}$). Finalmente, para cada conjunto de velocidade linear e angular, são conhecidas as distâncias linear (d_{obs}) e angular (θ_{obs}) para um obstáculo, existindo então um espaço de controlo que define o conjunto de controlos admissíveis U_A da seguinte forma:

$$U_A = \left\{\left(v \leq \sqrt{2d_{obs}a_{v_{min}}}, \omega \leq \sqrt{2\theta_{obs}a_{\omega_{min}}}\right)\right\}$$

Efetuando a interceção dos espaços U, U_R e U_A, obtemos um conjunto de controlos candidatos U_C que contém todos os controlos de velocidade que obedecem às restrições de velocidade e aceleração do robô sem levar o robô a colisões até à próxima iteração do ciclo de controlo. O próximo passo é selecionar um par de controlo u do espaço U_C que maximiza uma função objetivo. Dadas as funções de custo $f(u)$, que dá preferência a

controlos que se aproximam do ponto objetivo, $g(u)$, que dá preferência a controlos que se afastam de obstáculos, e $h(u)$, que dá preferência a determinado conjunto de velocidades, é escolhido o par de controlo do espaço U_C que maximiza uma função objetivo cujo valor é a soma dos valores devolvidos pelas funções de custo, tendo cada uma um peso α na função objetivo:

$$G(u) = \alpha_1 f(u) + \alpha_2 g(u) + \alpha_3 h(u)$$

Neste robô, os pesos α utilizados foram os seguintes:

$$\begin{cases} \alpha_1 = 0.8 \\ \alpha_2 = 0.02 \\ \alpha_3 = 0.6 \end{cases}$$

5.5. Mapas de custo

Para ser possível efetuar planeamento de trajetórias, é necessário um mapa de custos associado ao mapa do ambiente onde o robô se insere. No ROS, esta funcionalidade é implementa pelo módulo `costmap_2d`.

No pacote ROS referido acima, o mapa de custos toma a forma de uma grelha de ocupação que se sobrepõe ao mapa do ambiente e cada célula tem o custo associado a atravessar essa área no mapa do ambiente. Durante todo o planeamento de trajetória, considera-se que o robô ocupa só e apenas uma célula do mapa, negligenciando as suas dimensões espaciais, o que nos obriga a efetuar a insuflação dos obstáculos nos mapas de custo de forma a entrar em linha de conta com as dimensões do robô, evitando assim colisões. Por esta razão, as células do mapa de custos neste pacote podem tomar valores entre 0 (espaço vazio) e 254 (espaço ocupado), sendo os valores intermédios utilizados para indicar que, apesar da área representada pela célula não estar ocupada, navegar através daquela área poderá resultar numa colisão com um obstáculo. Na Figura 49, podemos ver o exemplo da *footprint* de um robô e os valores do mapa de custo quando é feita a insuflação dos obstáculos [21].

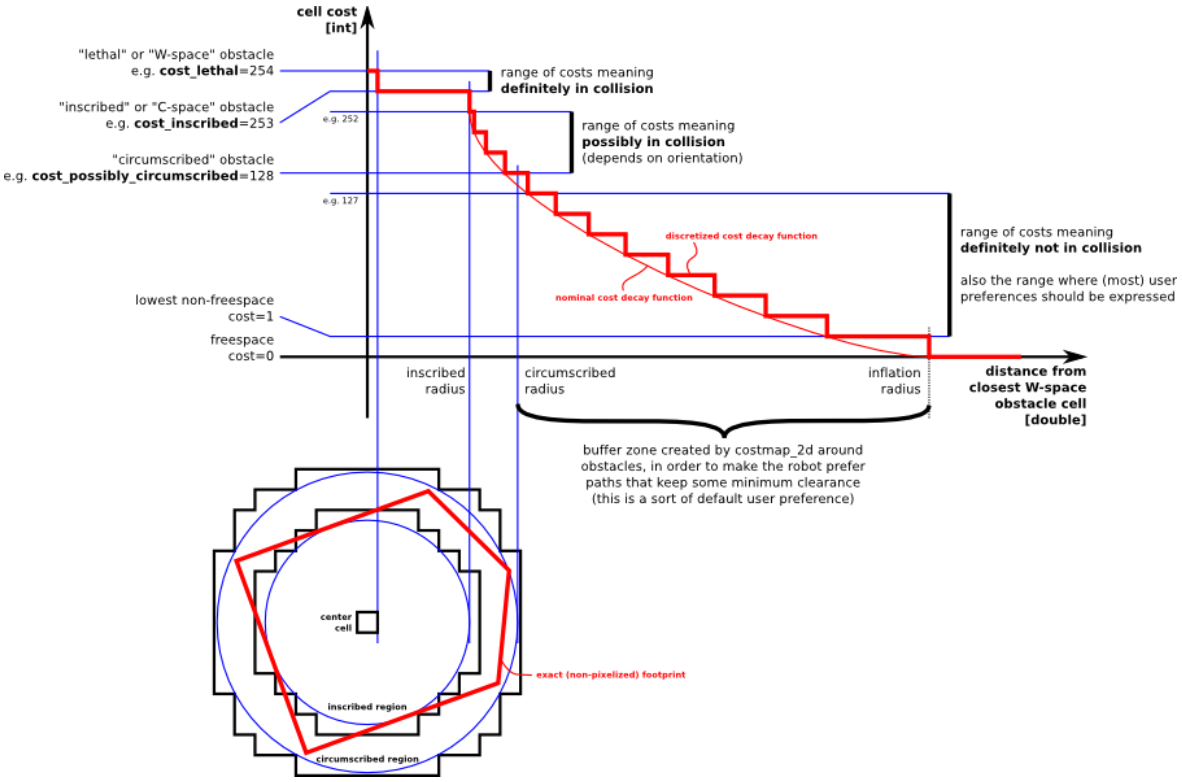

Figura 49 – Relação entre o custo de célula com a footprint de um robô

5.6. Implementação em ROS e parâmetros de configuração

Para colocar a componente de navegação do robô desenvolvido em funcionamento, é necessário correr o seguinte comando: `roslaunch turtlebot_navigation laser_amcl_demo.launch map_file:=/caminho/para/o/mapa.yaml`. Este comando coloca a correr servidor de mapas, o módulo de localização `amcl`, o nó `move_base` e os controladores reativos de segurança. Os parâmetros passados ao nó `move_base` (relativos aos mapas de custo e ao planeador local) são configurados através dos ficheiros na pasta `param/` do módulo `turtlebot_navigation`.

Dos parâmetros de configuração do nó `move_base`, realça-se os parâmetros de velocidade linear mínima (0,1 m/s) e máxima (0,3 m/s), a velocidade angular mínima e máxima (1rad/s em cada sentido), aceleração máxima linear (0,5m/s^2) e angular (0,9 rad/s^2) e frequência do controlador (5Hz).

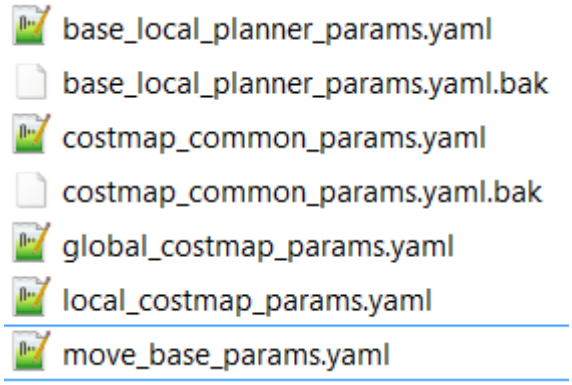

Figura 50 – Ficheiros de configuração do pacote `turtlebot_navigation`

6. Interface gráfica e controlador de alto nível

Um robô guia, por muito sofisticado que seja, precisa de interagir com o seu utilizador de forma a servir o seu propósito. Existe um sem número de formas de um humano interagir com uma máquina, e no âmbito deste projeto onde se espera que o robô acompanhe um visitante da receção de um edifício para um destino dentro do mesmo, faz todo o sentido utilizar uma interface tátil, desenhada para ser de fácil interação e intuitiva. Neste sentido foi desenvolvida uma interface que permite a interação do utilizador com o robô, utilizando um controlador que corre paralelamente à mesma e faz a ponte entre o sistema ROS e a interface gráfica.

O controlador de alto nível do robô e a interface gráfica foram desenvolvidos em Python e correm no mesmo processo em *threads* separadas. Foi escolhida esta metodologia para facilitar a passagem de informação entre a interface gráfica, uma vez que o sincronismo entre as duas *threads* e consistência da informação é garantida pelo *global interpreter lock* (GIL) [38] do Python e outros mecanismos disponibilizados pela biblioteca gráfica utilizada, e o controlador. Na Figura 51 é detalhada a implementação do controlador e da interface gráfica e como as duas se relacionam e trocam dados. De notar que as tarefas de comunicação dentro da *thread* do controlador funcionam de forma assíncrona e podem interromper o fluxo de controlo do robô em momentos não críticos.

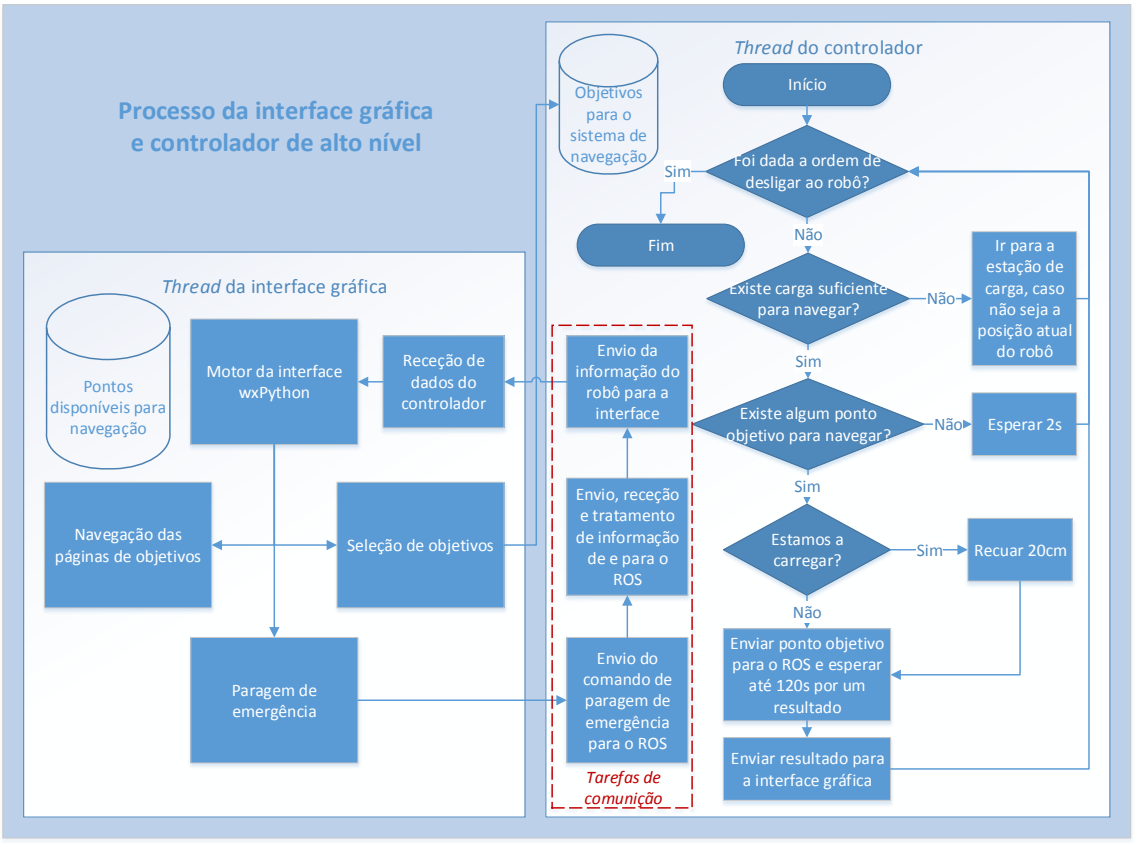

Figura 51 – Implementação do controlador e da interface gráfica e troca de informação entre si

A interface gráfica utiliza o ambiente gráfico wxPython, baseado em wxWidgets. Este ambiente gráfico permite que uma aplicação possa ser utilizada em diversas plataformas (Windows, Linux e Mac) utilizando o mesmo código [39]. A interface foi desenhada para correr em ecrã inteiro e ter dois ecrãs: um dos ecrãs apresenta os objetivos de navegação ao utilizador através de três botões selecionáveis a partir de um clique e duas setas para navegar a lista de objetivos; o outro apresenta o objetivo de navegação escolhido e um botão para efetuar a paragem de emergência do robô. O aspeto destes dois ecrãs foi já mostrado nas figuras das páginas 47 e 48. Internamente, o clique num botão com objetivo de navegação invoca uma função que espoleta o envio da pose objetivo associada, para o controlador do robô. Da mesma maneira, o clique no botão de paragem emergência invoca também uma função que espoleta o envio dessa informação para o controlador do robô. O código de cada uma destas funções pode ser visto na Figura 52.

```
def goToLocation(self, event):
    """Recebe o evento de click no objectivo, actualiza a interface,
    e envia o goal para o controller"""

    botao = event.GetEventObject()

    self.curr_objective_name = botao.GetName()
    self.curr_objective_description = botao.GetLabelText()


    self.label_1.SetLabel("Destino escolhido:\n\n" + self.curr_objective_description)
    self.statusbar.SetStatusText(u"A navegar - Objectivo: %s" % self.curr_objective_description, 0)
    self.showObjectiveFrame()

    if __name__ != "__main__":
        self._controller.goals.append(self._poses[self.curr_objective_name])
    else: #debug
        print "Goal:", self.curr_objective_name, "-", self.curr_objective_description
        print self._poses[self.curr_objective_name]
        print

    event.Skip()

def cancelNavigation(self, event):
    """Envia o pedido de paragem de emergencia para o robo"""
    self.statusbar.SetStatusText(u"Standby - Cancelada a navegação (objectivo anterior: %s)" % \
                                 self.curr_objective_description, 0)

    if __name__ != '__main__':
        self._controller.cancel = True
    self.showMainFrame()
```

Figura 52 – Pormenor do código desenvolvido para enviar os comandos do utilizador para o controlador

7. Testes experimentais e resultados

Neste capítulo serão descritos e analisados os testes experimentais efetuados às diversas partes do robô, assim como os seus resultados. Estes testes focaram-se essencialmente nas capacidades de navegação do robô, nomeadamente de mapeamento e planeamento de trajetórias, e na análise do comportamento da base robótica em vários contextos.

Estes testes foram realizados no edifício que alberga os antigos departamentos da Escola Superior de Tecnologia de Tomar (blocos G a L) do *campus* de Tomar do IPT, no piso térreo e no primeiro piso. Ambos os pisos apresentam um ambiente estruturado com uma quantidade não muito elevada de obstáculos estáticos à navegação, com pisos de tipos diferentes, corredores longos e áreas amplas, e uma quantidade aceitável de *landmarks* e características específicas da estrutura do edifício. Podemos ver as plantas do edifício nas figuras das páginas seguintes.

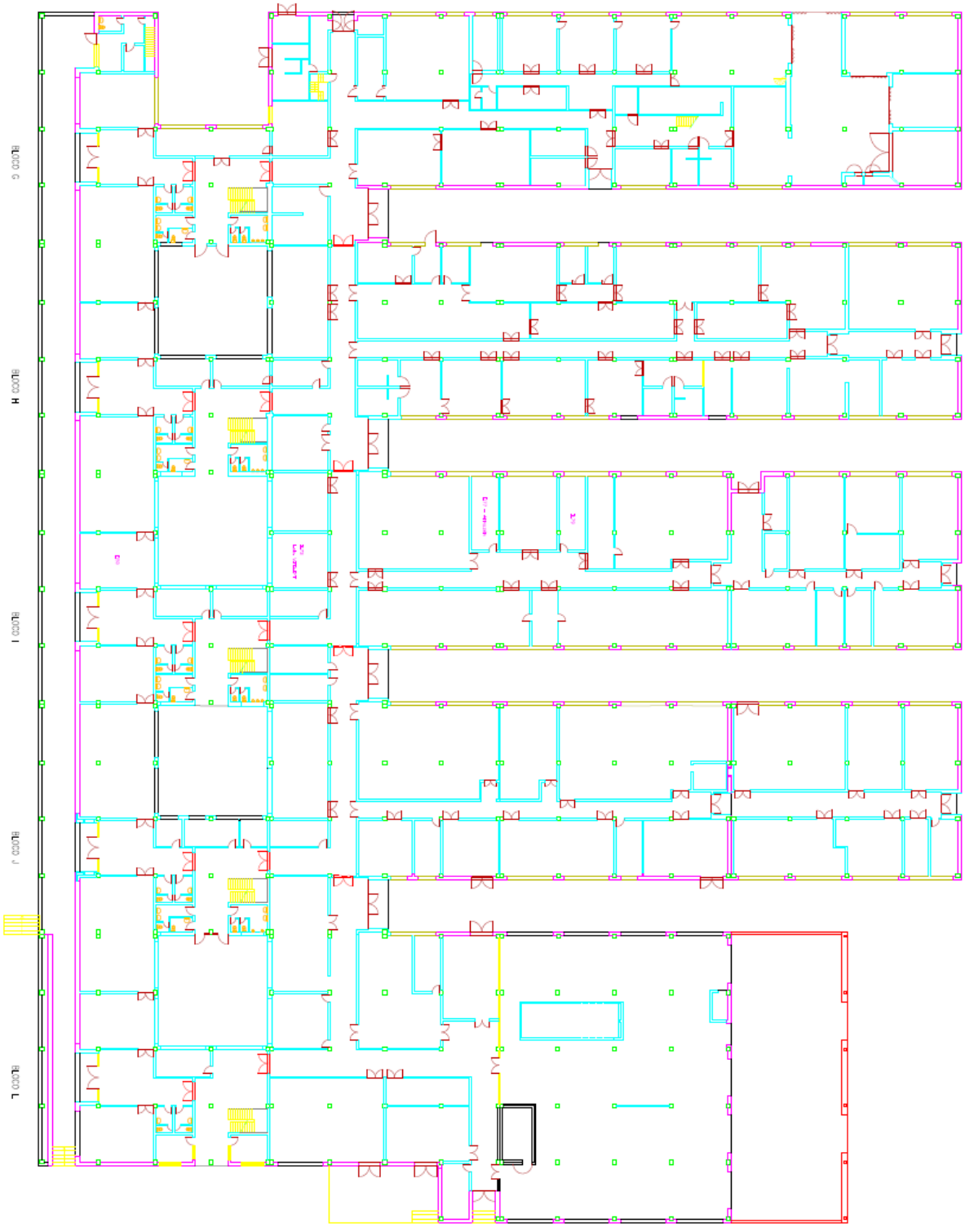

Figura 53 – Planta do piso térreo do edifício

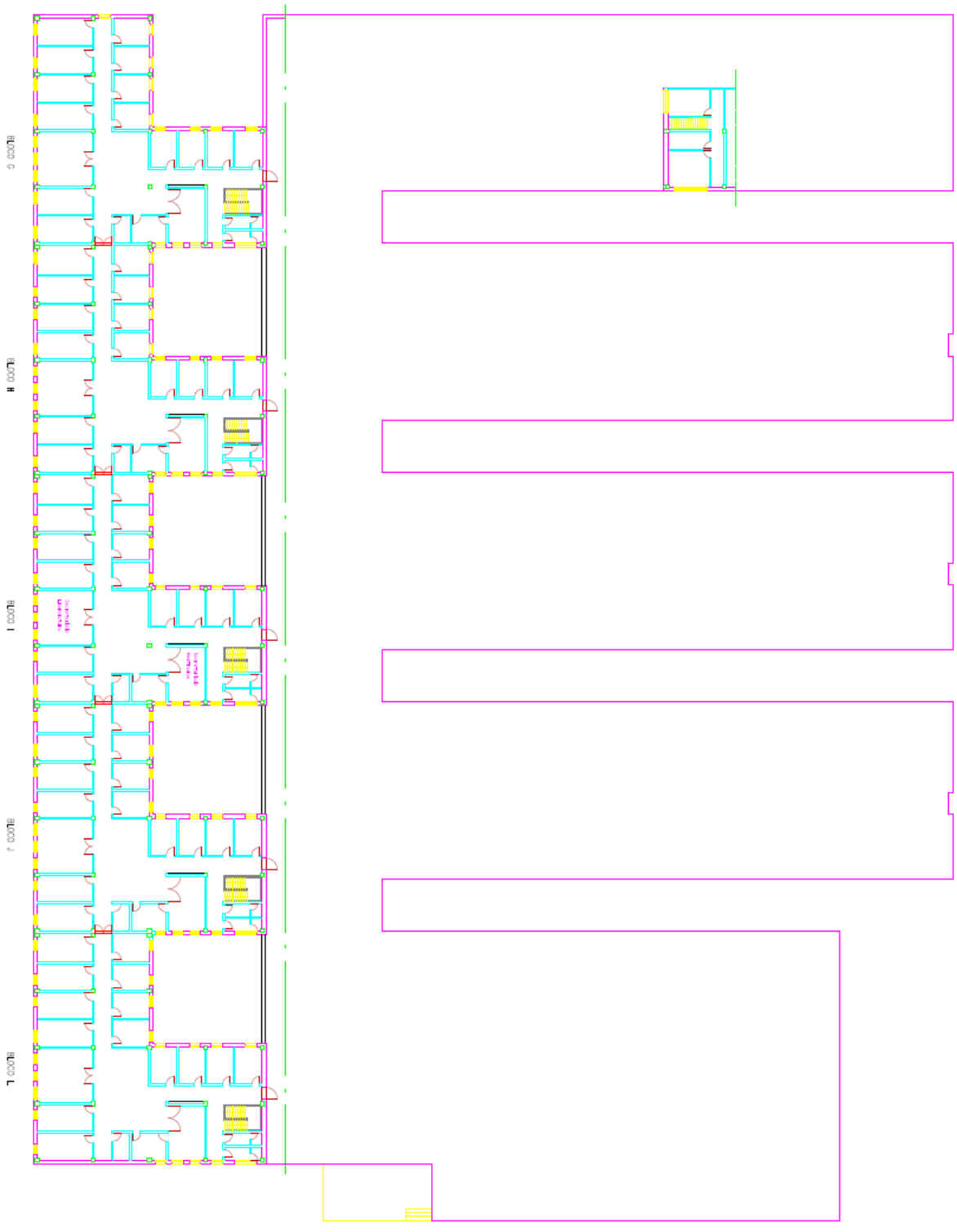

Figura 54 – Planta do primeiro píso do edifício

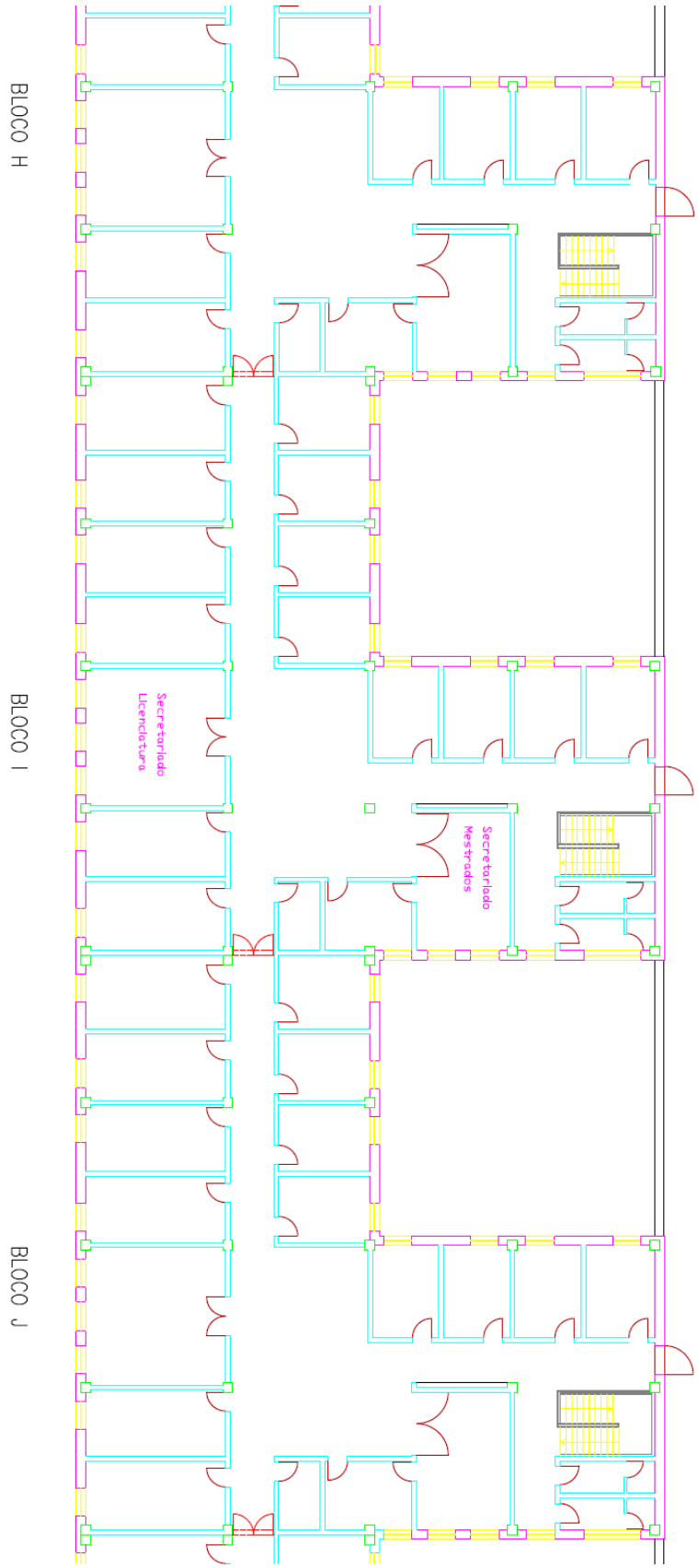

Figura 55 – Pormenor dos átrios dos blocos H a J

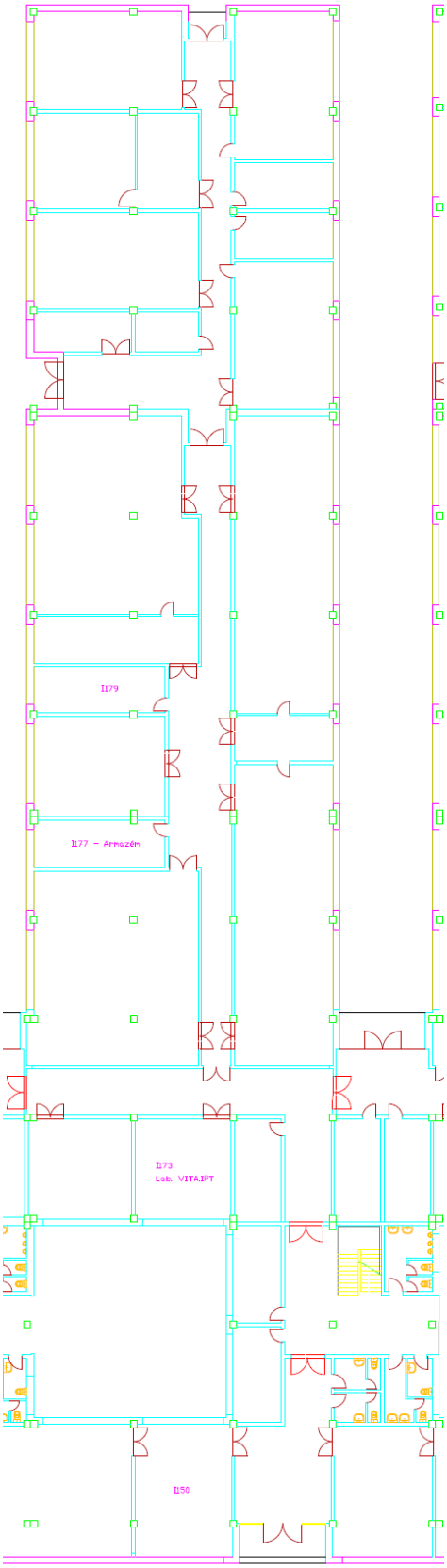

Figura 56 – Pormenor da planta do piso térreo do bloco I

7.1. Testes de SLAM

Como referido anteriormente nesta dissertação, para resolver o problema de SLAM neste robô foi utilizado o Gmapping, que faz a fusão de dados de odometria com leituras de LRF para construir um mapa do ambiente, que é posteriormente utilizado para navegação. É importante termos um mapa o mais fiel ao mundo real quanto possível, pelo que é importante analisar o comportamento do módulo de SLAM, identificar os problemas existentes e averiguar se os resultados obtidos são suficientemente satisfatórios.

Os primeiros testes efetuados foram efetuados no piso térreo do edifício, utilizando a estrutura original do TurtleBot com o LRF montado na plataforma mais elevada, como pode ser visto na Figura 57.

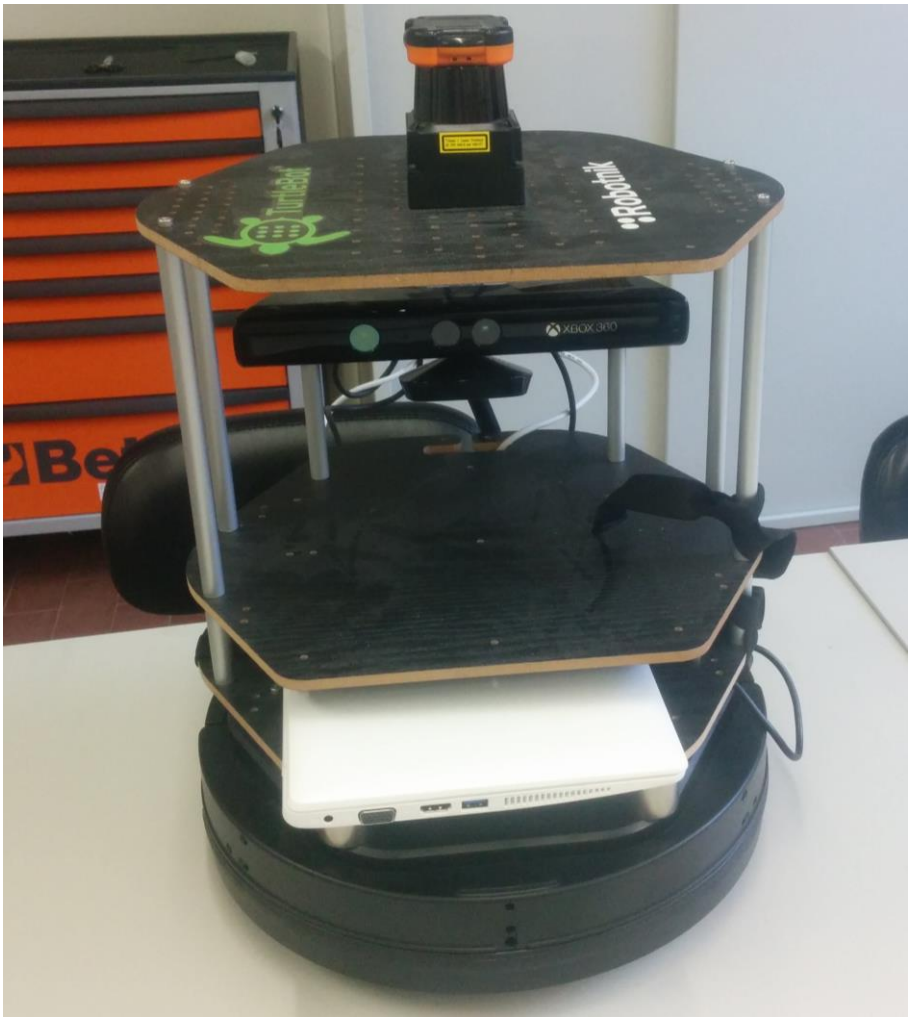

Figura 57 – Robô com a estrutura original do TurtleBot e o LRF montado na plataforma mais elevada

Os primeiros testes passaram pelo mapeamento do piso térreo do edifício I fora do horário letivo, de forma a serem mapeados apenas obstáculos estáticos. O robô navegou o edifício através de teleoperação, tendo começado dentro da sala I173 afeta ao laboratório VITA.IPT, seguiu até à entrada do edifício junto à sala I150, parando na zona das escadas para mapear a mesma. De seguida voltou até à entrada do laboratório, seguindo depois até à entrada do edifício junto ao parque de estacionamento e retornando até ao interior do laboratório. Sempre que possível, evitou-se passar por cima das tampas das caleiras técnicas, e foram fechadas as portas corta-fogo dos edifícios H e J. O mapa resultante pode ser visualizado na Figura 58.

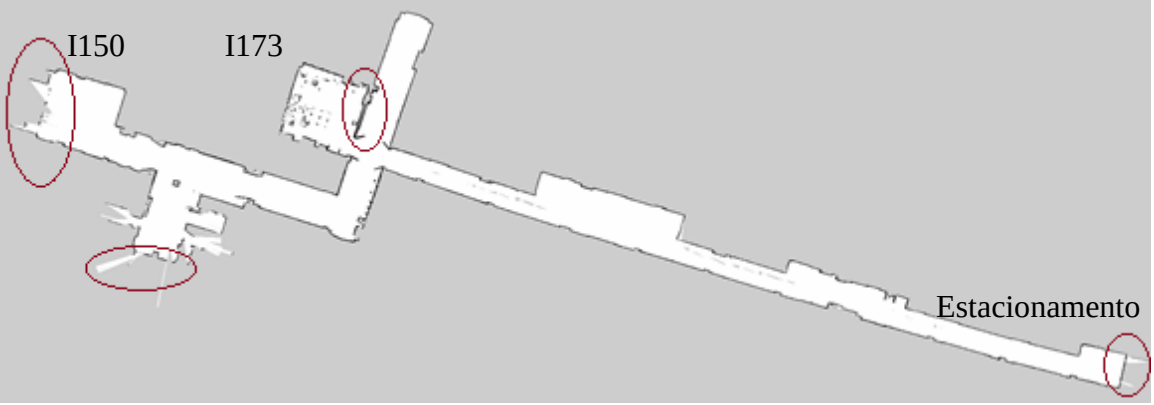

Figura 58 – Mapa resultante do primeiro mapeamento do edifício I com os problemas assinalados

Analisando o mapa podemos ver alguns erros no mesmo, designadamente: nas entradas do edifício e na zona do jardim interior as portas têm vidro à altura do LRF, sendo por isso invisíveis ao robô; apesar dos corredores serem quase perfeitamente direitos e perpendiculares entre si, podemos verificar que o corredor que dá ligação ao estacionamento foi mapeado como se o mesmo fosse em arco; adicionalmente, a parede entre o laboratório e o corredor tem a mesma espessura em todo o seu comprimento, o que não se verifica no mapa. É então fundamental analisar cada um destes erros para averiguar qual a origem dos mesmos, qual a sua influência na navegação do robô e possíveis soluções para os mesmos.

As portas das entradas do edifício são feitas essencialmente de vidro de forma a deixar entrar claridade dentro do mesmo. Isto apresenta um desafio ao mapeamento do edifício uma vez que o vidro é para todos os efeitos invisível para o LRF, uma vez que não reflete o raio laser de volta ao emissor. Uma vez que a mudança ou modificação de portas não é algo que se pretenda fazer nos ambientes a navegar, é necessário arranjar uma estratégia para as mesmas não terem uma influência indesejada na navegação do robô. No caso concreto deste edifício, as portas não são feitas de vidro inteiriço, sendo as mesmas essencialmente feitas de ferro com zonas envidraçadas. Este pormenor faz toda a diferença, uma vez que as zonas de ferro são detetadas pelo LRF, e estão por isso presentes no mapa. Caso isto não acontecesse, seria necessário corrigir manualmente o mapa, que poderia ter impacto na localização do robô durante a navegação, ou usar outros métodos e sensores para mapear as portas, como por exemplo sonares. Para avaliar o impacto da correção manual do mapa, o mesmo foi corrigido e utilizado para navegação do robô, tendo o robô

conseguido localizar-se corretamente dentro das zonas onde foram feitas correções. Para corrigir o mapa foi necessário converter o ficheiro PGM para PNG utilizando a aplicação XnConvert, editado o PNG utilizando a aplicação Paint.NET e foi convertido novamente para PGM utilizando a aplicação XnConvert, tendo em atenção que o PGM é de 16 bits e não de 8 bits. O mapa corrigido pode ser visto na Figura 59.

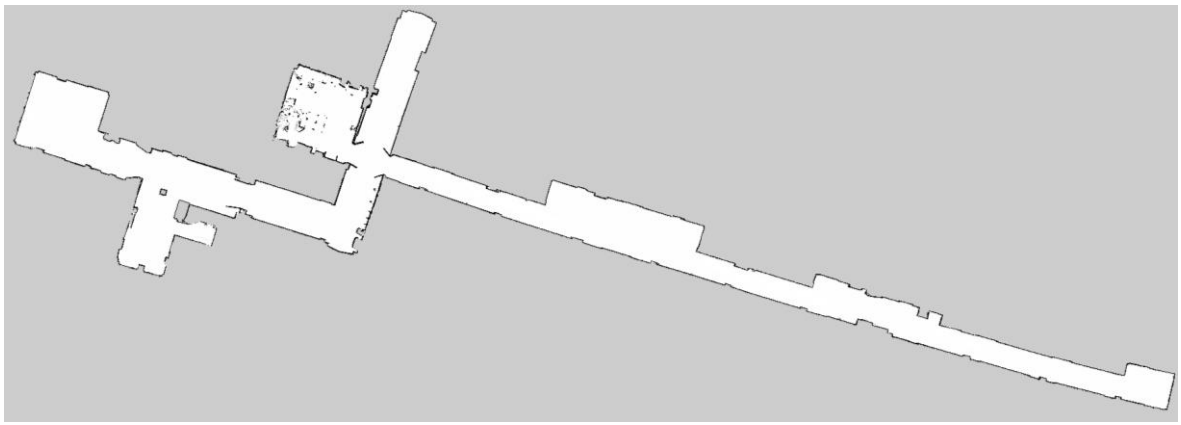

Figura 59 – Mapa do edifício I corrigido

O facto de a perpendicularidade e natureza retilínea dos corredores não se refletir no mapa foi avaliado através de um remapeamento do edifício e por visualização passo a passo do mapa durante a sua criação. Verificou-se que o erro é derivado de erros de odometria, causados tanto pelos codificadores óticos como pela passagem em cima das caleiras, sendo em algumas situações esse erro detetado pelo Gmapping e tomado em consideração na construção do mapa.

Para perceber até que ponto as caleiras estavam a interferir com a precisão da odometria avançou-se para o mapeamento do piso térreo do edifício completo desde o antigo departamento de conservação e restauro até ao de engenharia civil, uma vez que as caleiras não existem na mesma quantidade, disposição e estado de conservação nos vários departamentos. O mapeamento foi efetuado utilizando o método de teleoperação como anteriormente, começando por mapear o bloco I utilizando os percursos descritos acima, seguido dos edifícios H, G, J e L, navegando pelo centro dos corredores, independentemente da posição das caleiras. O mapa resultante pode ser visualizado na Figura 60. Uma vez que estes mapeamentos foram efetuados sempre que possível fora do

horário letivo, não foi possível mapear o corredor do departamento de conservação e restauro até à zona do estacionamento, uma vez que o mesmo se encontrava vedado por questões de segurança.

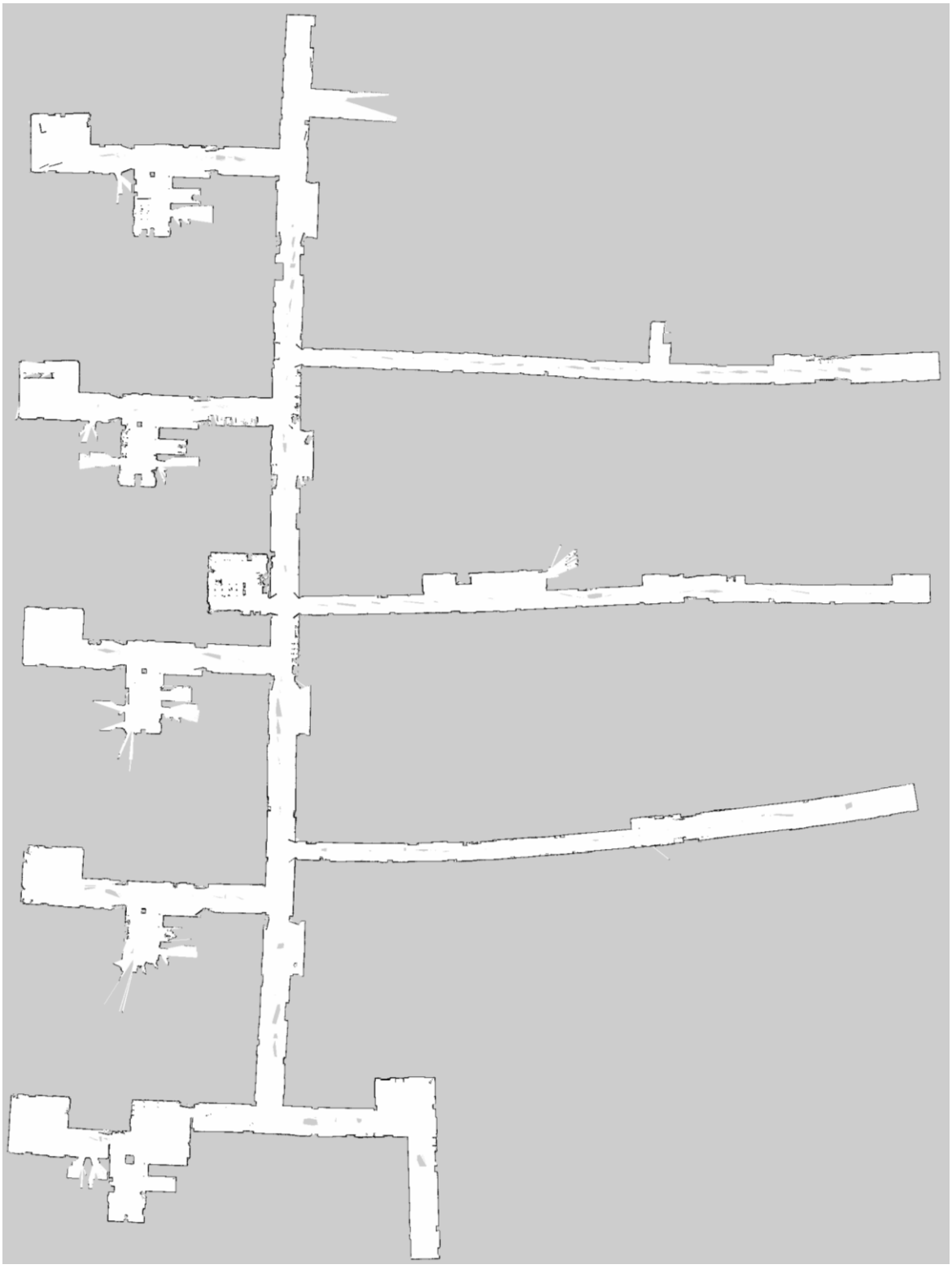

Figura 60 – Mapa do piso térreo do edifício

Como se pode ver, existe um grau de correção bastante diferente no mapeamento dos diversos corredores, em especial nos corredores que levam à zona de estacionamento, sendo estes os que possuem mais caleiras. Nos edifícios H e I, onde as caleiras se encontram em melhor estado de conservação e mais afastadas do centro do corredor, os resultados do mapeamento foram bastante satisfatórios, tendo sido verificado que as zonas onde as paredes se encontram mais deformadas no mapa são onde se encontram as caleiras em pior estado de conservação ou onde se encontram caleiras que atravessam o corredor perpendicularmente. No edifício J, onde as caleiras se encontram em pior estado de conservação, os resultados foram bem menos satisfatórios, tendo sido encontrada uma tampa de caleira a 1/3 do corredor que se encontrava abaulada cerca de 3 cm, tendo quase causado o tombo do robô que se deslocava a uma velocidade aproximada à de um humano a passo. É de notar que este teste foi realizado com a estrutura do TurtleBot, que tem um centro de gravidade mais baixo do que a estrutura final.

Foi repetido o mapeamento do piso inteiro a diversas velocidades, evitando ou não as caleiras, e foi determinada que a odometria se torna mais imprecisa a baixas velocidades, e que a melhor estratégia para mapear o piso térreo passa por evitar tanto quanto possível as caleiras e reduzir a velocidade quando é necessário navegar por cima das mesmas.

Para perceber até que ponto o Gmapping conseguia compensar os erros de odometria, foi feito o mapeamento do primeiro piso do edifício, uma vez que o pavimento não é de mosaico como o mapeado anteriormente e não tem caleiras. Mais uma vez, foi feito o mapeamento do edifício sempre que possível fora do horário letivo, evitando assim obstáculos dinâmicos. O percurso efetuado para o mapeamento do edifício foi semelhante ao utilizado no piso térreo, tendo-se começado o mapeamento na zona da escada e casas de banho do bloco I. O mapa resultante pode ser visualizado na Figura 61.

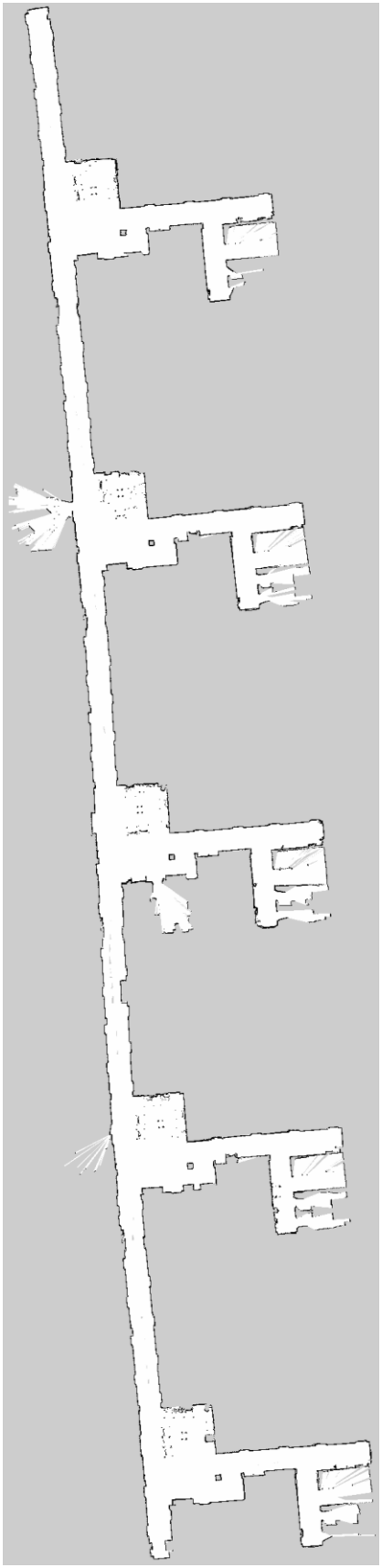

Figura 61 – Mapa do primeiro piso do edifício

Apesar do pavimento do primeiro piso do edifício não ter caleiras como o do piso térreo, este também apresentou desafios, uma vez que a sua menor utilização e o fato de ser mais encerado levou a que as rodas do robô patinassem quando a velocidade angular do mesmo era elevada, nomeadamente quando era pedida a rotação do robô sobre si mesmo. No entanto, como o Gmapping utiliza também a informação disponibilizada pelo LRF, o mesmo conseguiu ignorar os erros de odometria causados pelo patinar das rodas. Ainda assim, durante o mapeamento foi procurado evitar velocidades angulares elevadas para minimizar os erros de odometria que daí advém.

Nas zonas entre as casas de banho e os átrios de cada edifício podemos ver que existem poucos erros de mapeamento, uma vez que os erros de odometria existentes são compensados pela existência de diversas *landmarks*, ou características específicas da estrutura, e pela elevada precisão do LRF utilizado. Estes fatores permitem ao Gmapping ter uma excelente performance aquando da aplicação do método de *scan matching*, que consiste na avaliação do grau de coincidência do varrimento do LRF obtido num dado instante e o varrimento virtual obtido para uma determinada pose do robô, considerando o mapa atual do robô. É possível tirar esta conclusão uma vez que no corredor transversal, onde existem poucas características específicas (o corredor é muito direito), existe uma clara deformação do mesmo, fruto dos erros de odometria. No entanto, através da abordagem utilizada, onde a primeira passagem em cada metade do corredor ocorre em direções diferentes, foi possível minimizar o sem impacto no mapa final, havendo um desvio da parede inferior a 30 cm ao longo do corredor com quase 70 metros. Nos testes de navegação realizados, este erro não teve qualquer impacto na capacidade do robô se localizar no ambiente, pelo que, neste ambiente, a implementação de mecanismos que o resolvam são de importância reduzida, e a sua correção deverá ser possível através da utilização das funcionalidades disponibilizadas no pacote `turtlebot_calibration`.

É importante salientar que se existisse outro corredor paralelo ao transversal junto da área do estacionamento, era espetável que o *loop closure* do mapa melhorasse a qualidade do mesmo, em especial nos corredores deformados do mapa. A eventual aplicação de FAB-MAPs [40], baseado na informação proveniente de um sistema de visão como a Kinect, poderia ainda favorecer o *loop closure* e melhorar esta solução.

7.2. Testes de navegação

Sendo um dos objetivos primordiais deste robô a navegação autónoma, é importante analisar não só as suas capacidades de mapeamento do ambiente, utilizando o Gmapping, mas é também fundamental analisar o desempenho do módulo de localização do robô no mapa do ambiente já existente e do planeamento de trajetórias efetuado pelo robô. A maior parte dos testes realizados ao módulo de localização foram realizados no seguimento dos testes de SLAM, procurando ter um ambiente tão parecido com o mapeado quanto possível de forma a conseguir analisar o comportamento do módulo de localização e planeamento global.

Os primeiros testes realizados ao robô consistiram na navegação desde o interior do laboratório VITA.IPT até ao gabinete do eng. Pedro Neves (sala I179) e retorno especificando os pontos objetivo manualmente no mapa, usando o RViz. Este teste serviu para analisar o comportamento do planeador global, sendo espetável que a trajetória fosse sempre pelo centro do corredor e que no retorno o planeador indicasse que o caminho ótimo a seguir depois de uma rotação de 180º do robô fosse o já utilizado. Foi ainda testado o comportamento do planeador local, onde se pretendia analisar se os parâmetros do mesmo davam preferência a rotações do robô sobre si mesmo *versus* navegar em círculo e seguir o caminho mais próximo à trajetória dada pelo planeador global *versus* seguir um caminho com curvas suaves e convergir com a trajetória dada pelo planeador global mais à frente. O planeador global teve o comportamento esperado, resultando num caminho pelo centro do corredor até ao gabinete, e indicando exatamente o mesmo caminho para o retorno ao laboratório. Do planeamento local efetuado, conseguiu-se chegar à conclusão que os parâmetros do DWA favorecem uma trajetória com curvas suaves em vez de seguir o planeamento global mais fielmente, e como tal, no momento em que o robô tem de retornar ao laboratório, o mesmo efetua uma rotação até se encontrar a 90º da trajetória dada pelo planeador global, movimentando-se depois em arco até se encontrar com a trajetória global.

De seguida foi analisada a capacidade do robô se desviar de obstáculos não mapeados. Para tal, foi colocada uma palete junto ao armazém dos laboratórios de engenharia eletrotécnica (LEE), que iria obrigar o robô a corrigir a sua trajetória, primeiro

em cerca de 2 cm e depois em 10 cm, de forma a evitar a colisão quando o mesmo se encontrava no retorno até ao laboratório. O robô desviou-se como desejado do obstáculo, tendo o desvio feito pelo planeador local sido bastante suave, alterando a sua trajetória, relativamente à obtida sem obstáculo, a cerca de um metro e meio do obstáculo. Para confirmar que o desvio não tinha sido uma inconsistência do planeamento local, foi colocada uma mesa deitada de forma a obrigar o robô a desviar-se mais de um metro da trajetória original, o que aconteceu, demonstrando então que o planeador local estava efetivamente a efetuar o desvio de obstáculos dinâmicos.

Os testes seguintes passaram pela navegação entre três locais, utilizando uma versão simplificada da interface gráfica desenvolvida: o laboratório, o armazém e a entrada do bloco I junto à sala I150 variando a abertura da porta corta-fogo junto às escadas e a do corredor junto ao laboratório. Utilizou-se sempre o mesmo mapa com as portas totalmente abertas, de forma a analisar o comportamento do módulo de localização perante uma diferença de elevada magnitude entre o ambiente observado e o mapa criado *a priori*. Deste modo é possível analisar o comportamento do robô perante aberturas estreitas em que o robô passasse muito junto à porta ou em que a passagem fosse mesmo impossível sem existir colisão. Foi verificado que o robô se comportavacomo esperado, detetando quando a abertura era demasiado pequena para a passagem do robô, e passando quando a folga de cada lado era de pelo menos 2 cm. A diferença da posição das portas em relação ao mapa fornecido não conseguiu perturbar a localização do robô, que se continuava a localizar corretamente.

Finalmente, foram realizados os mesmos testes no primeiro piso do edifício entre a zona das escadas do edifício I, H e J e secretariados de licenciaturas e mestrados, tendo sido testada a deteção de obstáculos e a capacidade de decisão relativamente à transposição ou não dos mesmos recorrendo a *dossiers* colocados no corredor transversal, utilizando para estes testes a interface gráfica desenvolvida. Tal como nos testes efetuados no piso térreo, o robô comportou-se como esperado na maior parte dos testes, havendo no entanto duas situações a apontar: quando o robô detetou que se encontrava preso, uma vez que o obstáculo não o deixava chegar o ponto objetivo, o mesmo tentou voltar para trás e descobrir um caminho alternativo por zonas não mapeadas; adicionalmente, quando tentou voltar para trás, navegou através duma área com mesas e cadeiras sem colidir com nenhuma. Para corrigir a primeira situação, foi desativado o comportamento de

recuperação por omissão do robô em que o mesmo tenta encontrar uma trajetória alternativa sem pedir *feedback* ao utilizador, e optou-se por simplesmente falhar a tarefa de navegação e informar o utilizador do sucedido. Relativamente à segunda, para o ambiente em questão, bastaria aumentar o raio de insuflação dos obstáculos de forma a impedir a passagem do robô nessa área aquando da utilização do planeador global, mas a solução correta passaria pela criação de um mecanismo que introduzisse no mapa de custos zonas de navegação proibida, utilizando por exemplo um mapa semântico simples. Na Figura 62 podemos ver o átrio do bloco I onde ocorreu o sucedido, os mapas de custo global (desvanecido) e local (em cores vivas) junto do robô, sobrepostos no mapa do piso, e a posição do robô a preto.

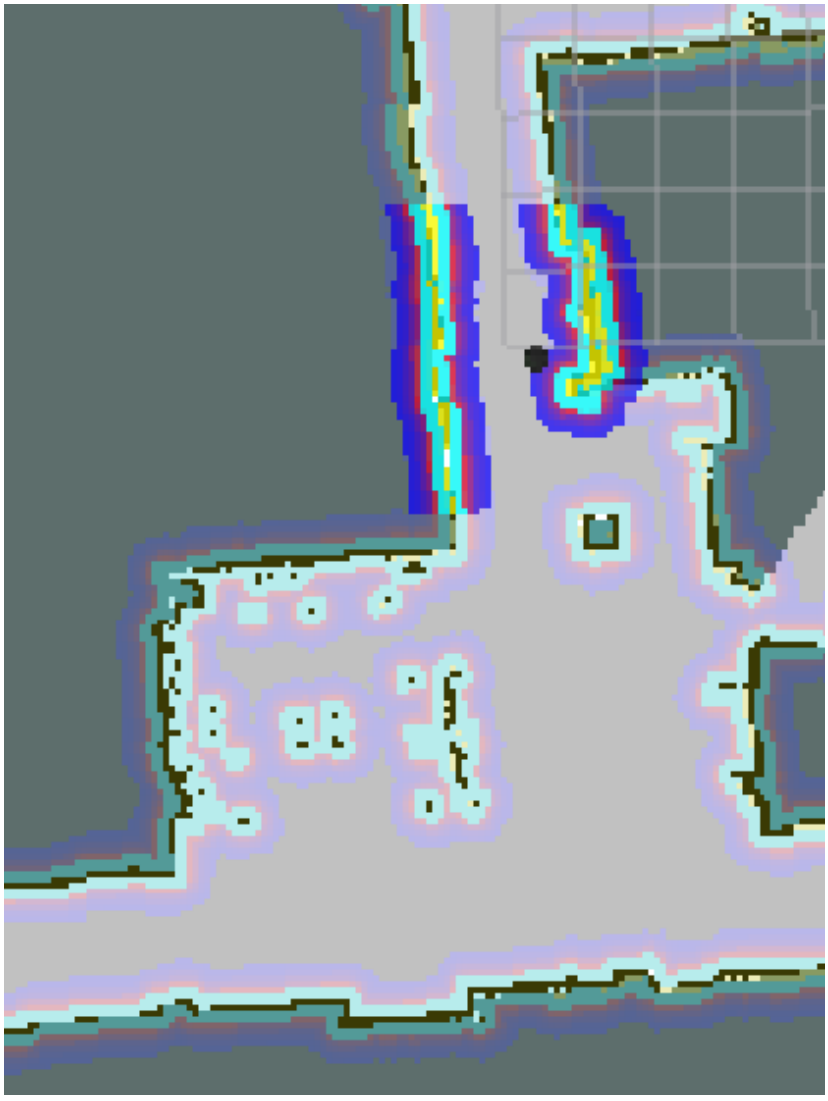

Figura 62 – Pormenor do mapa na zona de cadeiras no átrio do edifício I e mapa de custos com insuflação

7.3. Testes adicionais

No âmbito da edição de 2015 da competição de robótica "GreenT" do projeto "Escolher Robótica, Escolher Ciência" [41] realizada no auditório Doutor José Bayolo Pacheco de Amorim foi feita uma pequena apresentação sobre robótica móvel e o funcionamento do mapeamento de edifícios utilizando um LRF, tendo sido aproveitada a situação para testar o comportamento do LRF num espaço amplo. Para esta demonstração foi montada uma versão simplificada do robô apenas com o LRF e a base motorizada visível na Figura 63, tendo sido criado o mapa da Figura 64 navegando apenas na zona do palco.

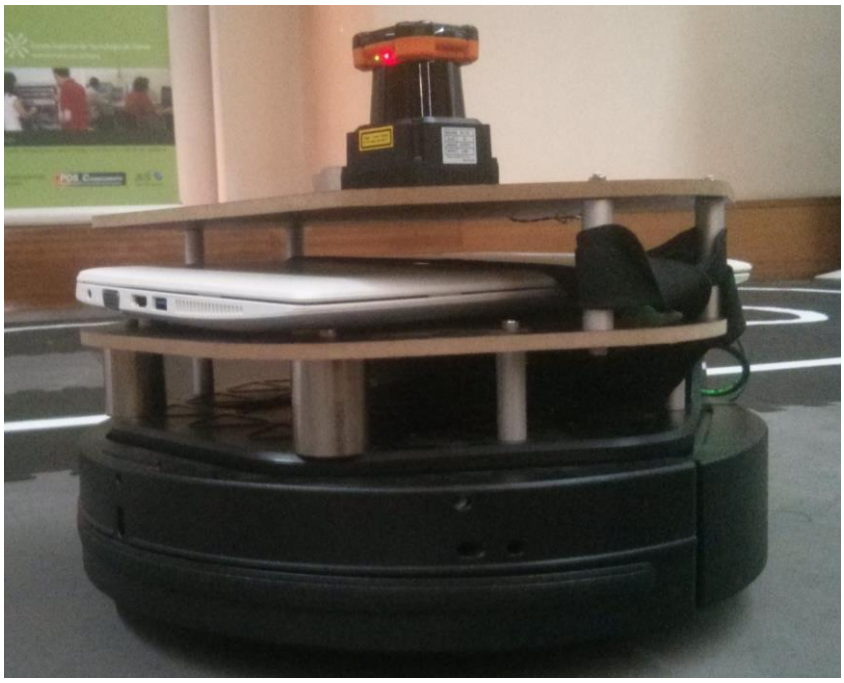

Figura 63 – Configuração do robô durante a demonstração na competição de robótica GreenT

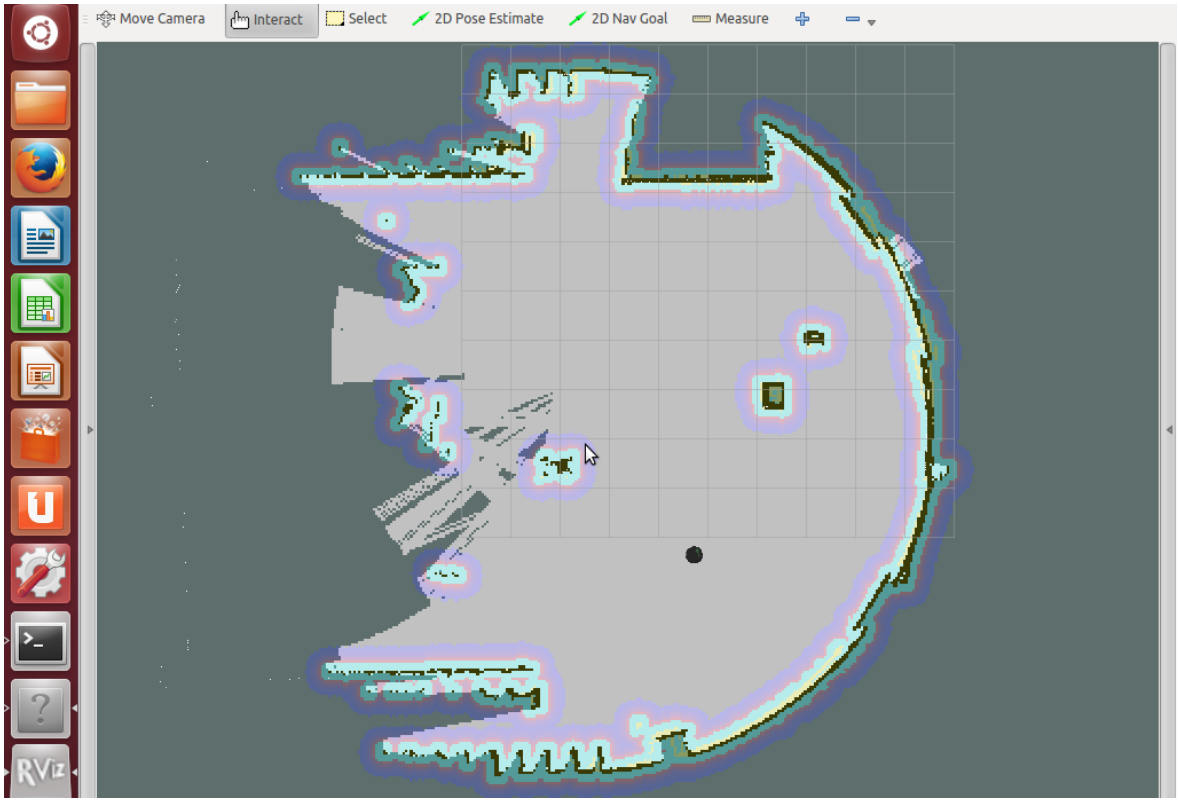

Figura 64 – Visualização do mapa criado durante a competição de robótica "GreenT" no RViz

8. Conclusões e trabalho futuro

Dos objetivos definidos no início deste projeto, nenhum ficou por alcançar: foi desenvolvido um robô guia capaz de navegar autonomamente dentro do edifício da Escola Superior de Tecnologia de Tomar, com uma interface tátil para interação com o mesmo, desenhado de forma modular e devidamente documentado de forma a permitir a continuação deste trabalho.

Um dos meus desejos é que o trabalho desenvolvido não caia no esquecimento, e que seja continuado por outros num futuro próximo, e que essa(s) pessoa(s) tenham a motivação e a possibilidade de fazer, entre outros, os seguintes melhoramentos e desenvolvimentos:

- O desenho de uma interface gráfica mais apelativa e com um maior grau de interação com o utilizador;
- O desenvolvimento de uma ferramenta que facilite a implementação do robô num novo ambiente, permitindo, entre outras funcionalidades, o mapeamento de um edifício com o clique de um botão e a possibilidade de marcar pontos objetivos diretamente no mapa;
- O estudo da utilização do pacote de SLAM da Team Hector, tendo em conta que a abordagem deste pacote dispensa dados de odometria;
- A implementação de uma base motorizada mais robusta, capaz de carregar um peso superior, e mais estável, e o embelezamento do robô;
- A implementação de um sistema de *auto-docking* no controlador do robô, uma vez que a *docking station* do TurtleBot e o seu modo de funcionamento é pouco fiável;
- A substituição do *netbook* por um NUC da Intel ou outro PC de pequenas dimensões.

9. Referências Bibliográficas

[1] D. Ferreira, A. Lagarto e L. Caetano, "Casa inteligente: sistema Ambient Assisted Living," Instituto Politécnico de Tomar, Tomar, 2013.

[2] Carnegie Mellon University, "Minerva - Carnegie Mellon's Robotic Tourguide Project," Carnegie Mellon University, [Online]. Available: https://www.cs.cmu.edu/~minerva/ring-index.html. [Acedido em 5 Novembro 2015].

[3] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte e D. Schulz, "MINERVA: A Second-Generation Museum Tour-Guide Robot," [Online]. Available: http://robots.stanford.edu/papers/thrun.icra_minerva.pdf. [Acedido em 5 Novembro 2015].

[4] The CompanionAble Consortium, "CompanionAble - Project Overview," The CompanionAble Consortium, [Online]. Available: http://www.companionable.net/index.php?option=com_content&view=category&layout=blog&id=8&Itemid=2. [Acedido em 5 Novembro 2015].

[5] C. Huijnen, H. v.d. Heuvel e A. Badii, "CompanionAble: a companion robot supporting people with mild memory impairments," em *Tomorrow in sight: from design to delivery - Proceedings of the 4th AAL Forum*, Eindhoven, 2012.

[6] MOnarCH Consortium, "MOnarCH - Multi-Robot Cognitive Systems Operating in Hospitals," MOnarCH Consortium, [Online]. Available: http://monarch-fp7.eu/pt/project/identification/. [Acedido em 5 Novembro 2015].

[7] MOnarCH Consortium, "Deliverable D2.1.1 (update) - MOnarCH Robots Hardware," [Online]. Available: http://users.isr.ist.utl.pt/~jseq/MOnarCH/Deliverables/D2.2.1_update.pdf. [Acedido em 5 Novembro 2015].

[8] A. Elkady e T. Sobh, "Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography," *Journal of Robotics,* vol. 2012, nº Sistemas de

middleware para robôs, 2012.

[9] Open Source Robotics Foundation, Inc., "ROS Introduction," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Acedido em 5 Novembro 2015].

[10] C. Swetenham, "ROS, Atlas and the DARPA Robotics Challenge," Advanced Robotics Laboratory - HKU, 2014. [Online]. Available: https://events.osrfoundation.org/wp-content/uploads/2014/06/cswetenham-roskong-talk.pdf. [Acedido em 5 Novembro 2015].

[11] Open Source Robotics Foundation, Inc., "ROS Concepts," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Acedido em 5 Novembro 2015].

[12] D. Thomas, "REP 127 - Specification of package manifest format," Open Source Robotics Foundation, Inc., [Online]. Available: http://www.ros.org/reps/rep-0127.html. [Acedido em 5 Novembro 2015].

[13] The Player Project, "Player Project," The Player Project, [Online]. Available: http://playerstage.sourceforge.net/. [Acedido em 5 Novembro 2015].

[14] OpenSLAM, "What is SLAM?," OpenSLAM, [Online]. Available: https://openslam.org/slam.html. [Acedido em 5 Novembro 2015].

[15] OpenSLAM, "Gmapping," OpenSLAM, [Online]. Available: https://www.openslam.org/gmapping.html. [Acedido em 5 Novembro 2015].

[16] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, O. von Stryk e U. Klingauf, "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots," TU Darmstadt, Darmstadt, Alemanha, 2013.

[17] Open Source Robotics Foundation, Inc., "ROS - amcl," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/amcl. [Acedido em 5 Novembro 2015].

[18] A. Patel, "Introduction to A*," Amit Patel, [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.

[Acedido em 5 Novembro 2015].

[19] Open Source Robotics Foundation, Inc., "global_planner - ROS," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/global_planner?distro=hydro. [Acedido em 5 Novembro 2015].

[20] Open Source Robotics Foundation, Inc., "base_local_planner - ROS," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/base_local_planner. [Acedido em 5 Novembro 2015].

[21] Open Source Robotics Foundation, Inc., "costmap_2d - ROS," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/costmap_2d. [Acedido em 5 Novembro 2015].

[22] Open Source Robotics Foundation, Inc., "TurtleBot 2: Open-source robot development kit for apps on wheels," [Online]. Available: http://www.turtlebot.com/. [Acedido em 5 Novembro 2015].

[23] Yujin Robot, "iClebo Kobuki," Yujin Robot, [Online]. Available: http://kobuki.yujinrobot.com/home-en. [Acedido em 5 Novembro 2015].

[24] Microsoft, "Kinect - Windows app development," Microsoft, [Online]. Available: https://dev.windows.com/en-us/kinect. [Acedido em 5 Novembro 2015].

[25] M. Montemerlo, S. Thurun, D. Koller e B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," em *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 2002.

[26] J. Nieto, T. Bailey e E. Nebot, "Recursive scan-matching SLAM," *Robotics and Autonomous Systems,* vol. 55, pp. 39-49, 2007.

[27] W. B. D. F. Sebastian Thrun, Probabilistic Robotics, MIT Press, 2005.

[28] A. Pronobis, "Semantic Mapping with Mobile Robots," KTH Computer Science and Communication, Stockholm, 2011.

[29] W. Burgard, G. Grisetti e C. Stachniss, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," 2007.

[30] D. Gonçalves, "RobChair 2.0: Simultaneous Localization and Mapping and

Hardware/Software Frameworks," Universidade de Coimbra, Coimbra, 2013.

[31] M. M. G. W. Will Maddern, "CAT-SLAM: Probabilistic Localisation and Mapping using a Continuous Appearance-based Trajectory," School of Electrical Engineering and Computer Science, Queensland University of Technology, Queensland, 2011.

[32] F. Dellaert, D. Fox, W. Burgard e S. Thrun, "Monte Carlo Localization for Mobile Robots," 1999.

[33] P. Lima e M. I. Ribeiro, "Kinematic models of mobile robots," IST, Lisboa, 2002.

[34] Open Source Robotics Foundation, Inc., "gmapping - ROS Wiki," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/gmapping?distro=hydro#Parameters. [Acedido em 9 Novembro 2015].

[35] A. Stentz, "The focussed D* algorithm for real-time replanning," em *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[36] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[37] Open Source Robotics Foundation, Inc., "move_base - ROS," Open Source Robotics Foundation, Inc., [Online]. Available: http://wiki.ros.org/move_base?distro=hydro. [Acedido em 10 Novembro 2015].

[38] D. Beazley, "Understanding the Python GIL," em *PyCon 2010*, Atlanta, 2010.

[39] wxPython, "What is wxPython?," wxPython, [Online]. Available: http://www.wxpython.org/what.php. [Acedido em 11 Novembro 2015].

[40] M. Cummins e P. Newman, "Probabilistic appearance based navigation and loop closing," em *2007 IEEE International Conference on Robotics and Automation*, Roma, 2007.

[41] Instituto Politécnico de Tomar, "Projeto Escolher Robótica, Escolher Ciência," Instituto Politécnico de Tomar, [Online]. Available: http://www.robotics.ipt.pt/. [Acedido em 12 Novembro 2015].

10. Anexos

10.1. Anexo 1 – Enunciado do Projeto

Robô rececionista

Orientadores:

Ana Lopes, anacris@ipt.pt

Gabriel Pires, gppires@ipt.pt

Resumo

O projeto tem por objetivo desenvolver um "robô rececionista" que irá operar em empresas/instituições cujas instalações têm dimensão considerável. O robô terá como função receber a pessoa à entrada das instalações, a qual introduzirá o destino pretendido. O robô irá então encaminhar a pessoa até ao destino, através de rotas previamente estabelecidas. No caso de encontrar obstáculos, o robô espera que o obstáculo desapareça até retomar a sua rota. A navegação será baseada em sistemas Laser range finder e sistemas de visão. Poderão ser acrescentadas funcionalidades tais como sínteses de voz e interação com o edifício (e.g. chamada automática de elevador).

Objectivos e Plano de Trabalhos

1) Levantamento de requisitos de hardware e software;
2) Adaptação de plataforma móvel já existente;
3) Ambientação com framework de desenvolvimento ROS (Robot Operating System) e com sistemas Laser;
4) Desenvolvimento de software de navegação usando as bibliotecas do ROS;
5) Implementação das diversas funcionalidades e experimentação;
6) Testes experimentais em cenários reais;
7) Escrita da tese;

Bibliografia

- http://www.ros.org/

10.2. Anexo 2 – Mini-guia de utilização do robô

Este projeto foi desenvolvido utilizando a versão Hydro do ROS no Ubuntu, e o modo de instalar e colocar o robô em funcionamento é em tudo semelhante ao TurtleBot. Como tal, é mais simples colocar o robô a funcionar em modo TurtleBot e só depois efetuar as modificações para servir de robô guia. Caso se faça a monitorização num PC diferente do que é utilizado pelo robô, o processo de instalação é igual, e corre-se os nós do robô no PC do robô, e o RViz ou outros nós no PC de monitorização. Abaixo encontra-se os passos necessários para colocar o robô em funcionamento:

- Instalar o Ubuntu (é necessário verificar nos repositórios quais as versões aconselhadas para a distribuição Hydro)

- Instalação do ROS: Seguir http://wiki.ros.org/hydro/Installation/Ubuntu

- Instalação dos pacotes TurtleBot: Seguir http://wiki.ros.org/turtlebot/Tutorials/hydro/Installation

- Sincronização do relógio (especialmente quando se utiliza mais do que um PC): Seguir http://wiki.ros.org/turtlebot/Tutorials/hydro/Post-Installation%20Setup

- Verificar se a base robótica aparece como `/dev/kobuki`. Caso isso não aconteça, seguir http://wiki.ros.org/turtlebot/Tutorials/hydro/Kobuki%20Base

- Verificar se os pacotes relativos ao LRF se encontram instalados (`hokuyo_node`). Caso não se encontrem instalados, executar `sudo apt-get install ros-hydro-hokuyo-node` para instalar.
  Depois seguir
  http://wiki.ros.org/turtlebot/Tutorials/hydro/Adding%20a%20Hokuyo%20laser%20to%20your%20Turtlebot
  Aconselha-se a leitura do seguinte artigo:
  http://www.iroboapp.org/index.php?title=Adding_Hokuyo_Laser_Range_Finder_to_Turtlebot

- Neste ponto já é possível testar o funcionamento do robô. Seguir http://wiki.ros.org/turtlebot_bringup/Tutorials/hydro/TurtleBot%20Bringup . Utilizar sempre o modo *minimal*. Efetuar teleoperação seguindo http://wiki.ros.org/turtlebot_teleop/Tutorials/Keyboard%20Teleop

- Verificar se a bateria do netbook é reportada no módulo `turtlebot_dashboard`. Caso não aconteça, seguir
  http://wiki.ros.org/turtlebot/Tutorials/hydro/Netbook%20Battery%20Setup
  Caso ainda assim não funcionar, é necessário atualizar o código existente no pacote linux_hardware, estando o código atualizado no GitHub:

https://github.com/turtlebot/turtlebot/tree/hydro/linux_hardware

- Para efetuar mapeamento, seguir http://wiki.ros.org/turtlebot_navigation/Tutorials/Build%20a%20map%20with%20SLAM

- Para experimentar navegar utilizando o mapa criado acima, seguir http://wiki.ros.org/turtlebot_navigation/Tutorials/Autonomously%20navigate%20in%20a%20known%20map

- Para efetuar a calibração da odometria, seguir http://wiki.ros.org/turtlebot_calibration/Tutorials/Calibrate%20Odometry%20and%20Gyro
  De notar que o giroscópio do robô não necessita de calibração, apenas a odometria.

- Para utilizar as configurações do robô guia, é necessário modificar os pacotes do turtlebot, nomeadamente o turtlebot_navigation, utilizando os ficheiros presentes no CD deste projeto.

- Para perceber como funciona o ROS e os comandos mais importantes, aconselha-se a leitura de http://www.cse.sc.edu/~jokane/agitr/ e os livros mencionados em http://wiki.ros.org/Books

- Para executar a interface gráfica e o controlador de alto nível, deverá ser executado o ficheiro `demo.py`. A interface gráfica pode no entanto ser executada sem o controlador de alto nível e sem ligação ao ROS executando o ficheiro `gui.py`. Os pontos objetivos e suas propriedades são definidos no ficheiro `locations.json`.

- Para ter uma versão diferente dos pacotes do TurtleBot sem modificar as originais, na consola entrar na página onde se encontram os pacotes (por exemplo, `cd ~/hydro`), seguido de `catkin_make` e `rospack profile` (para fazer *update* da *cache* do ROS para o `roscd`). Utilizar `rospack list` para ver os caminhos dos pacotes desejados antes e depois do procedimento descrito para confirmar que está tudo correto.

- Para definir a posição inicial do robô pode-se utilizar a aplicação RViz ou então publicando diretamente no tópico `/initialpose`. Exemplo: `rostopic pub --once /initialpose geometry_msgs/PoseStamped { header: {stamp: now, frame_id: "map"}, pose: { position: {x: 7.53958797455, y: -3.07890152931, z: 0.0}, orientation: {w: 1.0}}}`

- Para navegar utilizando os pacotes desenvolvidos, utilizar, por exemplo:
  `roslaunch turtlebot_navigation laser_amcl_demo.launch map_file:=/home/turtlebot/Dropbox/IPT_robo_rececionista/piso1_11_10_2015.yaml`