



Instituto Politécnico de Tomar

**Escola(s) Superior de Tecnologia de Tomar**

**Assistente Pessoal para Séniores:  
Desenvolvimento de uma plataforma  
robótica móvel**

Relatório de Projeto Final

**Carlos Miguel Rodrigues da Silva | 19424**  
**Daniel Henrique Martins Cabaceira | 19510**

Licenciatura em Engenharia Informática

**Outubro 2018**





**Instituto Politécnico de Tomar**

**Escola(s) Superior de Tecnologia de Tomar**

**Assistente Pessoal para Séniores:  
Desenvolvimento de uma plataforma robótica móvel**

**Carlos Miguel Rodrigues da Silva | 19424**

**Daniel Henrique Martins Cabaceira | 19510**

Orientado por:

Ana Cristina Lopes - Instituto Politécnico de Tomar

Pedro Neves - Instituto Politécnico de Tomar

Relatório apresentado ao Instituto Politécnico de Tomar  
para cumprimento dos requisitos necessários  
à obtenção do grau de Licenciado em  
Engenharia Informática



## Resumo

O avanço da tecnologia tem vindo a alargar e a modificar a maneira como vemos atualmente o mundo, destacando-se o impacto cada vez mais significativo da robótica e da inteligência artificial no quotidiano das pessoas.

Este projeto consiste, numa primeira fase, em integrar a plataforma GreenT, desenvolvida no IPT [54], com o sistema ROS e, numa segunda fase, desenvolver uma aplicação robótica que permita ao robô encontrar e distinguir uma pessoa entre um conjunto de objetos e outras pessoas, devendo ainda o robô ter a capacidade de seguir a pessoa identificada. Tendo por base estes objetivos, pretende-se desenvolver um Assistente Pessoal para Sêniores (APS) capaz de assistir e monitorizar pessoas sêniores nas suas tarefas diárias. Este trabalho enquadra-se no projeto de investigação financiado IC&DT *Vitasénior-MT: Senior Healthcare Assistance in Médio Tejo* (Assistência aos Cuidados de saúde de idosos no Médio Tejo), a decorrer no laboratório VITA.IPT. O projeto Vitasénior-MT tem como principal objetivo a monitorização de parâmetros de saúde e ambientais de utentes sêniores. O sistema tem como base a plataforma Raspberry Pi onde foi efetuado todo o desenvolvimento, no entanto o sistema também poderá ser executado num computador com o sistema operativo Ubuntu Linux.

O desenvolvimento deste projeto consistiu, numa primeira fase, na análise da plataforma existente, desde os sensores ao código para Arduino Mega 2560. Desta forma foi delineado um plano sobre que *framework* utilizar, bem como que linguagem de programação usar (desde que suportada pela *framework*). Optou-se por utilizar o *Robot Operating System* (ROS), o qual consiste numa camada de *middleware* utilizada por uma grande comunidade a nível mundial. Após todas as decisões, iniciou-se um estudo intensivo sobre como utilizar o ROS e adaptá-lo ao nosso caso concreto, através da definição da arquitetura de software do sistema.

Posteriormente foram criadas várias versões da plataforma, passando por controlar inicialmente os motores em malha aberta, sendo a odometria realizada no Arduino, posteriormente passou-se a controlar os motores em malha fechada com odometria feita no Raspberry Pi, até à versão mais atual que integra a funcionalidade de deteção de humanos

com base num sensor Microsoft™ Kinect 360 e realizar os cálculos de movimento com base na posição relativa da pessoa ao robô.

O projeto encontra-se disponível em:

- [https://github.com/carlossilva2/Projeto\\_Assistente\\_Pessoal\\_para\\_Seniores](https://github.com/carlossilva2/Projeto_Assistente_Pessoal_para_Seniores)

**Palavras-chave:** Automação, Visão por Computador, Processamento de Imagem, Robôs moveis, Python, ROS.

## Abstract

Technological advancement has been broadening and changing the way we currently see the world, standing out the increasingly significant impact of robotics and artificial intelligence on people's daily lives.

This project, initially, consisted of integrating the GreenT platform, developed in the IPT [54], with the ROS system, and in a second phase, developing a robotic application in which allows the robot to find and distinguish a person between a set of objects and other people, using a Microsoft™ Kinect 360. Besides that, the robot must be able to track and follow the identified person. Based on these goals, it is intended to develop a Senior Personal Assistant (SPA) capable of aiding and monitor elder people in their daily activities. This work is part of a research project financed by IC&DT Vitasenior-MT: Senior Healthcare Assistance in Medio Tejo, which takes place in the VITA.IPT lab. The Vitasenior-MT project main focus is the monitoring of health and environment parameters of senior users. All the development had as its main element of installation the Raspberry Pi, nevertheless it can run on a Linux based computer system.

The mobile platform proposed in this work had as its starting point the GreenT robot, developed in the IPT [54]. Firstly, we conducted an analysis to the existing platform, from the sensors to the code in an Arduino Mega. This way we created a plan on which framework would be used, as well as the coding language to use (ensuring that it would be supported by the framework). It was decided to use the Robot Operating System (ROS), enabling the creation of a middleware layer that is used by a worldwide community. After all the decisions were made, an intensive study on how to use ROS was carried out in order to adapt it to our needs, by defining the software architecture of the system.

Subsequently, many versions of the platform were created, starting with open-loop speed control of the motors, with odometry being calculated on the Arduino, evolving to controlling the motors in a closed-loop speed control, and finishing with the most recent version that integrates human recognition using a Kinect and the ability to do the movement calculus with the person relative position to the robot.

The project is available at:

- [https://github.com/carlossilva2/Projeto\\_Assistente\\_Pessoal\\_para\\_Seniores](https://github.com/carlossilva2/Projeto_Assistente_Pessoal_para_Seniores)

**Keywords:** Automation, Computer Vision, Image Processing, Mobile Robots, Python, ROS.



## **Agradecimentos**

Por trás deste projeto existe um grande trabalho em equipa, cujas sugestões, críticas e elogios foram de extrema importância. Equipa esta, formada por dois amigos que se conheceram no decorrer do curso de Licenciatura em Engenharia Informática. Gostaríamos de fazer um agradecimento especial às seguintes pessoas:

Aos nossos orientadores, Ana Cristina Lopes e Pedro Neves, que sempre nos apoiaram e nos deram grande incentivo durante a realização deste trabalho.

Ao nosso professor Luís Almeida, que nos ajudou a ultrapassar um dos maiores obstáculos deste projeto.

Este trabalho foi suportado financeiramente pelo projeto IC&DT VITASENIOR-MT CENTRO-01-0145-FEDER-023659 com fundos do FEDER através dos programas operacionais CENTRO2020 e FCT.

Agradecemos ao laboratório VITA.IPT e ao Instituto Politécnico de Tomar pelas condições proporcionadas para a concretização deste trabalho.

# Índice

|  |     |
|--|-----|
| <b>Resumo</b> .....  | V   |
| <b>Abstract</b> .....  | VII |
| <b>Agradecimentos</b> .....  | IX  |
| <b>Glossário</b> .....   | XII |
| <b>Índice de Figuras</b> .....   | XIV |
| <b>Índice de Tabelas</b> .....   | XV  |
| <b>Capítulo 1: Introdução</b> .....  | 1   |
| 1.1 Contexto e Motivação .....   | 1   |
| 1.2 Objetivos .....  | 2   |
| 1.3 Contribuições .....  | 2   |
| 1.4 Organização do Projeto .....   | 4   |
| <b>Capítulo 2: Estado da Arte em Robôs Móveis Assistentes Pessoais</b> ..... | 5   |
| 2.1 Robôs Móveis Assistentes Pessoais .....                                  | 5   |
| 2.1.1 O que é um Robô Móvel Assistente Pessoal? .....                        | 5   |
| 2.1.2 Aplicações .....   | 5   |
| 2.1.3 Robôs Móveis Assistentes Pessoais para Sêniores .....                  | 6   |
| 2.2 Integração do ROS em Robôs Móveis diferenciais .....                     | 7   |
| 2.3 Técnicas de Seguimento de Pessoas baseadas em Visão por Computador ..... | 9   |
| <b>Capítulo 3: Camada de Middleware ROS</b> .....                            | 10  |
| 3.1 Arquitetura do ROS - Componentes do Core .....                           | 11  |
| 3.1.1 Nós de ROS .....   | 11  |
| 3.1.2 Tópicos ROS .....  | 12  |
| 3.1.3 Mensagens ROS .....  | 13  |
| 3.2 Fatores de diferenciação .....   | 14  |
| 3.2.1 Infraestrutura de comunicação .....                                    | 14  |
| 3.2.2 Especificidades associadas aos robôs .....                             | 15  |
| 3.3 Ferramentas do ROS .....   | 16  |
| 3.4 Integração com outras ferramentas e bibliotecas .....                    | 18  |
| 3.5 Preparação do workspace ROS .....  | 19  |
| <b>Capítulo 4: Arquitetura de Hardware</b> .....                             | 21  |
| 4.1 Primeira Versão do GreenT (2015) .....                                   | 21  |
| 4.1.1 Arduino Mega 2560 .....  | 22  |

|   |           |
|---|-----------|
| 4.2 Nova Versão do GreenT – GreenTROS (2018).....                   | 23        |
| 4.2.1 Raspberry Pi.....   | 25        |
| 4.2.2 Kinect 360.....   | 26        |
| <b>Capítulo 5: Arquitetura de Software.....</b>                     | <b>28</b> |
| 5.1 Tutoriais de ROS .....  | 29        |
| 5.2 Comunicação série ROS - Arduino (rosserial).....                | 30        |
| 5.3 Implementação do ROS no Raspberry Pi.....                       | 31        |
| 5.4 Controlo em malha aberta .....                                  | 31        |
| 5.5 Controlo através de joystick com controlo em malha fechada..... | 33        |
| 5.6 Implementação do sensor Kinect.....                             | 35        |
| 5.7 Seguimento de linha baseado no sensor Kinect.....               | 37        |
| 5.8 Implementação do OpenNi .....                                   | 39        |
| 5.9 Seguimento de uma pessoa com base num sensor Kinect .....       | 39        |
| 5.10 Características dos componentes ROS .....                      | 41        |
| <b>Capítulo 6: Aplicações .....</b>                                 | <b>44</b> |
| 6.1 Seguimento de Linha .....                                       | 44        |
| 6.2 Seguimento de Pessoas .....                                     | 46        |
| 6.3 Resultados Experimentais .....                                  | 47        |
| <b>Capítulo 7: Conclusões e Trabalho Futuro.....</b>                | <b>50</b> |
| 7.1 Conclusão .....   | 50        |
| 7.2 Trabalho Futuro.....  | 50        |
| <b>Capítulo 8: Bibliografia .....</b>                               | <b>51</b> |
| <b>Capítulo 9: Anexos .....</b>                                     | <b>56</b> |

## Glossário

| Índice | Termo         | Descrição  |
|--------|---------------|--|
| 1      | Software      | Código compilado   |
| 2      | Scripts       | Ficheiro que contém uma ou mais instruções de computador                                 |
| 3      | Linux         | Sistema Operativo construído com base num <i>kernel</i> Linux                            |
| 4      | Framework     | Conjunto de <i>software</i> pré-feito por terceiros                                      |
| 5      | Malha Aberta  | Sistema que não contém realimentação   |
| 6      | Malha Fechada | Sistema com realimentação através de sensores  |
| 7      | Bumpers       | Geralmente botões que em caso de colisão reverterem ou param a marcha                    |
| 8      | Middleware    | Conjunto de <i>software</i> que funciona como intermediário entre duas ou mais entidades |
| 9      | Hardware      | Componentes físicos que constituem uma plataforma  |
| 10     | ROS           | Acrónimo de <i>Robot Operating System</i>  |
| 11     | Open Source   | <i>Software</i> não proprietário e permissível a alterações                              |
| 12     | MD5           | Tipo de cifra assimétrica <i>Hash</i>  |
| 13     | stream        | Sequência de dados digitalmente codificados  |
| 14     | UDP           | <i>User Datagram Protocol</i>  |
| 15     | TCP           | <i>Transmission Control Protocol</i>   |
| 16     | Socket        | É um terminal interno para enviar e receber dados  |
| 17     | Odometria     | Método para estimar a posição de um robô   |
| 18     | Debugging     | Processo de verificação de erros/problemas num pedaço de código                          |
| 19     | Encoder       | Conversor de posições angulares para sinais digitais/analógicos                          |
| 20     | GUI           | <i>Graphical User Interface</i>  |

|    |                 |  |
|----|-----------------|--|
| 21 | CMake           | Gestor do processo de compilação de <i>software</i>                        |
| 22 | PWM             | <i>Pulse Width Modulation</i>  |
| 23 | I/O             | Entradas de <i>Input</i> e <i>Output</i>                                   |
| 24 | ARM             | <i>Advanced RISC Machine</i>   |
| 25 | RISC            | <i>Reduced Instruction Set Computer</i>                                    |
| 26 | SoC             | <i>System on a Chip</i>  |
| 27 | RAM             | <i>Random Access Memory</i>  |
| 28 | GB              | Medida de tamanho de dados, <i>Gigabyte</i>                                |
| 29 | GPIO            | <i>General Pins for Input and Output</i>                                   |
| 30 | Slot            | Tipo de conector   |
| 31 | IoT             | <i>Internet of Things</i>  |
| 32 | Kernel          | Conjunto de operações básicas de um sistema operativo                      |
| 33 | Workspace       | Zona de trabalho reservada ao projeto                                      |
| 34 | API             | <i>Application Programming Interface</i>                                   |
| 35 | mAh             | miliAmpére hora: unidade de capacidade de uma bateria                      |
| 36 | Motor de física | Software de simulação de certos sistemas da Física.                        |
| 37 | Imagem stereo   | Uma imagem com a intenção de criar uma impressão visual em três dimensões. |

# Índice de Figuras

|   |    |
|---|----|
| Figura 1: Logótipo do ROS Kinetic .....   | 11 |
| Figura 2: Componentes do Core.....  | 14 |
| Figura 3: Vários nós de um robô.....  | 16 |
| Figura 4: Ferramenta de visualização RVIZ .....   | 17 |
| Figura 5: Logótipo do ROS Industrial.....   | 19 |
| Figura 6: GreenT (2015) .....   | 21 |
| Figura 7: Diagrama de blocos da arquitetura inicial. ....   | 22 |
| Figura 8: Arduino Mega 2560.....  | 22 |
| Figura 9: APS (2018) .....  | 23 |
| Figura 10: Camada aplicativa da nova arquitetura. ....  | 24 |
| Figura 11: Diagrama de Blocos da nova arquitetura. ....   | 24 |
| Figura 12: Raspberry Pi 3B.....   | 25 |
| Figura 13: Microsoft Kinect 360.....  | 27 |
| Figura 14: Janela de simulação do Turtlesim.....  | 29 |
| Figura 15: Simulação turtlesim .....  | 30 |
| Figura 16: Exemplo de um subscritor num Arduino.....  | 31 |
| Figura 17: Exemplo de um Publisher num Arduino.....   | 31 |
| Figura 18: Primeira versão da arquitetura de Nós ROS do GreenT .....  | 32 |
| Figura 19: Arquitetura de Nós ROS do GreenT com controlo por Joystick .....   | 33 |
| Figura 20: Joystick Xbox .....  | 34 |
| Figura 21: Estrutura de comunicação entre OpenCV e o ROS .....  | 36 |
| Figura 22: Exemplo de código Python para extrair uma imagem e processar os dados .....                                  | 37 |
| Figura 23: Arquitetura de nós ROS utilizando a Kinect .....   | 38 |
| Figura 24: Arquitetura de nós ROS atual do robô.....  | 40 |
| Figura 25: Primeira versão do GreenT.....   | 44 |
| Figura 26: Fluxograma do seguimento de linha através da Kinect. (Anexo 1).....  | 45 |
| Figura 27: Cálculo de velocidade e extração da posição do utilizador .....  | 48 |
| Figura 28: Representação gráfica de uma pessoa (Lado Esquerdo) e do robô Assistente Pessoal Sênior (Lado Direito) ..... | 49 |
| Figura 29: Detecção de uma pessoa usando Openni Tracker .....   | 49 |

## Índice de Tabelas

|  |    |
|--|----|
| Tabela 1: Comparação de Vantagens e Desvantagens de um Raspberry ..... | 25 |
| Tabela 2: ROS openni_tracker .....                                     | 41 |
| Tabela 3: ROS Node greenT_Kinect .....                                 | 41 |
| Tabela 4: ROS Node greenT_teleop .....                                 | 42 |
| Tabela 5: ROS Node greenT_pid_vel .....                                | 42 |
| Tabela 6: ROS Node greenT_tf.....                                      | 43 |
| Tabela 7: ROS Node rosserial Arduino Mega.....                         | 43 |

# Capítulo 1: Introdução

## 1.1 Contexto e Motivação

Atualmente, Portugal enfrenta um envelhecimento da população preocupante. Com a diminuição da taxa de natalidade e o aumento da esperança média de vida, a percentagem de idosos por cada 100 habitantes é cada vez maior. Segundo estatísticas da PORDATA, a idade média da população portuguesa era cerca de 44 anos em 2016. Em 2015, a percentagem de habitantes com idade igual ou superior a 65 anos por cada 100 habitantes era cerca de 20.5% e em 2016 cerca de 20.9%, e espera-se que em 2017 esta percentagem continue a subir [1][2][3][4]. Estes valores demonstram um rápido envelhecimento da população portuguesa, aliado ao facto de a esta tendência demográfica se seguir um maior número de pessoas com limitações físicas e cognitivas.

O projeto Vitasenior-MT, no qual se insere este trabalho, tem como objetivo desenvolver uma solução tecnológica de tele saúde/teleassistência para acompanhar e melhorar os cuidados de saúde de idosos a viver isoladamente na região Médio Tejo. É importante frisar que nesta região, em particular, a proporção entre o envelhecimento e a diminuição da população é superior à média nacional.

A plataforma robótica GreenT foi inicialmente concebida como uma plataforma didática suportada por um Arduino Mega 2560, com capacidade de ler diversos sensores e com o objetivo de realizar desvio e contorno de obstáculos e navegação por meio de seguimento de linha. Contudo, colocou-se o desafio de desenvolver uma segunda versão avançada desta plataforma robótica, de modo a transformá-la num robô de serviços, mais especificamente num robô Assistente Pessoal de Sêniores (APS). Para que tal fosse possível, foi necessário dotar a nova versão da plataforma robótica com certas capacidades, em particular, o APS devia ser capaz de identificar um determinado utilizador e navegar até ou com ele. A arquitetura de hardware da primeira versão GreenT, baseada numa placa de desenvolvimento Arduino Mega 2560, não possuía a capacidade de processamento nem os sensores necessários para realizar a identificação e seguimento de um utilizador.

Para transformar o GreenT num robô de serviços é fundamental dotá-lo de capacidades de navegação e perceção que permitam realizar tarefas de identificação e



seguimento do utilizador. Para tal optou-se por integrar a *framework* “*Robot Operating System*” (ROS), que funciona como camada de *middleware* na plataforma robótica. O ROS é utilizado por uma grande comunidade mundial, encontrando-se bem documentado e permitindo a reutilização de *software* para robótica desenvolvido por toda uma comunidade internacional. Quer a integração do ROS quer a aplicação de identificação e seguimento dos utilizadores requer uma nova arquitetura de hardware, de modo a aumentar a capacidade de processamento e a obter capacidade de perceção 3D. Para tal desenvolveu-se uma nova arquitetura de hardware baseada num Raspberry Pi 3B e num sensor Kinect 360 para computação 3D.

## 1.2 Objetivos

Este projeto tem os seguintes objetivos:

- Adaptação da Arquitetura existente para uma Arquitetura ROS – Substituição e desenvolvimento de novas arquiteturas na plataforma GreenT;
- Seguimento e Reconhecimento de Pessoas – Utilizar algoritmos de visão por computador de modo a obter uma representação 3D de uma pessoa;
- Navegação Autónoma – Dotar o robô com a capacidade de navegar o espaço, seguindo uma pessoa previamente identificada.

## 1.3 Contribuições

Quando nos foi apresentado o trabalho, este era apenas um robô seguidor de linha, com capacidade de se desviar de obstáculos utilizando os sensores infravermelhos e/ou um sonar, bem como dois *bumpers* que em caso de colisão invertiam o sentido do movimento. O nosso contributo passou por criar todo o software de modo a alcançar os objetivos mencionados na secção acima descrita.

## **Principais contribuições**

1. Reconfiguração do Hardware do robô, descrito em detalhe no capítulo 4, para as seguintes características:
  - a. Substituição de todos os sensores por uma Kinect;
  - b. Integração de um Raspberry Pi como principal fonte de processamento;
  - c. Substituição da fonte de alimentação.
2. Desenvolvimento da Arquitetura de Software do sistema. Esta arquitetura encontra-se descrita no capítulo 5 e é constituída por cinco Nós de ROS:
  - a. greenT\_tf – Encarregue de calcular e publicar a odometria;
  - b. greenT\_pid\_vel – Gere o controlador PID;
  - c. greenT\_teleop – Calcula a que velocidade a atingir para chegar ao objetivo;
  - d. greenT\_Kinect – Realiza o algoritmo de seguimento e deteção de pessoas. Envia a uma mensagem do tipo Twist de acordo com a distância.
3. Desenvolvimento de um algoritmo de seguimento de linha para ROS (descrito no capítulo 6):
  - a. Utilização do sensor Kinect para recolha de imagens;
  - b. Processamento das informações recolhidas no Raspberry Pi;
  - c. Processamento das imagens recolhidas;
  - d. Tomadas de decisão baseada no varrimento.
4. Desenvolvimento de um algoritmo de seguimento de pessoas com base na Kinect (descrito no capítulo 6):
  - a. Utilização de uma representação cartesiana 3D das posições de uma pessoa;
  - b. Mapear velocidades através da posição relativa da pessoa;
  - c. Verificar o sentido de orientação;
  - d. Deteção de colisões.

## 1.4 Organização do Projeto

Nesta secção é apresentada uma breve introdução sobre o conteúdo abordado em cada capítulo.

No Capítulo 2, referente ao estado da arte, são abordados aspetos gerais sobre robôs móveis assistentes pessoais, como funciona a integração da *framework* ROS e que técnicas de seguimento de pessoas existem.

O Capítulo 3 aborda a composição da camada de *middleware* do ROS e são descritas as principais funcionalidades do projeto.

No Capítulo 4 é efetuada uma descrição de todo o hardware existente na plataforma e de como sofreu alterações ao longo das várias iterações do projeto.

No Capítulo 5 apresenta-se a arquitetura de software implementada no projeto, mais concretamente a arquitetura de Nós de ROS.

No Capítulo 6 são discutidas as várias aplicações desenvolvidas no âmbito deste projeto.

No Capítulo 7 são efetuadas as considerações finais a respeito deste trabalho, mostrando quais foram os pontos chave deste desenvolvimento, visando a viabilidade de robôs deste nível.

## Capítulo 2: Estado da Arte em Robôs Móveis Assistentes Pessoais

### 2.1 Robôs Móveis Assistentes Pessoais

#### 2.1.1 O que é um Robô Móvel Assistente Pessoal?

Um robô assistente pessoal é um tipo de robô capaz de ser movimentar (por meio de rodas, lagartas, pernas, etc.), em que o principal objetivo é a interação humana e em parte, através de algoritmos, a capacidade de condução autónoma objetiva. Estes robôs tendem a melhorar a qualidade de vida das pessoas, no ambiente em que estão inseridos, proporcionando ao utilizador a sensação de companhia e uma interface com o mundo digital (através de ecrãs táteis).

Numa apresentação da Universidade de Stanford [34], um robô assistente pessoal é descrito de diferentes maneiras, como um robô que realiza uma tarefa do foro físico para o bem-estar de uma pessoa com incapacidade, ou, como um robô capaz de sentir, processar dados através de sensores e realizar ações que beneficiem pessoas com dificuldades ou a população sénior.

#### 2.1.2 Aplicações

Quando se fala em robôs assistentes pessoais, parte-se do princípio que se fala, sobretudo, de robôs de assistência à população sénior ou com deficiência. No entanto, esta é uma premissa falsa. Existem vários tipos de robôs inseridos nesta arquitetura, tais como guias de museus, assistentes em hospitais, assistentes séniores, guias de aeroportos, etc.

Alguns exemplos:

- **Robô *Minerva*:** Um robô que funciona como guia em museus, fornecendo informação sobre o que estão a ver durante o percurso e entretenimento. Este robô está inserido no museu Smithsonian, museu da História Americana nos Estados Unidos [35];

- **Robô *Tub***: Funciona como um assistente na área da medicina, de modo a facilitar o transporte de comida ou medicamentos aos pacientes. Este tem capacidade de condução autónoma e deteção de obstáculos e é usado internacionalmente [36];
- **Projeto *GrowMeUp***: Este é um projeto português, criado na Universidade de Coimbra, com o objetivo de melhorar a qualidade de vida da população sénior, tentando mitigar a sensação de solidão neste conjunto de pessoas [37];
- **Projeto Assistente Pessoal para Séniores**: Projeto inserido no programa Vitasenior-MT, desenvolvido no Instituto Politécnico de Tomar, com o âmbito de, tal como o projeto *GrowMeUp*, melhorar a qualidade de vida da população sénior.

### 2.1.3 Robôs Móveis Assistentes Pessoais para Séniores

Existe uma panóplia de robôs assistentes a à população sénior no mercado, no entanto iremos apenas ilustrar 4 exemplos.

Começando pelo robô *Aibo ERS-III0*, criado pela *Sony* pela primeira vez em 1999, totalmente contruído em plástico e tinha ao seu dispor vários sensores como sensores táteis, infravermelhos, colunas e uma câmara. Para além disso movimentava-se através de 4 (quatro) pernas, no entanto, a cabeça e a cauda eram móveis. Utiliza sistema operativo proprietário, o *Aperios*, e tem um processador da arquitetura RISC de 64 bits, com uma velocidade de relógio de 100 MHz, e 8 MB de DRAM. O objetivo de robô era o entretenimento, contudo, houve estudos que procuram os efeitos deste robô na qualidade de vida e sintomas de stress [38] [39] [40].

*Paro* é um robô terapêutico sob a forma de uma foca e é usado de forma a proporcionar um efeito de relaxamento e calma em hospitais e lares de idosos. Foi desenvolvido no Japão e está equipado com um processador duplo de 32 bits, 3 microfones e 12 sensores táteis cobertos por pelo. Para além disso contém motores silenciosos para se poder movimentar. Segundo a regulamentação dos Estados Unidos, o *Paro* é considerado um dispositivo médico de classe 2 desde 2009 [38] [41] [42].

O *Pearl* é um robô desenvolvido pela Universidade de Pittsburgh e pela Universidade de Canergie Mellon para ajudar a população sénior. O âmbito deste projeto era criar um robô móvel pessoal que auxilia idosos com doenças crónicas nas suas atividades diárias em lares

de idosos. Tem uma interface de fácil que dá conselhos e ajuda cognitiva. Alguns dos componentes que o constituem são um sistema de movimentação diferencial, 2 computadores, sensores sonar, lasers medidores de distâncias e uma câmara [38] [43] [44].

*RoboCare* é um robô de cuidados domésticos para os idosos criado na Universidade de Roma. Foi criado de forma a ajudar a supervisionar a população sénior e pessoas com problemas físicos severos e problemas mentais a tomar decisões no seu dia-a-dia, lembrando de tarefas, perigos adjacentes à atividade que praticam, etc. Para isso utilizou-se uma combinação de software, sensores e robôs para alcançar o objetivo [38] [45].

## **2.2 Integração do ROS em Robôs Móveis diferenciais**

Espalhados pelo mundo existem diversas plataformas robóticas (diferenciais) que utilizam o ROS de forma a possibilitar a criação de uma arquitetura escalável e poderosa. Em seguida serão citadas algumas como exemplo.

### **TurtleBot 2**

- **Características**
  - Base Kobuki
    - Detecção de sobrealimentação nos motores
    - Giroscópio
    - Bumpers – Esquerda, Centro e Direita
    - Sensor de Queda
    - Alimentação: Bateria Li-Ion de 14.8V, 2200 mAh/4400mAh
    - Sensor Infravermelho
    - Sensores de profundidade
    - 8 pins de *Input* e 4 pins de *Output*
  - Câmara Asus Xion Pro Live
  - Um Netbook compatível com ROS
  - *Hardware* para Kinect

## **TurtleBot 3 Waffle Pi**

Versão miniaturizada do TurtleBot 2, com capacidade para algoritmos SLAM.

- **Características**
  - Raspberry Pi 3
    - Explorado em detalhe no Capítulo 4
  - LiDAR de 360°
  - Raspberry Pi Camera
  - OpenCR (32-bit ARM Cortex-M7)
    - Arquitetura ARMv7E-M
    - 1MB de memória Flash
    - 320KB de memória RAM
    - Relógio de 216 MHz ou 462 DMIPS
  - Bateria Li-Po de 11.1V a 1800 mAh
  - 2 Motores DYNAPIXEL

## 2.3 Técnicas de Seguimento de Pessoas baseadas em Visão por Computador

Visão por computador é, da perspectiva da engenharia, um processo de automatização de tarefas que o sistema visual humano consegue fazer. Inclui tarefas como adquirir, processar e analisar imagens digitais de forma a obter uma representação digital da mesma. Para isso é necessário recorrer a algoritmos que tenham capacidade para tal, como é o caso do algoritmo usado neste projeto.

Existem vários algoritmos, alguns públicos outros privados, que permitem a deteção e distinção de pessoas, no entanto, iremos apenas mencionar dois deles, o utilizado neste projeto e o KLT (Kanade-Lucas-Tomasi).

O algoritmo KLT utiliza mecanismos de *machine-learning* de modo a extrair pontos importantes num conjunto de imagens, numa escala de cinzentos, para criar vetores de deslocamento. Este algoritmo utiliza uma representação em pirâmide, de maneira a que as imagens se tornem mais suaves e faz uso de recursividade de modo a alcançar este objetivo. Face a estas características, o algoritmo consegue detetar pequenos deslocamentos nos pixéis inerentes à imagem [49] [50] [51] [52] [53];

O algoritmo utilizado no Assistente Pessoal Sénior (APS) é algo bastante mais rudimentar, derivado à natureza das suas especificações. Este algoritmo tira partido do referencial cartesiano que o ROS cria aquando da instanciação do projeto, fazendo com que a Kinect tenha um ponto de referência constante. No entanto, de maneira a ter possibilidade de seguir pessoas, é extraída as informações cartesianas do “torso” do utilizador e aplicando-se comparadores de posição. Estes comparadores verificam se houve movimento lateral ou horizontal fazendo, ao mesmo tempo da verificação, um mapeamento de velocidades de acordo com as posições nos eixos.



## Capítulo 3: Camada de Middleware ROS

O *Robot Operating System* (ROS) é uma *framework open-source* que providencia ferramentas e bibliotecas para a criação de software no âmbito da robótica.

O ROS foi construído de maneira a formar um ecossistema com todos os seus utilizadores em cooperação, encorajando o desenvolvimento de software de robótica de forma colaborativa. Sendo uma plataforma flexível, é facilitado o uso e integração de ferramentas ou bibliotecas criadas por outras entidades promovendo assim o espírito descrito em cima.

Devido à sua arquitetura modular e distribuída, é possível usar, de entre todos os disponíveis, apenas os componentes necessários e úteis à aplicação e ainda implementar componentes desenvolvidos pelo utilizador. Neste sentido, estima-se que além dos componentes designados de ROS Core existam mais de 3000 pacotes criados por utilizadores, sendo estes apenas os que foram tornados públicos [5].

A comunidade ROS encontra-se em crescimento, no início a sua maioria resumia-se aos laboratórios de investigação em robótica, no entanto tem vindo a expandir-se até áreas como a educação, o setor comercial e a indústria.

O core do ROS está licenciado segundo a BSD (*Berkeley Software Distribution*), uma licença *open-source* contendo poucas restrições e sendo bastante permissiva, permitindo o uso, alterações e comercialização sem qualquer tipo de custos [33].

Muitas das capacidades atuais da robótica avançada para manipulação, perceção e navegação foram desenvolvidas em ROS. Grandes empresas internacionais como a Airbus e a Boeing, numa área tão exigente como a aviação, usam ROS em diversas das suas aplicações. Além do mais, *software* desenvolvido em ROS é cada vez mais comum em aplicações como robôs manipuladores, robôs de serviço, *drones*, carros autónomos, entre outros [6].

### 3.1 Arquitetura do ROS - Componentes do Core

O ROS é apresentado sobre a forma de distribuições. Neste momento existe mais do que uma distribuição suportada, no entanto o trabalho realizado assentou na ROS Kinetic Kame (Figura 1), lançada em maio de 2016 e com suporte até abril de 2021.



Figura 1: Logótipo do ROS Kinetic

Os sistemas operativos suportados pela versão Kinetic do ROS são: Ubuntu (Willy e Xenial) e Debian (Jessie). Além destes sistemas Linux, o ROS já possui compatibilidade experimental com OS X (Homebrew) [16].

Em termos de estrutura do sistema importa identificar algumas partes do core, as suas funcionalidades, especificações e qualidades por forma a obter uma visão geral do ambiente ROS.

#### 3.1.1 Nós de ROS

Um Nó de ROS é um processo que executa um determinado trabalho de computação. Os Nós são combinados e interligados num grafo do sistema e comunicam usando um fluxo de dados baseado em Tópicos de ROS, entre outros métodos de troca de mensagens. Os Nós de ROS normalmente operam com um elevado nível de granularidade, sendo um sistema de ROS constituído por vários. Ou seja, o sistema como um todo constitui-se de vários nós que completam tarefas específicas, por exemplo, um nó controla a locomoção, outro nó controla a navegação, outro nó faz o planeamento e assim sucessivamente.

O uso de nós traz diversos benefícios à arquitetura, nomeadamente: tolerância a falhas, uma vez que os problemas ficam isolados em cada nó; complexidade do código reduzida, quando comparada com arquiteturas monolíticas pois o sistema pode ser estruturado por forma a isolar as diferentes tarefas em nós específicos, sendo mais rápido o

acesso, percepção e modificação do código; detalhes da implementação mais protegidos, devido a exposição mínima de apenas uma API ao resto do grafo.

Todos os nós de um grafo são distinguidos com um nome único que os identifica no sistema e podem ser programados usando bibliotecas para clientes ROS como o `roscpp` e o `rospy`, para C++ e Python respetivamente, tendo sido este último o usado no APS com exceção de algumas ferramentas e bibliotecas importadas em C++ [11].

### 3.1.2 Tópicos ROS

Os Tópicos são barramentos ou fluxos de dados sobre os quais os Nós trocam Mensagens ROS. Nestes existe uma mecânica de separação da informação que é consumida e produzida através de um sistema de publicação/subscrição. Em geral um Nó não sabe com quem comunica e apenas tem interesse em determinados dados que são produzidos por um Tópico subscrito. Por outro lado, Nós que produzem informação relevante fazem publicação para que esta fique disponível para consumo. Um Tópico pode ter múltiplas publicações e subscrições [12].

Cada Tópico depende fortemente do tipo de mensagens que foram usadas para a sua publicação e os Nós só podem receber mensagens desse mesmo tipo. Além do mais todos os clientes ROS usam MD5 para assegurar que os Nós foram compilados consistentemente. Isto quer dizer que em cada mensagem trocada é gerado um código *hash* de 128-bit que é adicionado ao cabeçalho da mesma e na receção é confirmada a integridade dos dados comparando o valor de *hash* após ser executado o algoritmo de *hashing* com o valor de *hash* recebido [7].

O transporte dos Tópicos é baseado nos protocolos TCP/IP e UDP. No caso da vertente TCP/IP é o designado TCPROS, que efetua o *stream* dos dados das mensagens através de ligações persistentes TCP/IP, sendo este o protocolo por defeito do ROS e também o único protocolo de transporte que tem de ser suportado por todas as bibliotecas. O transporte baseado no UDP, UDPROS apenas é suportado no `roscpp` (C++) e separa as mensagens em pacotes UDP, o que apesar de atingir uma baixa latência de comunicação é mais suscetível a perdas de informação, sendo mais usado em teleoperação.

A camada de transporte TCPROS suporta Mensagens ROS (assíncronas) e Serviços ROS (síncronos), sendo que as conexões de entrada são recebidas via TCP Server Socket e filtradas consoante o seu cabeçalho, que contem as informações da mensagem e de encaminhamento. Se o cabeçalho contiver o campo “*topic*” será encaminhado como uma ligação a um ROS *Publisher* [13].

Um *Subscriber* de TCPROS requer o envio dos seguintes campos:

- *message\_definition*: texto completo da definição da Mensagem ROS
- *callerid*: nome do *Subscriber*
- *topic*: nome do Tópico ROS ao qual o *Subscriber* se liga
- *md5sum*: a verificação MD5 do tipo da mensagem
- *type*: tipo da Mensagem ROS

Por outro lado, um *publisher* TCPROS necessita de responder com os campos abaixo para assegurar uma ligação bem-sucedida:

- *md5sum*: a verificação md5 do tipo da mensagem
- *type*: tipo da Mensagem ROS

### 3.1.3 Mensagens ROS

Como referido anteriormente os Nós ROS comunicam entre si através da publicação/subscrição aos diferentes Tópicos ROS. Uma Mensagem ROS é a estrutura de dados que é transmitida através dos barramentos ou fluxos de dados. Os campos da estrutura suportam os tipos de dados tradicionais (*integer*, *floating point*, *boolean*, etc.) bem como o uso de *arrays* [14].

Todo o conceito de Tópicos, publicação/subscrição, Nós e Mensagens de ROS pode ser resumido na Figura 2 [8].

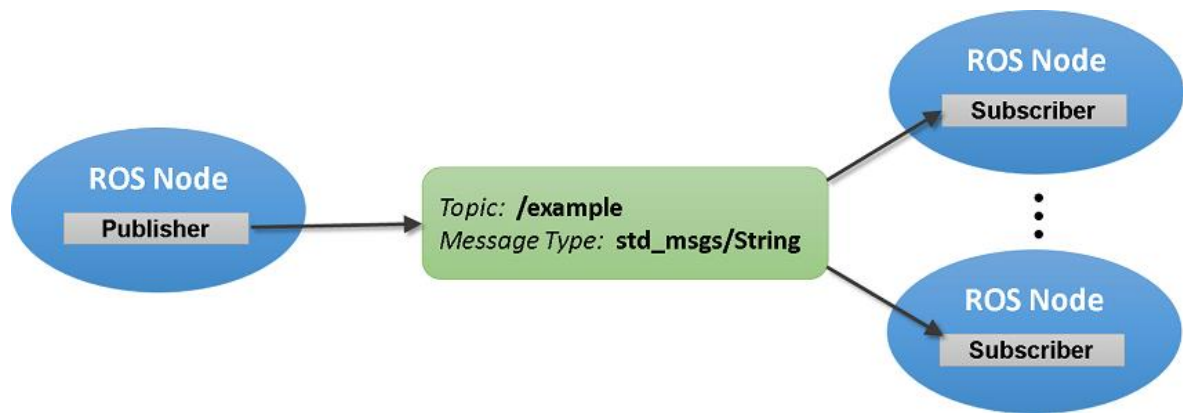


Figura 2: Componentes do Core

### 3.1.3.1 Twist

Um dos tipos de Mensagens ROS mais usados neste projeto são as mensagens Twist. Por forma a mover um robô no ROS é necessário publicar mensagens de Twist, compostas por componentes lineares e angulares das velocidades. A componente linear traduz-se em velocidades lineares para os eixos x, y e z, representada por um vetor no espaço livre que apenas indica a direção. A componente angular refere-se à velocidade angular em torno dos referidos eixos (x, y, z) também representada por um vetor no espaço livre que apenas indica a direção. Em suma, o Twist expressa a velocidade no espaço livre dividida entre as suas partes lineares e angulares [15].

## 3.2 Fatores de diferenciação

### 3.2.1 Infraestrutura de comunicação

No nível mais baixo o ROS oferece uma interface de transmissão de mensagens que providencia comunicação entre os diferentes processos que constituem a camada de *middleware*.

Ao implementar uma aplicação de robótica um dos primeiros paradigmas é a implementação de um sistema de comunicação. O ROS gere a comunicação entre os vários

nós distribuídos através do sistema de publicar e subscrever tópicos. Desta forma, e sendo as mensagens anónimas, os dados são melhor encapsulados e a reutilização de código é promovida.

Como o sistema de publicação/subscrição (Tópicos ROS) é anónimo e assíncrono, os dados são facilmente capturados e usados sem nenhuma alteração ao código

Caso seja necessário o uso de interações síncronas o *middleware* ROS proporciona esta capacidade através do uso de serviços ROS [9].

### **3.2.2 Especificidades associadas aos robôs**

Com vista a complementar a camada de abstração criada pelos componentes Core do ROS, o *middleware* proporciona normas de mensagens para robôs.

A maioria das aplicações de robôs estão bem definidos nos formatos de mensagens suportados pelo ROS, estando disponíveis: conceitos geométricos como a pose, transformadas e vetores; sensores como câmaras, lasers, etc.; dados de navegação avançados, odometria, caminhos e mapeamento.

O uso deste tipo definido de mensagens assegura que os componentes criados para o ROS (Nós), funcionam em sintonia com o restante ecossistema, desde ferramentas a bibliotecas, até ao próprio ROS Core.

Uma dessas bibliotecas tem por função disponibilizar a localização das diferentes partes do robô em relação aos diferentes referenciais. A *tf* (*transform*) é uma biblioteca que permite gerir e coordenar os dados das transformadas dos robôs, quer as suas partes sejam estáticas ou móveis e estejam fixas a um referencial móvel ou imóvel [17]. São assim eliminados problemas causados pelo facto de os diferentes produtores e consumidores de dados estarem distribuídos pelo sistema bem como a informação ser atualizada com intervalos de tempo diferentes. A biblioteca *tf* mantém as relações entre as diferentes coordenadas das partes num buffer, estruturadas em forma de árvore e em relação com o tempo, permitindo ao utilizador transformar e relacionar pontos e vetores entre quaisquer dois referenciais em qualquer momento [9].

Por forma a melhor visualizar todos estes conteúdos e ainda conferindo a capacidade de personalizar e descrever o robô numa linguagem legível para o sistema, o ROS dispõe de uma série de ferramentas para descrever e modelar o robô. A linguagem URDF (Unified

Robot Description Format) consiste num documento em formato XML onde se descrevem as propriedades físicas do robô e até mesmo as suas características físicas.

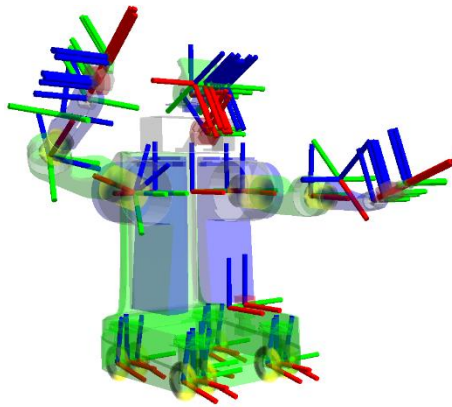


Figura 3: Vários nós de um robô

Uma vez definidas, estas propriedades podem ser usadas pelo restante sistema ROS, usados pela biblioteca *tf* ou feito o seu *render* em 3D e usados em simulações ou planeamento de movimento através da ferramenta *rviz*, abordada mais adiante.

Os casos de uso de comunicação para a área da robótica, na perspectiva do ROS, são quase completamente cobertos pelo sistema de publicação/subscrição e pelos serviços.

### 3.3 Ferramentas do ROS

A par das bibliotecas, as ferramentas do ROS formam um dos seus pontos mais fortes. Estas podem consistir em Nós que dão suporte a uma diversidade de sistemas de apoio como sensores e câmaras, mas também se traduzir em pequenos programas que ajudam o utilizador a fazer uma melhor introspeção, *debugging*, programação e visualização do estado do sistema que se está a desenvolver.

Além do mais todo o sistema ROS pode ser acedido via linha de comandos (Linux). Todas as funcionalidades do Core, lançamento de Nós, teste de Tópicos, leitura e escrita de dados em Mensagens, entre outros, podem ser feitos através dos diversos comandos desta interface.

Não obstante esta abstração de interface gráfica, a mesma pode ser usada através de ferramentas visuais como o referido anteriormente rviz (Figura 4) [10].

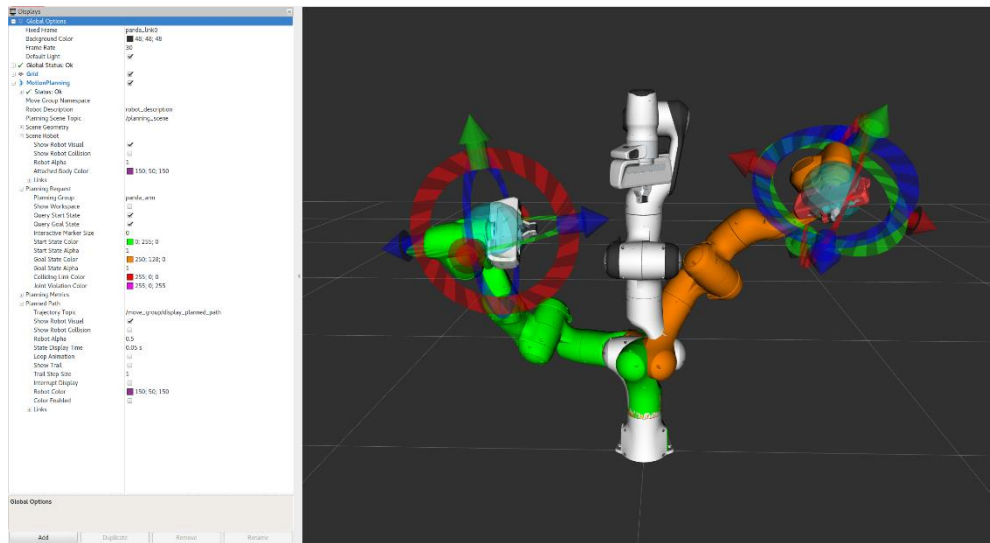


Figura 4: Ferramenta de visualização RVIZ

Aquela que é provavelmente a interface mais conhecida do ROS, o rviz proporciona, a partir da descrição de um robô via URDF, visualização a três dimensões do sistema completo. O rviz permite a definição dos referenciais para os diversos pontos do sistema em sintonia com a biblioteca tf e a transformação gráfica do movimento, pose e informação dos diferentes sensores. Acresce ainda o suporte para os diversos tipos de Mensagens ROS comuns ao ROS como o caso de *laserscans*, *point clouds* (nuvens de ponto) 3D e imagens de câmara.

Em termos de *frameworks* para visualização de dados o *middleware* ROS usa a ferramenta rqt. Apoiadas nos diversos plugins existem três funções principais: introspeção e visualização em tempo real do sistema com uma GUI que mostra os gráficos de computação do ROS (rqt\_graph) [18]; monitorização de *encoders* e outras variáveis que possam ser representadas por um valor a variar no tempo numa GUI 2D (rqt\_plot) [19]; monitorização e uso de Tópicos consultando qualquer número de Tópicos a serem usados no sistema e permitindo a publicação de Mensagens ROS próprias para qualquer Tópico (rqt\_topic, rqt\_publisher) [20] [21].

Com o intuito de criar e aceder a registos o ROS usa o formato Bag. A GUI criada pelo plugin rqt\_bag mostra e executa os ficheiros Bag onde ficam guardadas as Mensagens



ROS de um determinado período no sistema, sendo possível comparar os dados dos diversos Tópicos e o estado geral do ambiente [22] [23].

### 3.4 Integração com outras ferramentas e bibliotecas

Mantendo a sua abstração característica em termos de tecnologia, linguagem e hardware, a integração de sistemas exteriores é inerente ao ROS.

No caso do APS foi aplicada a biblioteca OpenCV (Open Source Computer Vision Library), a principal ferramenta de visão por computador da atualidade, usada tanto em ambiente académico como aplicada em vários produtos já existentes em todo o mundo. O ROS proporciona uma forte integração com o OpenCV, permitindo aos utilizadores o acesso rápido a dados produzidos por vários tipos de câmaras e o encaminhamento desses dados para os respetivos algoritmos desenvolvidos, como por exemplo os de seguimento. Através do OpenCV, o ROS dispõe de bibliotecas que podem ser usadas para calibrar as câmaras, processar imagens *stereo* e monoculares e ainda imagens de profundidade, independentemente da interface usada para a ligação ao sistema (USB, Ethernet) [25].

Do ponto de vista da simulação 3D, o ROS possui plugins para emparelhamento com o Gazebo, um software realista, para ambientes interiores e exteriores e com um motor de física incorporado. Além do mais, sendo as interfaces de troca de mensagens entre os plugins e o resto do ecossistema ROS iguais, é possível desenvolver Nós de ROS compatíveis com a simulação e hardware utilizado. Após a simulação, a aplicação pode ser enviada para o sistema físico do robô com poucas ou nenhuma alteração no código desenvolvido [24].

Na ótica da perceção a PCL (*Point Cloud Library*) foca a manipulação e processamento de dados a três dimensões e imagens em profundidade. Entre as funcionalidades presentes estão diversos algoritmos de *point cloud* incluído filtragem, deteção, registo e outros. A biblioteca PCL auxilia na recolha, transformação, visualização e processamento de dados de sensores tridimensionais como é o caso da Microsoft Kinect [24].

Não obstante as capacidades avançadas do ROS foi criado um projeto *open-source* que as estende às diversas áreas da indústria. O ROS-Industrial (Figura 5) [26] inclui interfaces para os sistemas robóticos mais comuns a nível industrial, sejam manipuladores,

sensores ou redes de dispositivos. Adicionalmente existem bibliotecas de software para calibração automática de sensores 2D e 3D, processamento de caminhos e planeamento de trajetórias. O ROS-I é suportado por um consorcio internacional da indústria do qual fazem parte grandes empresas a nível mundial [27].



Figura 5: Logótipo do ROS Industrial

### 3.5 Preparação do workspace ROS

O sistema oficial de criação de *workspaces* no ROS é o catkin. Este combina macros de CMake e scripts de Python, permitindo uma melhor distribuição de packages, melhor suporte para compiladores de diferentes plataformas e melhor portabilidade [28].

De maneira a conseguir instalar o projeto num sistema Linux, é necessário correr os seguintes comandos:

#### Criar o Package

##### Criar o Workspace

```
source /opt/ros/<distro>/setup.bash
mkdir -p ~/catkins_ws1/src
cd ~/catkin_ws1
catkin_make
source ~/catkin_ws1/devel/setup.bash
cd
echo "source ~/catkin_ws1/devel/setup.bash" >> ~/.bashrc
```

Copiar o ficheiro `launch` para a pasta `~/catkin_ws1/src/<nome pkg>/Launch`  
Copiar o ficheiro `packages.xml` para a pasta `~/catkin_ws1/src/<nome pkg>/`

```
mkdir ~/kinect
cd ~/kinect
git clone https://github.com/OpenNI/OpenNI.git
cd OpenNI/Platform/Linux/CreateRedist/
chmod +x RedistMaker
sudo ./RedistMaker
cd ../Redist/OpenNI-Bin-Dev-Linux-x64-v1.5.2.23/
sudo ./install.sh
```

```
cd ~/kinect
git clone https://github.com/avin2/SensorKinect.git
cd SensorKinect/Platform/Linux/CreateRedist
sudo ./RedistMaker
cd ../Redist/Sensor-Bin-Linux-x64-v5.1.0.25/
chmod +x install.sh
sudo ./install.sh
```

```
cd ~/kinect
wget https://www.dropbox.com/sh/jsygeyx8dwa49w6/AAAFcQc_JczB_KdtVTj7RHUFa/NiTE%20v1_5_2_23?dl=0&preview=NITE-Bin-Li
nux-x64-v1.5.2.23.tar.zip
tar -xvf NITE-Bin-Linux-x64-v1.5.2.23.tar.zip
cd NITE-Bin-Linux-x64-v1.5.2.23/Data
```

Modificar os ficheiros *Sample-Scene.xml*, *Sample-Tracking.xml* e *Sample-User.xml*  
Trocar:

```
<License vendor="PrimeSense" key=""/>
```

Por:

```
<License vendor="PrimeSense" key="0KOIk2JeIBYClPwVnMoRKn5cdY4="/>
```

```
cd ..
sudo ./install.sh
```

```
cd ~/catkin_ws1/src/
git clone https://github.com/ros-drivers/openni_tracker.git
cd ..
catkin_make
catkin_make install
```

## Lançar o Projeto

Abrir 4 instâncias do terminal (por SSH ou localmente)

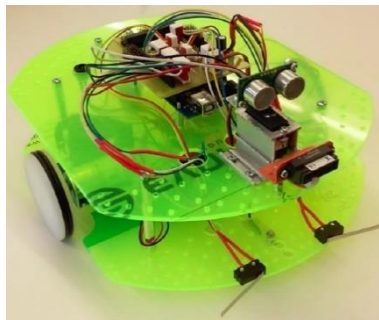
1. `roslaunch <nome do projeto> demo_launch.launch`
2. `rosserial`
3. `roslaunch oppenni_tracker oppenni_tracker`
4. `roscd <nome do projeto> && python kinect.py`

## Capítulo 4: Arquitetura de Hardware

Neste capítulo será explorado a composição das várias versões que constituíram esta plataforma, sublinhando com grande importância que a ideologia por detrás dela sofreu alterações de versão para versão.

### 4.1 Primeira Versão do GreenT (2015)

O GreenT é uma plataforma robótica diferencial desenvolvida integralmente no Instituto Politécnico de Tomar em 2015 no âmbito do projeto Ciência Viva denominado por “Escolher Robótica, Escolher Ciência” [54]. O GreenT (Figura 6) versão 2015 é baseado numa placa de desenvolvimento Arduino Mega 2560, e está vocacionado para o ensino da robótica móvel a principiantes.



*Figura 6: GreenT (2015)*

Como habitual, os robôs diferenciais são compostos por duas rodas motrizes ligadas a uma drive de potência (ponte H) para possibilitar o movimento horizontal positivo e negativo (para a frente e para trás respetivamente) e por uma roda livre, ou roda castor, sendo a trajetória do movimento dependente da diferença de velocidades entre as rodas de tração.

Esta versão tinha uma panóplia de sensores que possibilitavam a implementação de algoritmos de seguimento de linha, de desvio de obstáculos e uma versão rudimentar de seguimento de objetos. Nesta lista de sensores constam, um sensor infravermelho, para medir distâncias, um sonar, também para medir distâncias, mas através de ondas de ultrassons, três sensores de linha infravermelho, para algoritmos de seguimento de linha, dois *bumpers* para deteção de colisão e, por fim, um servo motor para possibilitar o varrimento de 180 graus horizontais com o sonar.

De forma a conseguir realizar a comunicação e a obtenção de dados entre os componentes utilizou-se uma placa de desenvolvimento Arduino, mais concretamente o Arduino Mega 2560.

A figura 7 mostra uma breve representação da arquitetura desta versão do GreenT, a forma como os componentes comunicam e estão conectados à placa de desenvolvimento Arduino.

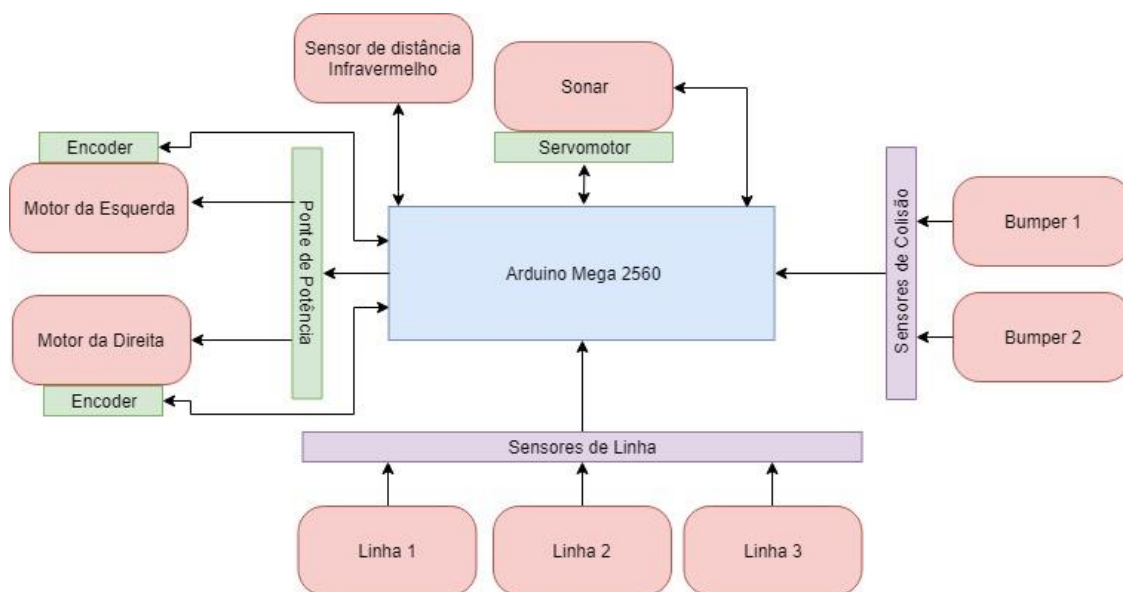


Figura 7: Diagrama de blocos da arquitetura inicial.

#### 4.1.1 Arduino Mega 2560

A placa Arduino Mega 2560 (Figura 8) utiliza um microcontrolador ATmega2560 e dispõe de 256 KB de memória flash. Tem 54 (cinquenta e quatro) portas de I/O, 15 das quais com capacidade de ser utilizadas como saídas PWM. Estas plataformas são, normalmente, programadas na linguagem de programação C ou C++ e é fortemente apreciada por jovens programadores devido à sua simplicidade e o seu potencial. Este dispositivo é o cérebro por trás de toda a movimentação do robô.

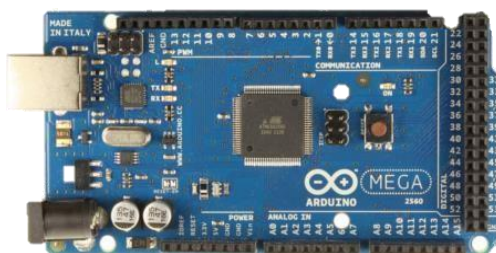


Figura 8: Arduino Mega 2560

## 4.2 Nova Versão do GreenT – GreenTROS (2018)

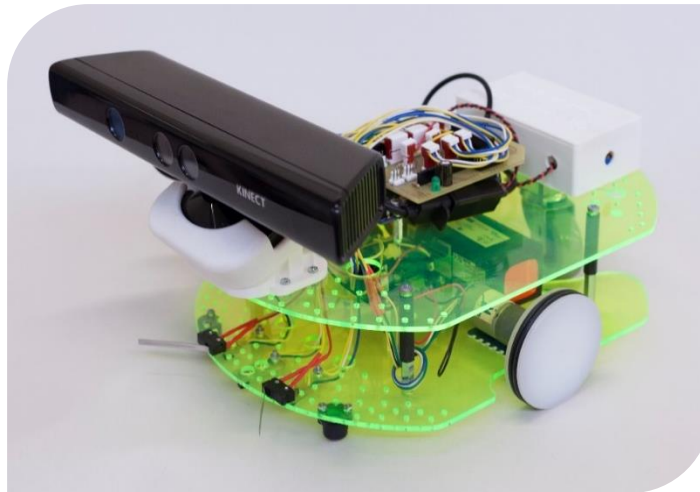


Figura 9: APS (2018)

Para possibilitar a integração do ROS, houve a necessidade de reformular e re-projetar a plataforma GreenT 2015. De entre as alterações necessárias, aquela que surge com mais expressão é necessidade de aumentar a capacidade de processamento e memória. A utilização da placa de desenvolvimento Arduino Mega 2560 verifica-se, neste sentido, como insuficiente. Por outro lado, a utilização de um processador de maior capacidade de processamento e memória, neste caso um Raspberry Pi 3B, permite ainda adicionar novos sensores, tais como um sensor Kinect, que darão um novo potencial ao GreenT e permitirão a realização de novas aplicações robóticas que até agora lhe estavam vedadas, tais como a utilização de métodos de navegação mais sofisticados baseados, por exemplo, em planeamento de trajetórias (globais, locais ou híbridas), havendo ainda a possibilidade de se determinar a localização do robô com maior precisão.

Após a reestruturação do robô (Figura 9), o GreenT deixou de ser um robô didático para passar a ser um assistente pessoal sénior. Como referido, a plataforma sofreu alterações substanciais, onde os sensores iniciais da plataforma foram substituídos por apenas uma Kinect, mas mantendo o Arduino de modo a permitir a comunicação entre o Raspberry Pi e o GreenT, funcionando apenas como plataforma base de motores e *encoders* (Figura 10). Neste cenário, o Arduino serve apenas como interveniente às aplicações de baixo nível, mais concretamente dar instruções aos motores a que velocidade devem girar, pois todos cálculos são realizados no Raspberry.

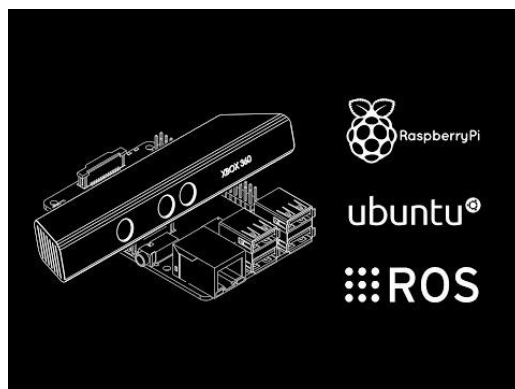


Figura 10: Camada aplicativa da nova arquitetura.

Contudo, as alterações não se ficaram pelos sensores, o sistema de alimentação passou de acumuladores recarregáveis Ni-MH para uma bateria LiPO de 11.1V e 4000mAh.

Com estas mudanças, o robô passou de um mero robô móvel com pouca inteligência e capacidade evolutiva limitada, para um robô bastante avançado com detecção de pessoas em tempo real.

A figura 11 representa a arquitetura da mais recente versão do GreenT, como os componentes comunicam e estão conectados à placa de desenvolvimento Arduino, como o Raspberry interage com o Arduino, e o novo sistema de alimentação.

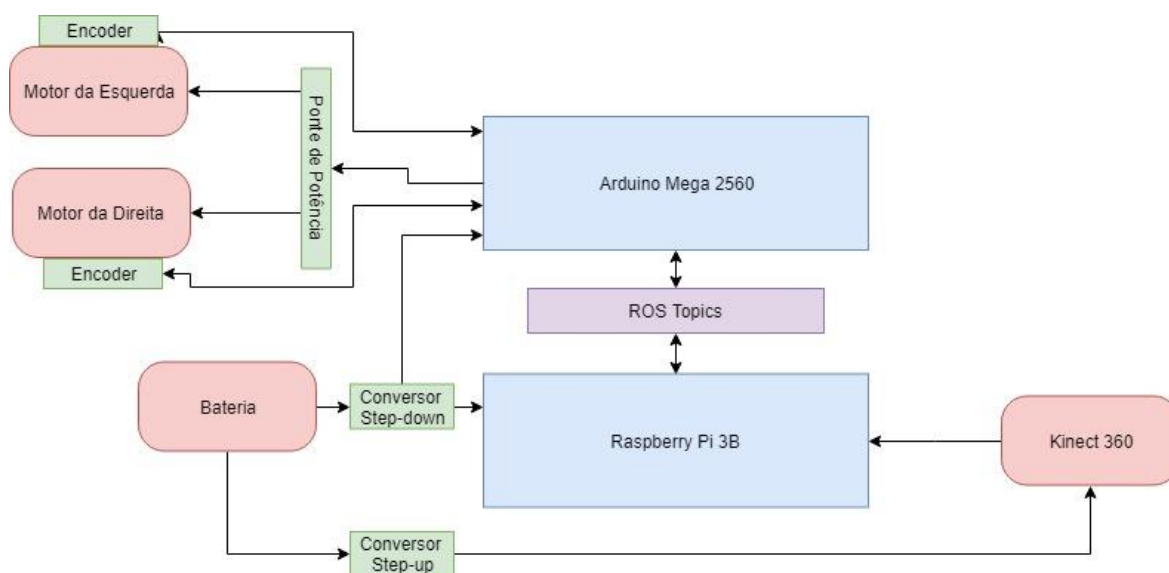


Figura 11: Diagrama de Blocos da nova arquitetura.

### 4.2.1 Raspberry Pi

O Raspberry Pi é uma placa de desenvolvimento, baseado num processador ARM (arquitetura RISC) e é um SoC. O Raspberry Pi 3B, contém 1GB de RAM, comunicações sem fios (Wi-Fi e Bluetooth), Gigabit Ethernet, quatro portas USB, quarenta pinos GPIO, uma porta HDMI e uma *slot* para um cartão SD.

É maioritariamente usado na área de robótica pois suporta sistemas operativos da família Debian, ou seja, Ubuntu, CentOS, Fedora, ou até mesmo o sistema operativo criado pela própria empresa, o Raspbian. No entanto, existe uma versão do sistema operativo Windows que corre nesta plataforma, o Windows 10 IoT Core.

Motivado pela maior necessidade de processamento e pelo facto de o ROS necessitar de correr sobre um sistema operativo Ubuntu, escolheu-se a Raspberry Pi 3B (Figura 12) como o hardware de suporte ao *middleware* ROS.

Tabela 1: Comparação de Vantagens e Desvantagens de um Raspberry

| Vantagens   | Desvantagens  |
|---|---|
| Pequeno   | Não corre sistemas operativos x86                                     |
| Baixo custo   | Algumas distribuições Linux e Windows mais comuns não são compatíveis |
| Oferece poder de computação suficiente para fazer de servidor | CPU com baixa capacidade de processamento                             |
| Boa eficiência energética                                     |   |



Figura 12: Raspberry Pi 3B



#### 4.2.2 Kinect 360

A Kinect é uma câmara na qual tem como principal objetivo permitir a deteção de movimento para a consola Xbox 360. Foi construída e desenvolvida em 2010 pela empresa Microsoft em conjunto com a atual subsidiária Apple, PrimeSense.

Tal como referido, a Kinect dispõe de vários sensores com propósitos diferentes. Estes componentes são:

- Uma câmara RGB;
- Uma câmara Monocromática;
- Um sensor de profundidade infravermelho;
- Um *array* de cinco microfones;
- Processador proprietário;
- Um motor *tilt*.

Através desta junção de sensores, o poder desta câmara era inigualável e chamou a atenção de muitos programadores e criadores na área da robótica. Contudo a ligação requerida para a obtenção de dados é proprietária da Microsoft, obrigando os interessados em utilizá-la a desenvolver um adaptador que permitiria a conversão para uma entrada USB tipo B. Aquando do lançamento da Kinect para Windows, a Microsoft lançou um transformador com conversão em entrada USB.

Após vários anos desde o seu lançamento, esta criação permitiu que os vários sensores existentes no mercado tiveram como base a Kinect. Um exemplo mais recente é o iPhone X que utiliza uma versão muito reduzida dos sensores da Kinect para o desbloqueio com reconhecimento facial.

Neste projeto optou-se pela primeira versão da Kinect (Figura 13) em detrimento da mais recente por questões de consumo energético, de forma a manter a plataforma portátil. Acresce ainda o fato de a versão mais recente utilizar a norma USB 3.0, a qual não é suportada pelo Raspberry Pi, o que iria trazer problemas de largura de banda na ligação entre os dois componentes.

Este sensor, servindo-se da maior capacidade de computação do Raspberry Pi 3, permite expandir o leque de funcionalidades e complexidade que podem ser aplicados no GreenT.



*Figura 13:* Microsoft Kinect 360

## Capítulo 5: Arquitetura de Software

O ROS sendo uma plataforma *open-source*, modular, escalável e multifuncional, acumula pormenores que lhe conferem um grau de complexidade elevado.

Por forma a obter uma curva de aprendizagem suave foram constituídos os seguintes objetivos que compuseram a metodologia de trabalho:

- Tutoriais de ROS
- Comunicação serial ROS - Arduino (rosserial)
- Implementação de ROS no Raspberry Pi (Package)
- Controlo de malha aberta
- Controlo através de joystick em malha fechada
- Implementação do sensor Kinect
- Controlo através de imagem (Seguimento de linha)
- Implementação de OpenNi
- Seguimento de uma pessoa baseado na identificação de um utilizador, com base num sensor Kinect.

## 5.1 Tutoriais de ROS

Dada a falta de familiarização com o *middleware*, optou-se pela realização de alguns tutoriais para melhor compreender o funcionamento de packages de ROS e para tomar conhecimento de algumas ferramentas como o Turtlesim (Figura 14), bibliotecas como a *tf* [32] e aplicações como o *rviz*.

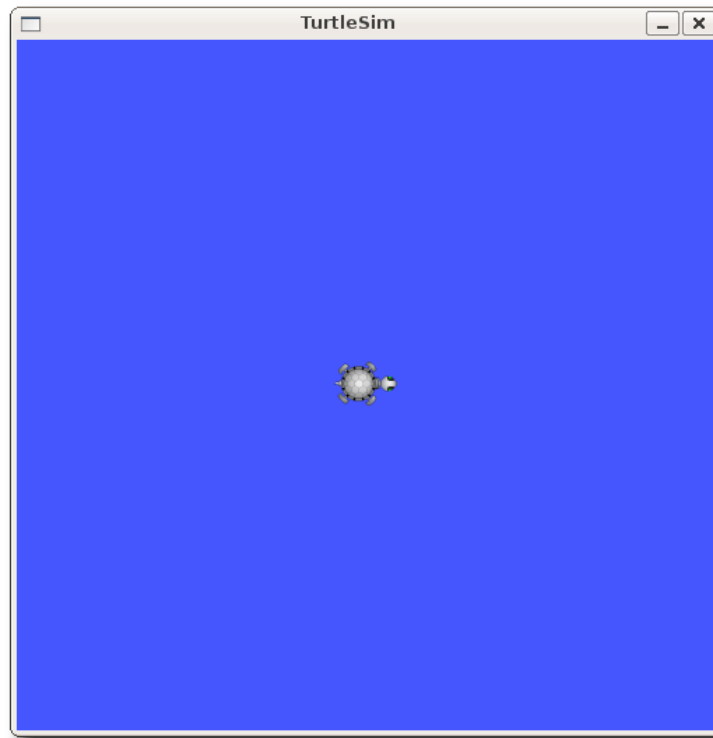


Figura 14: Janela de simulação do Turtlesim

O *turtlesim* é uma ferramenta concebida para o ensino de ROS e consequente exploração dos pacotes ROS [29].

O Nó ROS que a aplicação gera, providencia um simulador simples para aprender os conceitos de ROS.

Este Nó faz a subscrição aos comandos de velocidade linear e angular (*cmd\_vel*) para o *turtleX*, designação da tartaruga visível na janela da Figura 15 e conseguindo-se desta forma a simulação do movimento da tartaruga de acordo com os comandos de velocidade subscritos. Por outro lado, o Nó do *turtlesim* faz a publicação da pose do *turtleX* que se constitui nas posições relativas a cada eixo (*x*, *y*, *theta*) e nas velocidades atuais (linear e angular).

A biblioteca *tf*, já abordada no Capítulo 3, foi usada em conjunto com o *turtlesim* para consolidar conhecimentos relativos aos múltiplos referenciais que podem existir num sistema (*frames*).

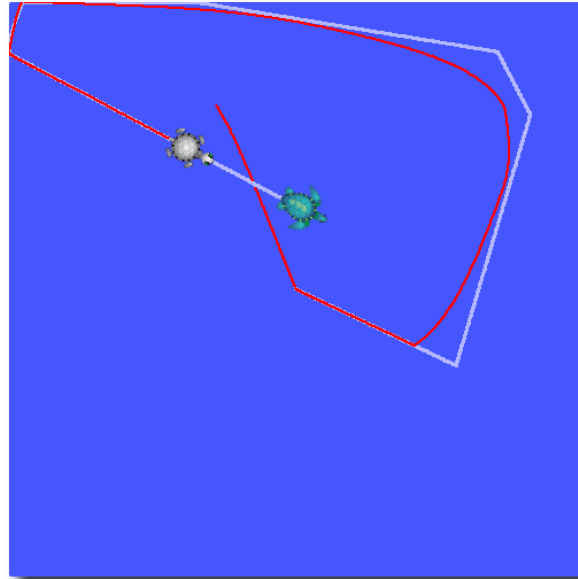


Figura 15: Simulação turtlesim

Neste caso a turtle1 (verde) era controlada através do teclado e a turtle2 (cinzento) seguia a primeira. Este tutorial usava a biblioteca *tf* para criar três referenciais: o referencial base, o referencial da turtle1 e o referencial da turtle2. Um *tf broadcaster* fazia a publicação das coordenadas das tartarugas e um *tf listener* efetuava a computação da diferença entre os referenciais, fazendo com que uma tartaruga seguisse a outra.

Por fim foi usado o *rviz* para melhor visualizar e examinar os diferentes referenciais *tf* e adquirir alguma experiência nessa aplicação.

## 5.2 Comunicação série ROS - Arduino (rosserial)

Seguidamente optou-se pela realização de testes de pequena dimensão, focados na integração de ROS com o Arduino Mega. O *rosserial* é um protocolo que permite o envio de Tópicos, Mensagens e serviços ROS para uma porta série.

Foi utilizada a biblioteca *rosserial\_arduino* para possibilitar a comunicação entre o ROS e o Arduino Mega do projeto. Inicialmente foram testados os sensores e motores que ainda estavam ligados ao GreenT na sua versão inicial.

Através da programação do Arduino foram acionados os motores em malha aberta, sendo a introdução de velocidades feita através da linha de comandos do Linux, pelo ROS. O script continha, portanto, apenas uma subscrição para o `cmd_vel` e uma função que acionava os motores (Figura 16), sendo totalmente feita no microcontrolador.

```
void messageCb( const geometry_msgs::Twist& msg) {  
    moveFront(msg.linear.x, msg.linear.y);  
}  
  
ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", &messageCb );
```

Figura 16: Exemplo de um subscritor num Arduino

De seguida procedeu-se à operação inversa, que consistiu na realização da leitura de sensores na linha de comandos do Linux, mantendo a programação do lado do Arduino. Desta vez criou-se um Publisher que transmite a informação lida de um sonar (Figura 17).

```
std_msgs::Float32 sonar_msg;  
ros::Publisher pub_sonar("sonar", &sonar_msg);  
ros::NodeHandle nh;
```

Figura 17: Exemplo de um Publisher num Arduino

Por fim, consolidaram-se ambos os casos através do acionamento de um servo motor e leitura da respetiva posição através da linha de comandos do Linux.

### 5.3 Implementação do ROS no Raspberry Pi

Uma vez concluídos os exemplos de funcionamento do *workspace* ROS e testadas as funcionalidades elementares foi necessário assegurar a comunicação entre o hardware utilizado.

### 5.4 Controlo em malha aberta

O primeiro grande objetivo após a estruturação do ambiente base foi efetuar o controlo do GreenT através do comando dos seus motores em malha aberta.

Nesta primeira versão foram desenvolvidos dois Nós ROS programados em Python (a vermelho na Figura 18) e implementada, em C para Arduino, a integração do microcontrolador com o ROS (a azul na Figura 18).



Figura 18: Primeira versão da arquitetura de Nós ROS do GreenT

O microcontrolador Arduino Mega na fase inicial do projeto era responsável pelas operações de baixo nível, aplicando os comandos aos motores obtidos através dos valores de `cmd_vel` subscritos e realizando a publicação dos dados relativos ao posicionamento dos codificadores óticos incrementais (*encoders*). No entanto o Arduino efetuava, ainda, a operação necessária à biblioteca `tf` através da publicação dos dados de Twist dos referenciais base e do GreenT.

Os dados do movimento linear e angular foram inseridos diretamente no código do Arduino, sendo o comportamento do robô predefinido.

No Nó ROS “Odometry” eram feitos os cálculos da pose, alimentando-se dos Tópicos ROS disponibilizados pelo Arduino, através de `rosserial`, recebia os dados sobre os referenciais atual e alvo. Após o cálculo de *tf* esta era publicada juntamente com as informações dos *encoders*.

O Nó ROS “teleOp” recebia os dados necessários para responder às perguntas “onde é que eu estou?” e “para onde vou?”, e, comparando as velocidades atuais dos motores com

as velocidades necessárias para se dirigir com determinada velocidade angular e linear, enviava os comandos `cmd_vel` ao Arduino que por fim atuava os motores.

## 5.5 Controlo através de joystick com controlo em malha fechada

Na segunda iteração do projeto (Figura 19) restringiu-se as funções do Arduino exclusivamente aos procedimentos de baixo nível, passando o controlo dos motores a ser realizado em malha fechada. O comando de velocidade passou a ser gerado por meio de um joystick da Microsoft Xbox 360 – Figura 20.



Figura 19: Arquitetura de Nós ROS do GreenT com controlo por Joystick



Nesta segunda versão foram desenvolvidos quatro Nós ROS programados em Python e implementada, em C para Arduino, a integração do microcontrolador com o ROS.



Figura 20: Joystick Xbox

O Arduino fazia apenas a subscrição dos comandos de velocidade que aplicava aos motores e a publicação dos valores dos *encoders*

O Nó ROS responsável pela leitura de comandos de velocidade fornecidos pelo joystick, “greenT\_joy”, recebe através dos drivers do sistema operativo referentes ao joystick ligado ao Raspberry Pi, por via USB, comandos de movimento que variavam de -1 a 1 ao longo de quatro eixos. Estes valores eram convertidos em Mensagens ROS de Twist, sendo ainda feita a publicação das mesmas.

Os dados da mensagem Twist que estavam a ser aplicados no GreenT eram recebidos pelo *Subscriber* existente no Nó ROS “greenT\_teleop”. Este Nó efetuava os cálculos para determinar que velocidades, linear e angular, eram necessárias para realizar o movimento indicado pelo Twist. Por fim, as velocidades alvo eram transmitidas através da publicação do Tópico ROS “wheel\_vtarget”.

Combinando os valores desse Tópico ROS referente aos *encoders* com as velocidades alvo referidas anteriormente, o Nó ROS “greenT\_pid\_vel” efetuava o controlo PID, com base no erro entre a velocidade atual, inferida com base nos dados dos *encoders* e a velocidade desejada, a qual é providenciada pelo comando do joystick. Foi implementada a publicação da velocidade atual do GreenT em metros por segundo, que poderá vir a ser útil dado a escalabilidade da arquitetura ROS.

Por último o Nó ROS “greenT\_tf” passou a calcular e a disponibilizar os dados relativos à odometria, pose e *tf* do GreenT que poderão ser usados em expansões futuras do projeto.

## 5.6 Implementação do sensor Kinect

Com o intuito de aproveitar a maior capacidade de computação conferida ao GreenT pelo Raspberry Pi foi integrado um sensor Kinect de primeira geração.

A biblioteca ROS freenect providencia as interfaces necessárias para controlar e aceder ao hardware da Kinect. São suportados o acionamento dos motores, o acesso às imagens RGB e de profundidade, acelerómetro, LED e áudio.

Uma vez que com a Kinect os comandos de velocidade para o robô iriam deixar de ser enviados com base em teleoperação, através do joystick, e passariam a ser efetuados com base no processamento de dados fornecidos por um sensor Kinect, foram necessários alguns testes para que o algoritmo a implementar tivesse sucesso. Numa primeira fase a Kinect foi simplesmente ligada a um computador a correr o sistema operativo Ubuntu, com o objetivo de aceder à câmara e visualizar os seus conteúdos no *rviz* através do ROS.

Foram instaladas as dependências e Nós ROS da biblioteca freenect e através do *freenect\_launch* foi possível aceder aos *Publishers* da Kinect que puderam ser observados no *rviz*.

Com os dados necessários disponíveis foram testados alguns Nós ROS em Python que permitissem guardar as imagens da câmara para que fosse possível aplicar-lhes o algoritmo de seguimento desejado.

O primeiro grande obstáculo surgiu na manipulação das imagens dentro dos Nós ROS, uma vez que as Mensagens ROS provenientes da câmara eram do tipo ROS *Image Message*. Por forma a converter as imagens para OpenCV, suportado pelo Python, usou-se a biblioteca de ROS CvBridge (Figura 21) [30].

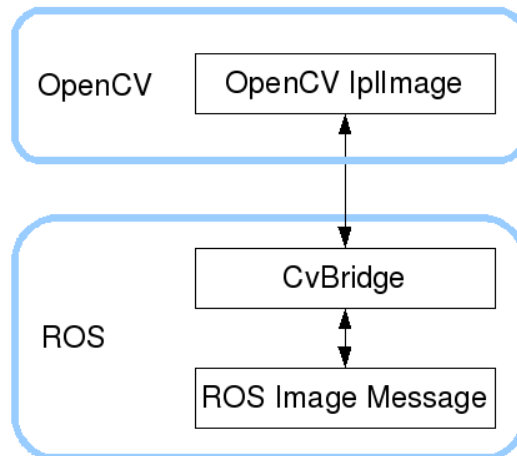


Figura 21: Estrutura de comunicação entre OpenCV e o ROS

O conceito desta biblioteca assenta no facto de o ROS trocar imagens em mensagens no seu formato próprio, *sensor\_msgs/image*. Incluída no *package cv\_bridge*, a biblioteca CvBridge faz a interface entre o ROS e o OpenCV.

O primeiro nó de teste realizado, *greenT\_get\_video*, do qual se pode observar um excerto na Figura 22, pode ser explicado da seguinte forma:

1. O *Subscriber* deste Nó ROS recebe os dados da imagem pelo *Publisher* da *freenect*.
2. Através da biblioteca CvBridge efetua a conversão da imagem
3. O método “*processimage*”, calcula e retorna a decisão da direção que o APS deve tomar.
4. O Twist (velocidades lineares e angulares) relativo a esse deslocamento é transmitido pelo *Publisher* deste Nó ROS.

```

from cv_bridge import CvBridge, CvBridgeError

class image_converter:

    def start(self):
        global pub
        self.previousD = ""
        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("/camera/rgb/image_color", Image, self.callback)
        pub = rospy.Publisher('twist', Twist, queue_size=10)
        rospy.init_node('image_converter')
        rospy.spin()

    def callback(self, data):
        twist = Twist()
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "rgb8")
            #cv2.imwrite("frame.jpg", cv_image)

            #self.decision, self.angle = getAngle("frame.jpg")
            self.decision = processimage(cv_image)

```

Figura 22: Exemplo de código Python para extrair uma imagem e processar os dados

## 5.7 Seguimento de linha baseado no sensor Kinect

Na iteração seguinte do projeto juntou-se o conhecimento adquirido sobre a plataforma ROS em termos de deslocamento e controlo do APS e o uso de processamento de imagem como meio de geração dos comandos de velocidade para o robô.

Neste terceiro desenvolvimento as alterações incidiram sobre o Nó ROS que gera os comandos de velocidade. Programou-se em Python um Nó ROS com o intuito de substituir a ação do joystick, passando os comandos associados à mensagem Twist (velocidades lineares e angulares) a serem dependentes dos dados obtidos pelo sensor Kinect.

O ROS Node lançado pela biblioteca freenect (a verde na Figura 23), efetua a publicação da imagem da câmara.



Figura 23: Arquitetura de nós ROS utilizando a Kinect

O nó `greenT_Kinect`, criado nesta versão do projeto, adquiria a imagem através da subscrição ao Tópico “`camera/rgb/image_color`” e guardava a mesma no armazenamento SD do Raspberry Pi. Pelo meio da conversão `CvBridge` o algoritmo de seguimento de linha (abordado no capítulo 6), dispunha dos dados necessários ao cálculo do Twist (velocidades lineares e angulares) o qual enviava através do *Publisher* para consumo dos nós implementados anteriormente e assim mover o APS de forma adequada.

## 5.8 Implementação do OpenNi

OpenNi ou *Open Natural Interaction* é um software que visa a certificação e melhoramentos nos sistemas de interface natural, tanto para dispositivos como para aplicações e *middleware*. Um dos fundadores da organização era a empresa responsável pela tecnologia base da Kinect.

No que diz respeito ao ROS, este dispõe de um pacote, `openni_launch`, que permite aceder aos dados da câmara e convertê-los para imagens de profundidade e *point clouds*.

No âmbito deste projeto foi usado um outro pacote, `openni_tracker`, que por sua vez transmite já as linhas do esqueleto de uma pessoa bem como os pontos de articulação, na forma de referenciais da biblioteca `tf` [31].

O `openni_tracker`, após a deteção de um novo utilizador identificado pela pose de rendição, efetua a publicação do Tópico `openni_depth_frame`. Este Tópico contém dois campos: (i) `frame` que se constitui dos seguintes nomes: `/head`, `/neck`, `/torso`, `/left_shoulder`, `/left_elbow`, `/left_hand`, `/right_shoulder`, `/right_elbow`, `/right_hand`, `/left_hip`, `/left_knee`, `/left_foot`, `/right_hip`, `/right_knee`, `/right_foot`; e (ii) `user` que guarda a lista de todos os utilizadores detetados.

## 5.9 Seguimento de uma pessoa com base num sensor Kinect

A última iteração efetuada ao nosso projeto consistiu na consolidação de todos os conhecimentos adquiridos ao longo do mesmo, integrando o hardware do APS, o *middleware* ROS, a comunicação entre as diversas partes do sistema, a arquitetura de controlo do APS, o uso de processamento de imagem para inferir os comandos de velocidade para o robô e a aplicação do algoritmo de seguimento de pessoa, abordado em detalhe no Capítulo 6.

O Nó ROS lançado pela biblioteca *openni\_tracker* (a roxo na Figura 24), através do seu *Publisher*, disponibiliza então os dados da biblioteca *tf* referentes às diferentes partes do corpo, após a detecção de uma pessoa.

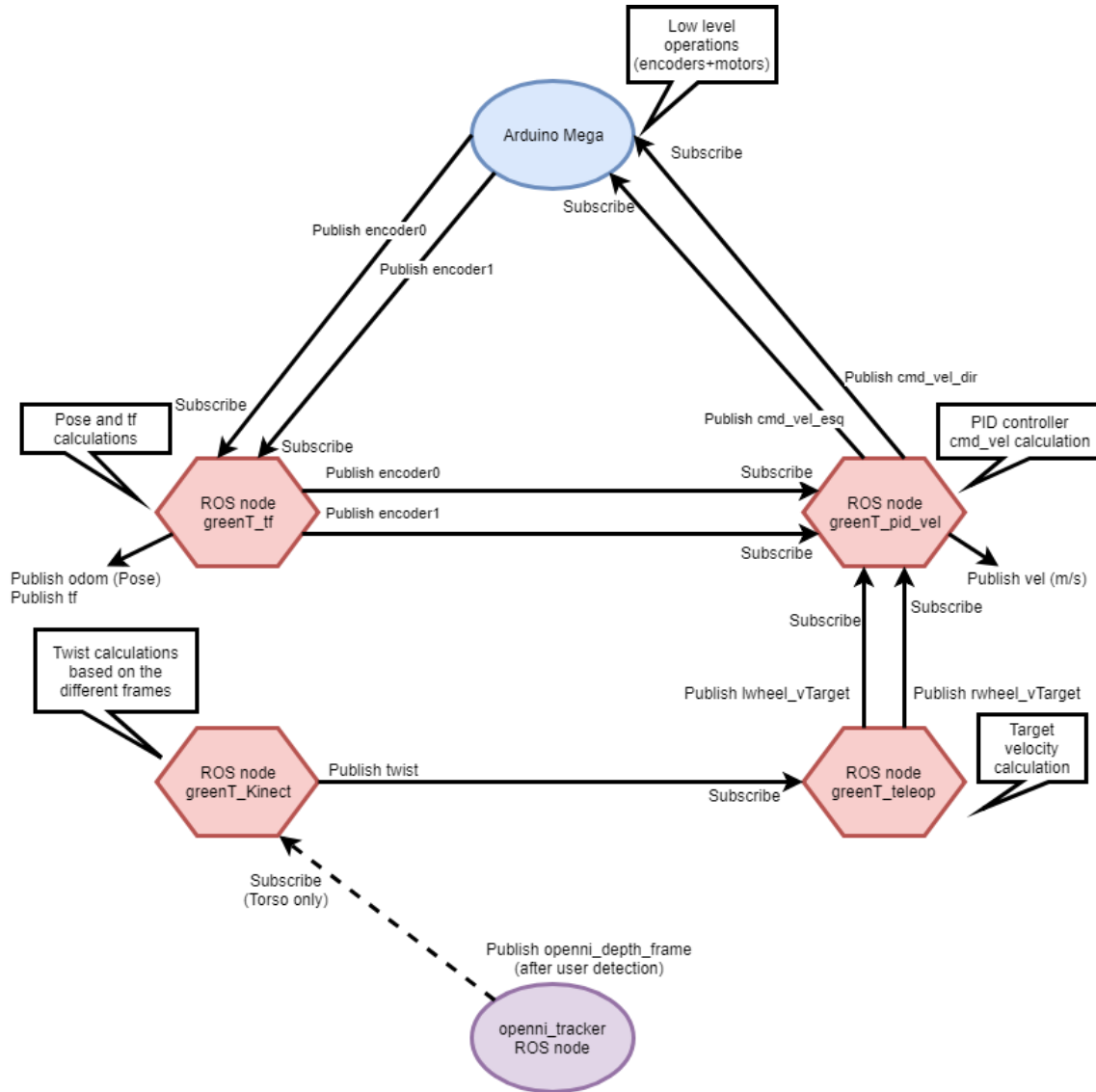


Figura 24: Arquitetura de nós ROS atual do robô

No que diz respeito à arquitetura o *greenT\_kinect* foi alterado para implementar o *Subscriber* ao módulo do torso do Tópico *openni\_depth\_frame* e aplicado o algoritmo de seguimento de pessoa tendo em conta os dados da biblioteca *tf*, intercetando os referenciais do APS e da pessoa a seguir. Após tomada a decisão, o *Twist* (velocidades lineares e angulares) é disponibilizado e o *Publisher* transmite os dados para a restante arquitetura.

## 5.10 Características dos componentes ROS

Os diversos Nós ROS que constituem o sistema possuem uma série de entradas e saídas que estão identificadas nas tabelas seguintes (Tabelas 2 a 7) bem como o trabalho de computação realizado por cada um destes.

Tabela 2: ROS *openni\_tracker*

| Nó criado pela biblioteca <i>openni_tracker</i> |   |
|---|---|
| Descrição                                       |   |
| <b>Dados de entrada</b>                         | Informações adquiridas pela câmara  |
| <b>Computação efetuada</b>                      | Programação proprietária  |
| <b>Dados de saída (Tópicos)</b>                 | Publicação <i>openni_depth_frame</i> – <i>frame</i> e respetivo <i>user</i> |

Tabela 3: ROS Node *greenT\_Kinect*

| Nó programado em Python para calcular o Twist |  |
|---|--|
| Descrição                                     |  |
| <b>Dados de entrada (Tópicos)</b>             | Subscrição <i>openni_depth_frame</i> – Dentro deste Tópico seleciona o utilizador e a <i>frame</i> do Torso apenas.  |
| <b>Computação efetuada</b>                    | Através dos dados da posição dos referenciais do robô e da pessoa, aplica o algoritmo para decidir que Twist (velocidades lineares e angulares) deve ser aplicado para efetuar o seguimento. |
| <b>Dados de saída (Tópicos)</b>               | Publicação Twist – velocidades linear e angular  |



Tabela 4: ROS Node greenT\_teleop

| Descrição                            | Nó programado em Python para calcular a velocidade alvo   |
|--------------------------------------|---|
| <b>Dados de entrada</b><br>(Tópicos) | Subscrição Twist – Captura as velocidades linear e angular enviadas pelo Twist  |
| <b>Computação</b><br><b>efetuada</b> | Através das equações do deslocamento determina a que velocidades devem as rodas andar por forma a mover o APS com a velocidade pedida pelo Twist. |
| <b>Dados de saída</b><br>(Tópicos)   | Publicação rwheel_vtarget – velocidade alvo para a roda direita<br>lwheel_vtarget – velocidade alvo para a roda esquerda                          |

Tabela 5: ROS Node greenT\_pid\_vel

| Descrição                            | Nó programado em Python para efetuar o controlo PID   |
|--------------------------------------|---|
| <b>Dados de entrada</b><br>(Tópicos) | Subscrição rwheel_vtarget – velocidade alvo para a roda direita<br>lwheel_vtarget – velocidade alvo para a roda esquerda<br>encoder0 – valor do <i>encoder</i> direito<br>encoder1 – valor do <i>encoder</i> esquerdo   |
| <b>Computação</b><br><b>efetuada</b> | Através dos parâmetros base de controlo PID determina a velocidade atual e tem em consideração o erro no cálculo da diferença entre a velocidade atual e a velocidade pretendida. Após os cálculos, determina que comandos se devem dar ao motor para que este gira à velocidade desejada, tendo em conta a velocidade a que já se encontram. |
| <b>Dados de saída</b><br>(Tópicos)   | Publicação cmd_vel – comando de velocidade para o motor direito<br>cmd_vel1 – comando de velocidade para o motor esquerdo<br>vel – velocidade atual do robot em metros por segundo  |

Tabela 6: ROS Node greenT\_tf

| Descrição                            | Nó programado em Python para disponibilizar a odometria  |
|--------------------------------------|--|
| <b>Dados de entrada</b><br>(Tópicos) | Subscrição encoder0 – valor do <i>encoder</i> direito<br>encoder1 – valor do <i>encoder</i> esquerdo   |
| <b>Computação</b><br><b>efetuada</b> | Através do valor dos <i>encoders</i> este Node estima a mudança de posição ao longo do tempo do robot, relativamente ao ponto inicial. É efetuado o cálculo da distância percorrida bem como as velocidades e consoante o intervalo de tempo e posição inicial é estimada a posição atual. |
| <b>Dados de saída</b><br>(Tópicos)   | Publicação encoder0 – valor do <i>encoder</i> direito<br>encoder1 – valor do <i>encoder</i> esquerdo<br>odom – posição do robot em todos os eixos<br>tf – frame do robot fica disponibilizada  |

Tabela 7: ROS Node roserial Arduino Mega

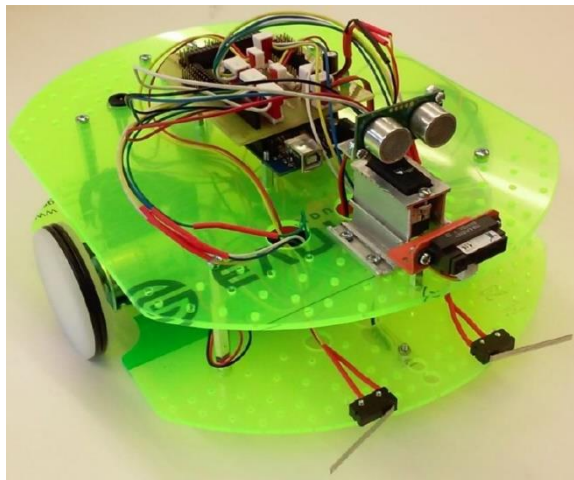
| Descrição                            | Nó programado em C para Arduino criado pelo Arduino Mega   |
|--------------------------------------|--|
| <b>Dados de entrada</b><br>(Tópicos) | Subscrição cmd_vel – comando de velocidade para o motor direito<br>cmd_vel1 – comando de velocidade para o motor esquerdo  |
| <b>Computação</b><br><b>efetuada</b> | O microcontrolador apenas realiza as operações de baixo nível, acionando os motores através de funções e após as leituras de hardware, disponibiliza os valores dos <i>encoders</i> na forma de Tópicos ROS. |
| <b>Dados de saída</b><br>(Tópicos)   | Publicação encoder0 – valor do <i>encoder</i> direito<br>encoder1 – valor do <i>encoder</i> esquerdo   |

## Capítulo 6: Aplicações

Neste capítulo serão explicados em detalhe o funcionamento dos algoritmos utilizados pelas várias versões deste assistente pessoal.

### 6.1 Seguimento de Linha

Este algoritmo é dos mais simples de usar devido à quantidade de hardware necessária para realizar todas as tarefas que possibilitam o seguimento de linha.



*Figura 25: Primeira versão do GreenT*

Em muitos casos, o seguimento de linha é feito com base em sensores infravermelhos, que, ao detetar uma mudança de cor (claro ou escuro) enviam ao processador apropriado, neste caso um Arduino, um sinal de 0 ou 1. Ou seja, pegando na primeira versão do GreenT (Figura 25) que estava equipado com 3 sensores de linha, quando o robô está perante uma linha a 90° na vertical os sensores da esquerda, centro e direita iriam enviar ao Arduino os valores 1 1 1 respetivamente, fazendo com que o robô fizesse os cálculos de modo a conseguir o momento horizontal no sentido positivo (andar em frente). Contudo, se a linha tiver um ângulo obtuso, ou seja, uma linha com orientação para a direita, os sensores transmitiam os valores 0 1 1 (ou 0 0 1 dependendo do ângulo de incisão), fazendo com que a roda esquerda girasse a uma velocidade maior que a roda direita, de modo a possibilitar a rotação à direita.

Por vezes, de maneira a complementar e a tornar o robô mais completo, são usados sensores de distância infravermelhos e sonares de modo a possibilitar o desvio de obstáculos.

Começando pelo sonar, este envia ondas ultrassônicas através de um emissor e calcula o tempo que a onda demorou a retornar ao receptor. Com esta informação e com a utilização de um algoritmo conversor de tempo em distância, conseguimos saber que um objeto se encontra naquele espaço, seja uma parede ou um dedo. Aliado a este sensor tínhamos um sensor de distância infravermelho, que, apesar de ter a mesma função que o sonar, difere na maneira de obtenção de dados, enquanto o sonar utiliza sons este usa feixes de luz invisível ao olho humano, mas consegue uma leitura precisa da distância. No caso de todos os sensores falharem, por motivo alheio, existem dois *bumpers* de maneira a haver uma prevenção neste tipo de situações.

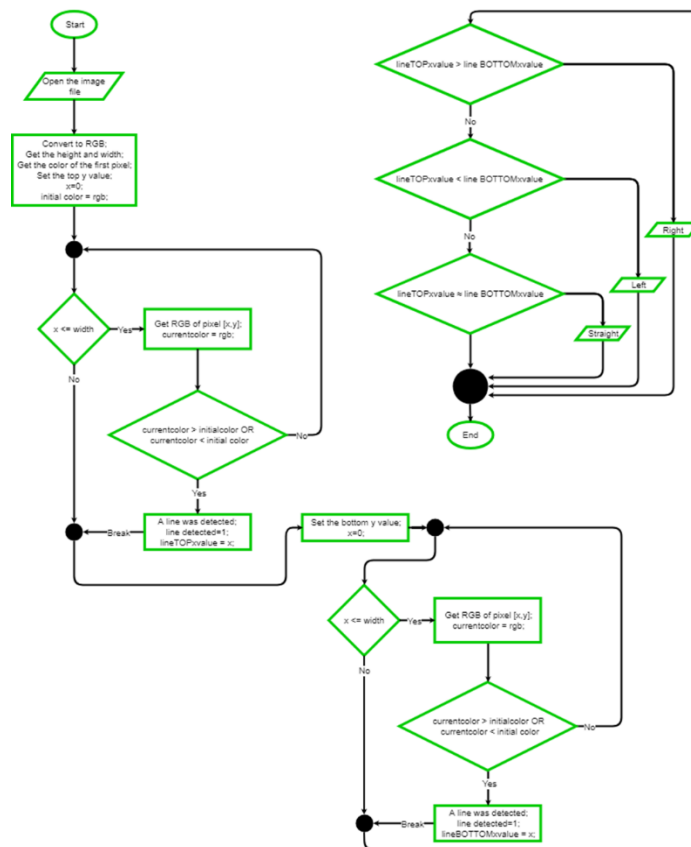


Figura 26: Fluxograma do seguimento de linha através da Kinect.  
(Anexo 1)

Posteriormente, na versão do GreenT com câmara Kinect ainda foi testada uma versão modificada deste algoritmo. No entanto, não foram utilizados nenhum dos sensores mencionados neste capítulo, apenas a Kinect. A maneira como conseguimos realizar tal tarefa foi através de uma biblioteca do ROS chamada Freenect, que trata da conexão e compreensão entre a câmara e o sistema operativo, e do módulo OpenCV para Python que

processa os dados enviados pela Kinect (Figura 26). Ao lançar o script criado por nós, o primeiro passo é criar uma ligação à câmara e receber os dados enviados por ela, de seguida utilizar um script, também criado por nós, de maneira a processar os dados recebidos com o auxílio dos vários métodos disponibilizados pelo OpenCV. Este script é responsável por realizar um varrimento de todos os pixéis que perfazem uma imagem, verificando quando existe uma mudança de cor na parte superior e inferior da imagem, guarda as coordenadas da matriz de dados, e analisa se existe um desfasamento nas coordenadas: se a coordenada superior for maior que a inferior significa que deve virar à direita e vice-versa. Todo este processo é realizado sem nunca se gravar uma única imagem no Raspberry Pi. Deste modo consegue-se reduzir os requisitos computacionais, em grande parte, que seriam necessários caso fosse necessário remover e guardar a imagem em formato “.jpg”. Este problema resolveu-se com a aplicação de um dos métodos do OpenCV, que converte dados genéricos de uma câmara (neste caso a Kinect) num *Array* legível pelo OpenCV e por todo o *software* desenvolvido. Após este processamento, é calculado a que velocidade as rodas devem girar e por fim enviam-se dados ao nó de odometria.

Contudo, verificámos que esta câmara era demasiado poderosa para esta finalidade face às necessidades do algoritmo, porém o desenvolvimento desta aplicação ajudou-nos no desenvolvimento do objetivo primário que consiste na deteção e seguimento de pessoas.

## **6.2 Seguimento de Pessoas**

Após iniciarmos o seguimento de linha, verificámos que a visão por computador pode ter um papel central na robótica. Em termos leigos a visão por computador consiste na capacidade de um computador simular a visão humana e fazer reconhecimento de pessoas, bem como deteção de obstáculos. Para isso, recorreu-se ao sensor Kinect, que apesar de ser um sensor demasiado poderoso para aplicações simples como o seguimento de linha, para este cenário é um componente essencial devido às suas características e funcionalidades (como foi explicado no capítulo 4, subcapítulo 4.2.2).

De maneira a conseguir tal aplicação, usou-se uma biblioteca do ROS chamada Openni tracker, que apesar de não ter manutenção nem atualização de software desde 2015, ainda possibilita a sua utilização nas versões mais atuais do ROS, e um script criado por nós

de modo a aceder à biblioteca do Openni e realizar todos os cálculos necessários ao reconhecimento e seguimento de pessoa.

O script funciona de uma maneira muito simples. Em primeiro lugar verifica se existe uma instância da biblioteca, se não existir este suspende a sua execução e avisa o utilizador sobre o erro, caso contrário passa à obtenção de dados, mais concretamente das coordenadas x y z do torso da pessoa. De notar que este script apenas funciona para uma pessoa de cada vez. Em seguida, após receber as coordenadas, é realizada uma verificação, tanto a nível do eixo dos X como nos Y, isto para saber se o utilizador realizou algum movimento em comparação com o valor de x ou y inicial e os valores anteriores.

Caso tenha realizado movimento, é utilizada uma função de mapeamento semelhante à existente na linguagem de programação para Arduino, de modo a obter uma velocidade, lateral e horizontal, de acordo com as distâncias na qual o utilizador se encontra. No caso de o utilizador estar muito próximo, ou sair fora do ângulo de visão da câmara, a velocidade será zero de modo a parar o robô.

Após os cálculos será enviada, através de mensagens do tipo twist, as velocidades para o nó de odometria.

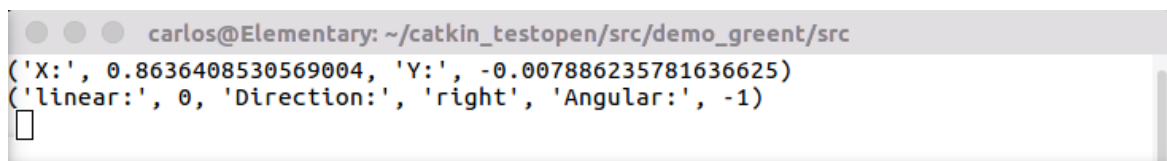
### **6.3 Resultados Experimentais**

Depois de toda a parte de computação e lógica, passámos à implementação de toda a camada de *software* no Raspberry Pi. Contudo, derivado às características inerentes deste e das condições necessárias ao bom funcionamento do *software* criado, não foi possível implementar o projeto no Raspberry Pi. No entanto, de forma a validar os nossos dados, utilizou-se um computador com sistema operativo Ubuntu + ROS e a ferramenta ROSBag para aquisição de dados.

A forma como conseguimos validar que o nosso algoritmo é algo viável, o robô foi colocado num suporte em que lhe permitisse ter as rodas motrizes livres enquanto realizávamos os testes. Desta maneira, conseguimos obter uma perceção do comportamento do robô de acordo com os dados obtidos através da biblioteca Openni Tracker (Figura 29), que é responsável pela identificação de pessoas e da publicação das coordenadas de cada

parte do corpo (explicado em detalhe no Capítulo 5, subcapítulo 5.8), mesmo estando este estacionário.

Posteriormente, após iniciar o nosso *software* (Figura 27) e a identificação de uma pessoa através do OpenNI, é possível ver na ferramenta de simulação Rviz o percurso e a trajetória do robô (Figura 28 do lado direito), bem como a representação esquelética da pessoa identificada (Figura 28 do lado esquerdo)

A terminal window with a title bar showing 'carlos@Elementary: ~/catkin\_testopen/src/demo\_greent/src'. The terminal displays two lines of output: ('X:', 0.8636408530569004, 'Y:', -0.007886235781636625) and ('linear:', 0, 'Direction:', 'right', 'Angular:', -1). A cursor is visible on the line following the second output.

```
carlos@Elementary: ~/catkin_testopen/src/demo_greent/src
('X:', 0.8636408530569004, 'Y:', -0.007886235781636625)
('linear:', 0, 'Direction:', 'right', 'Angular:', -1)
█
```

Figura 27: Cálculo de velocidade e extração da posição do utilizador

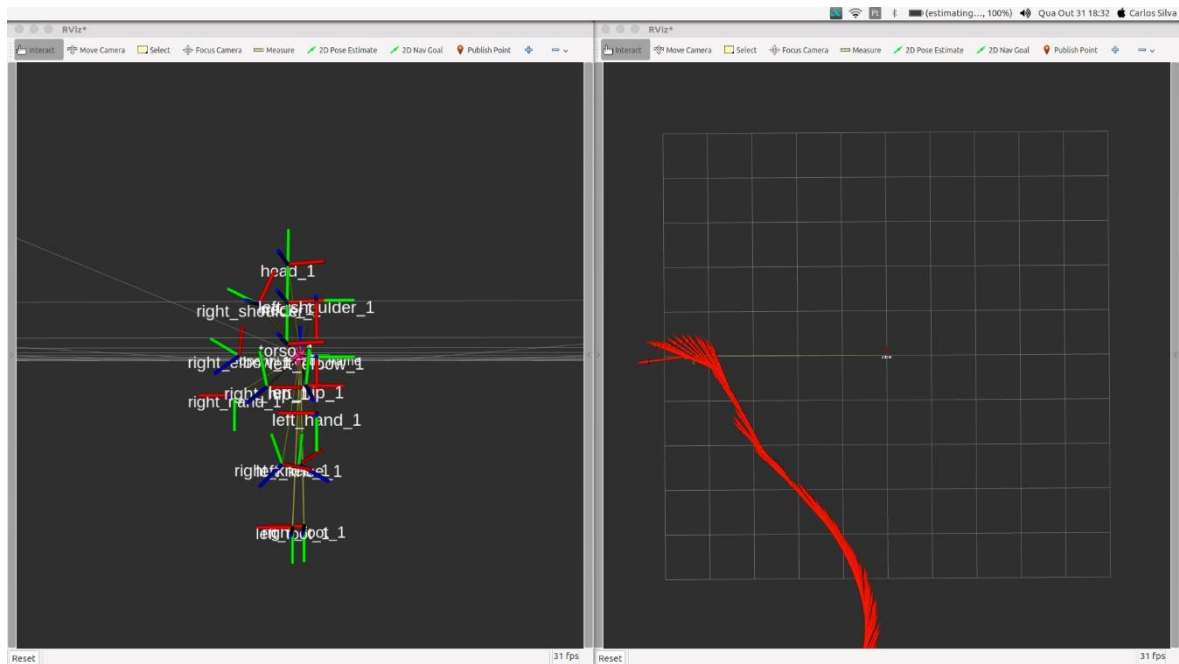


Figura 28: Representação gráfica de uma pessoa (Lado Esquerdo) e do robô Assistente Pessoal Sênior (Lado Direito)

```

carlos@Elementary: ~
carlos@Elementary:~$ roslaunch openni_tracker openni_tracker
[ INFO] [1541002684.923249832]: New User 1
[ INFO] [1541002687.406323180]: Pose Psi detected for user 1
[ INFO] [1541002687.441962942]: Calibration started for user 1
[ INFO] [1541002689.213770477]: Calibration complete, start tracking user 1
^Ccarlos@Elementary:~$ roslaunch openni_tracker openni_tracker
[ INFO] [1541004138.768649046]: New User 1
[ INFO] [1541004140.102476394]: Pose Psi detected for user 1
[ INFO] [1541004140.133631322]: Calibration started for user 1
[ INFO] [1541004145.645672231]: Calibration complete, start tracking user 1
^Ccarlos@Elementary:~$ roslaunch openni_tracker openni_tracker
[ INFO] [1541010180.930580424]: New User 1
[ INFO] [1541010182.465270422]: Pose Psi detected for user 1
[ INFO] [1541010182.497018030]: Calibration started for user 1
[ INFO] [1541010183.603292598]: Calibration complete, start tracking user 1

```

Figura 29: Detecção de uma pessoa usando Openni Tracker



## Capítulo 7: Conclusões e Trabalho Futuro

### 7.1 Conclusão

Neste relatório foi realizado um estudo intensivo sobre a plataforma ROS e como integrar dita plataforma de acordo com as necessidades do projeto.

O objetivo de atualizar o robô GreenT [54] implementando a *framework* ROS foi concluído com sucesso, demonstrando-se fiável durante toda a execução de testes mencionados ao longo deste relatório. O objetivo secundário de implementar algoritmos de seguimento de pessoas, apesar de não ser o principal foco deste trabalho, foi concluído, também, com sucesso. O ROS provou o seu poder ao permitir a criação, fácil, de métodos para o GreenT, bem como testar os ditos métodos num ambiente de simulação.

Contudo, apesar de os objetivos terem sido cumpridos, houve uma grande dificuldade nos processos intermédios. O primeiro impacto com a plataforma ROS foi algo complexo, devido ao grau de conhecimento necessário de modo a conseguir ter, com sucesso, uma estrutura viável e escalável. Os tutoriais existentes no *website* da plataforma apenas cobrem uma parte do conhecimento necessário, fazendo com que seja necessárias horas de pesquisa em fóruns, que, em muitas das vezes, não contêm respostas aos problemas.

A maior dificuldade encontrada no decorrer do projeto foi a implementação do *software* no Raspberry Pi, pois não foi possível realizar a transição do sistema vindo de um computador. Este problema deve-se à arquitetura do processador do Raspberry Pi, arquitetura essa que não corresponde aos requisitos de *software* do OpenNI, biblioteca de identificação de pessoas através de um sensor Kinect, que sem essa correspondência não funciona corretamente.

### 7.2 Trabalho Futuro

Para assegurar o melhor desempenho da plataforma é necessário realizar mais testes em diferentes condições. No entanto, derivado do tempo restante para a conclusão do projeto, algumas funcionalidades e opções ficaram por investigar.

O próximo objetivo deste projeto será encontrar uma alternativa ao uso da Kinect no Raspberry Pi e reestruturar a camada aplicativa de modo a permitir o uso da nova alternativa.

## Capítulo 8: Bibliografia

- [1] PORDATA, “PORDATA - População residente: idade mediana,” [Online]. Available:  
<https://www.pordata.pt/Europa/Popula%C3%A7%C3%A3o+residente+idade+median+2265>. [Acedido em 5 1 2018].
- [2] PORDATA, “PORDATA - população residente: total e por grandes grupos etários (%)” [Online]. Available:  
[https://www.pordata.pt/Portugal/Popula%C3%A7%C3%A3o+residente+total+e+por+grandes+grupos+et%C3%A1rios+\(percentagem\)-3018](https://www.pordata.pt/Portugal/Popula%C3%A7%C3%A3o+residente+total+e+por+grandes+grupos+et%C3%A1rios+(percentagem)-3018). [Acedido em 5 1 2018].
- [3] PORDATA, “PORDATA - Indicadores de envelhecimento,” [Online]. Available:  
<https://www.pordata.pt/Portugal/Indicadores+de+envelhecimento-526>. [Acedido em 5 1 2018].
- [4] PORDATA, “PORDATA - Índices de envelhecimento segundo os Censos,” [Online]. Available:  
<https://www.pordata.pt/Municipios/%C3%8Dndice+de+envelhecimento+segundo+os+Censos-348>. [Acedido em 5 1 2018].
- [5] ROS, “Is ROS for me” [Online]. <http://www.ros.org/is-ros-for-me/> [Acedido em 27-08-18]
- [6] Edx, “Hello Real World with ROS” [Online]. <https://www.edx.org/course/hello-real-world-with-ros-robot-operating-system> [Acedido em 02-09-18]
- [7] Wikipedia, “MD5” [Online]. <https://en.wikipedia.org/wiki/MD5> [Acedido em 30-08-18]
- [8] Mathworks, “Exchange Data with ROS Publishers” [Online]. <https://fr.mathworks.com/help/robotics/examples/exchange-data-with-ros-publishers.html;jsessionid=2a8a2fbd094a537b2251d3506692> [Acedido em 27-08-18]

- [9] ROS, “Core Components” [Online]. <http://www.ros.org/core-components/>  
[Acedido em 11-08-2018]
- [10] ROS, “Quickstart in RVIZ Tutorial” [Online].  
[http://docs.ros.org/kinetic/api/moveit\\_tutorials/html/doc/quickstart\\_in\\_rviz/quickstart\\_in\\_rviz\\_tutorial.html](http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/quickstart_in_rviz/quickstart_in_rviz_tutorial.html) [Acedido em 27-08-2018]
- [11] ROS, “Nodes” [Online]. <http://wiki.ros.org/Nodes> [Acedido em 27-07-2018]
- [12] ROS, “Topics” [Online]. <http://wiki.ros.org/Topics> [Acedido em 27-07-2018]
- [13] ROS, “TCPROS” [Online]. <http://wiki.ros.org/TCPROS> [Acedido em 27-07-2018]
- [14] ROS, “Messages” [Online]. <http://wiki.ros.org/Messages> [Acedido em 27-07-2018]
- [15] ROS, “Twist” [Online].  
[http://docs.ros.org/api/geometry\\_msgs/html/msg/Twist.html](http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html)  
[Acedido em 30-07-2018]
- [16] Wikipedia, “Robot Operating System” [Online].  
[https://en.wikipedia.org/wiki/Robot\\_Operating\\_System](https://en.wikipedia.org/wiki/Robot_Operating_System) [Acedido em 04-06-2018]
- [17] ROS, “tf” [Online]. <http://wiki.ros.org/tf> [Acedido em 27-07-2018]
- [18] ROS, “rqt\_graph” [Online]. [http://wiki.ros.org/rqt\\_graph](http://wiki.ros.org/rqt_graph) [Acedido em 03-08-2018]
- [19] ROS, “rqt\_plot” [Online]. [http://wiki.ros.org/rqt\\_plot](http://wiki.ros.org/rqt_plot) [Acedido em 03-08-2018]
- [20] ROS, “rqt\_topic” [Online]. [http://wiki.ros.org/rqt\\_topic](http://wiki.ros.org/rqt_topic) [Acedido em 16-08-2018]
- [21] ROS, “rqt\_publisher” [Online]. [http://wiki.ros.org/rqt\\_publisher](http://wiki.ros.org/rqt_publisher) [Acedido em 16-08-2018]
- [22] ROS, “rqt\_bag” [Online]. [http://wiki.ros.org/rqt\\_bag](http://wiki.ros.org/rqt_bag) [Acedido em 03-08-2018]
- [23] ROS, “Bags” [Online]. <http://wiki.ros.org/Bags/Format> [Acedido em 03-08-2018]

- [24] ROS, “Integration” [Online]. <http://www.ros.org/integration/> [Acedido em 06-09-2018]
- [25] OpenCV “OpenCV” [Online]. <https://opencv.org/> [Acedido em 16-08-2018]
- [26] ROS Industrial, “ROS Industrial” [Online]. <https://rosindustrial.org/> [Acedido em 14-08-2018]
- [27] ROS Industrial, “Current Members” [Online]. <https://rosindustrial.org/ric/current-members/> [Acedido em 14-08-2018]
- [28] ROS, “Catkin Conceptual Overview” [Online]. [http://wiki.ros.org/catkin/conceptual\\_overview](http://wiki.ros.org/catkin/conceptual_overview) [Acedido em 03-08-2018]
- [29] ROS, “Turtlesim” [Online]. <http://wiki.ros.org/turtlesim> [Acedido em 03-09-2018]
- [30] ROS, “CV\_Bridge” [Online]. [http://wiki.ros.org/cv\\_bridge](http://wiki.ros.org/cv_bridge) [Acedido em 03-09-2018]
- [31] ROS, “Openni Tracker” [Online]. [http://wiki.ros.org/openni\\_tracker](http://wiki.ros.org/openni_tracker) [Acedido em 10-10-2018]
- [32] ROS, “Introduction to tf”, [Online]. <http://wiki.ros.org/tf/Tutorials/Introduction%20to%20tf> [Acedido em 22-06-2018]
- [33] Wikipedia, “Licença BSD”, [Online]. <https://pt.wikipedia.org/wiki/LicençaBSD> [Acedido em 27-07-2018]
- [34] Universidade de Stanford, “Perspetivas in Assistive Technologies” [Online]. <https://web.stanford.edu/class/engr110/2012/04b-Jaffe.pdf> [Acedido em 31-10-2018]
- [35] Universidade de Stanford, “Minerva” [Online]. [http://robots.stanford.edu/papers/thrun.icra\\_minerva.pdf](http://robots.stanford.edu/papers/thrun.icra_minerva.pdf) [Acedido em 31-10-2018]

- [36] Wired, “Tug, the busy little robot, will see you now” [Online].  
<https://www.wired.com/story/tug-the-busy-little-robot-nurse-will-see-you-now/>  
 [Acedido em 20-10-2018]
- [37] Euronews, “Coimbra testa robôs para ajudar idosos” [Online]  
<https://pt.euronews.com/2017/02/20/coimbra-testa-robos-para-ajudar-idosos>  
 [Acedido em 07-08-2018]
- [38] Research Gate, “Assistive Social Robots in Elderly Care” [Online]  
[https://www.researchgate.net/publication/229058790\\_Assistive\\_social\\_robots\\_in\\_elderly\\_care\\_A\\_review](https://www.researchgate.net/publication/229058790_Assistive_social_robots_in_elderly_care_A_review) [Acedido em 04-09-2018]
- [39] Wikipedia, “Aibo” [Online] <https://en.wikipedia.org/wiki/AIBO> [Acedido em 13-10-2018]
- [40] Operating System, “Aperios” [Online]. [https://www.operatingsystem.org/betriebssystem/\\_english/bs-aperios.htm](https://www.operatingsystem.org/betriebssystem/_english/bs-aperios.htm) [Acedido em 13-10-2018]
- [41] Wikipedia, “Paro” [Online]. [https://en.wikipedia.org/wiki/Paro\\_\(robot\)](https://en.wikipedia.org/wiki/Paro_(robot)) [Acedido em 23-10-2018]
- [42] Wall Street Journal, “It’s Not a Stuffed Animal, It’s a \$6000 Medical Device” [Online].  
<https://www.wsj.com/articles/SB10001424052748704463504575301051844937276>  
 [Acedido em 31-10-2018]
- [43] Robotics Today, “Pearl” [Online]. <https://www.roboticstoday.com/robots/pearl-description> [Acedido em 20-10-2018]
- [44] Telegraph, “Meet Pearl, She’s the Robo Nurse to Look After the Elderly” [Online].  
<https://www.telegraph.co.uk/news/worldnews/northamerica/usa/1457427/Meet-Pearl-shes-the-robo-nurse-designed-to-look-after-the-elderly.html> [Acedido em 27-10-2018]
- [45] Semantic Scholar, “RoboCare: an Integrated Robotic System for the Domestic Care of the Elderly” [Online].

<https://pdfs.semanticscholar.org/4717/08337028d1def2db5dc16c42b3d12bcc3922.pdf> [Acedido em 30-10-2018]

- [46] Quora, “Which programming Language Does Boston Dynamics Use for Their Robot” [Online]. <https://www.quora.com/Which-programming-language-does-Boston-Dynamics-use-for-their-Robot> [Acedido em 31-10-2018]
- [47] GazeboSim, “Atlas control over ROS with Python” [Online]. [http://gazebo.org/tutorials?cat=&tut=drcsim\\_ros\\_python](http://gazebo.org/tutorials?cat=&tut=drcsim_ros_python) [Acedido em 31-10-2018]
- [48] Boston Dynamics, “Atlas” [Online]. <https://www.bostondynamics.com/atlas> [Acedido em 25-07-2018]
- [49] Wikipedia, “KLT: Kanade-Lucas-Tomasi feature tracker” [Online]. [https://en.wikipedia.org/wiki/Kanade-Lucas-Tomasi\\_feature\\_tracker](https://en.wikipedia.org/wiki/Kanade-Lucas-Tomasi_feature_tracker) [Acedido em 21-10-2018]
- [50] Universidade de Yonsei, “Kanade-Lucas-Tomasi Tracker” [Online]. <http://web.yonsei.ac.kr/jksuhr/articles/Kanade-Lucas-Tomasi%20Tracker.pdf> [Acedido em 30-10-2018]
- [51] Wikipedia, “Feature Extraction” [Online]. [https://en.wikipedia.org/wiki/Feature\\_extraction](https://en.wikipedia.org/wiki/Feature_extraction) [Acedido em 21-10-2018]
- [52] Wikipedia, “Computer Vision” [Online]. [https://en.wikipedia.org/wiki/Computer\\_vision](https://en.wikipedia.org/wiki/Computer_vision) [Acedido em 21-10-2018]
- [53] Wikipedia, “Pyramid” [Online]. [https://en.wikipedia.org/wiki/Pyramid\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Pyramid_(image_processing)) [Acedido em 21-10-2018]
- [54] Ana Lopes; Carlos Ferreira; Gabriel Pires; Paulo Coelho; Pedro Correia; Pedro Neves – GreenT, “Escolher Robótica, Escolher Ciência!”, Tomar, 2015

## Capítulo 9: Anexos

### Anexo 1

