

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Robô com Unidade Principal de Processamento Implementada em FPGAs

Manuel Luís Campos Reis

Mestrado em Engenharia Electrotécnica e de Computadores

Orientador: João Manuel Paiva Cardoso (Prof. Dr.)

Co-orientador: João Paulo de Castro Canas Ferreira (Prof. Dr.)

Julho de 2009

A Dissertação intitulada

“ROBÔ COM UNIDADE DE PROCESSAMENTO PRINCIPAL IMPLEMENTADA EM FPGAs”

foi aprovada em provas realizadas em 23/Julho/2009

o júri



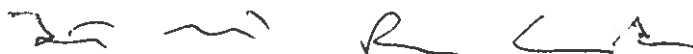
Presidente Professor Doutor José Alfredo Ribeiro da Silva Matos

Professor Catedrático do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto



Professor Doutor Mário Pereira Véstias

Professor Coordenador do Departamento de Engenharia Electrónica e Telecomunicações e de Computadores do Instituto Superior de Engenharia de Lisboa



Professor Doutor João Manuel Paiva Cardoso

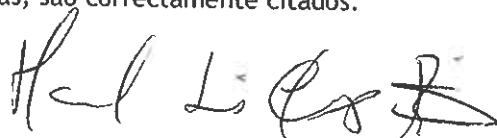
Professor Associado do Departamento de Engenharia Informática da Faculdade de Engenharia da Universidade do Porto



Professor Doutor João Paulo de Castro Canas Ferreira

Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.



Autor - MANUEL LUIS CAMPOS REIS

Faculdade de Engenharia da Universidade do Porto

Resumo

O desenvolvimento de sistemas embebidos requer conhecimentos em áreas muito abrangentes. Actualmente, os sistemas embebidos exigem elevada integração e funcionalidade, baixo consumo energético e dimensões reduzidas. Nos últimos tempos têm surgido soluções baseadas em FPGAs para providenciar a performance necessária. As possibilidades de projecto e a repartição de Software e Hardware nas proporções correctas são inúmeras.

Neste contexto, o campo da robótica móvel aparece como um domínio apropriado para avaliar sistemas embebidos como sistemas capazes de suportar e executar tarefas complexas de computação. Um dos objectivos deste trabalho era construir um protótipo de um robô móvel autónomo para implementar e estudar uma arquitectura de sistemas embebidos baseados em FPGAs. Como forma de avaliar o rendimento desta solução, o protótipo foi testado com algoritmos de mapeamento. Como plataforma robótica é utilizado o kit da VEX Robotics, pois foi considerada a melhor alternativa após um estudo preliminar a vários kits de robótica. O kit da VEX Robotics apresenta boa robustez e é composto por componentes que conferem flexibilidade na montagem e um baixo peso.

Para validação do protótipo escolhemos dois algoritmos de mapeamento utilizados na robótica móvel. Estes algoritmos foram seleccionados por serem predominantemente probabilísticos e por exigirem grande poder computacional. A implementação destes algoritmos baseia-se numa grelha de ocupação do ambiente. A esta grelha é sobreposto um modelo de incerteza do sensor para a actualização das probabilidades. São utilizados como sensores preferenciais o SONAR para detectar obstáculos e os codificadores de impulsos para determinar a postura do robô.

O resultado final é um robô em que a Unidade Principal de Processamento está embebida no FPGA, executando o algoritmo de mapeamento e controlo. O design embebido é optimizado com a inclusão de módulos de hardware adicionais, como memória Cache e *Floating Point Unit*, para aumentar o processamento em tempo real.

Palavras-chave: FPGA; sistemas embebidos; algoritmos probabilísticos; robótica; robôs móveis; grelhas de ocupação; mapeamento; robô autónomo.

Abstract

Embedded systems development requires a wide range of knowledge in several areas. Nowadays, embedded systems require low energy consumption and size as well as having to allow many functionalities and being easily integrated with other systems. Lately FPGA based solutions have been known to meet the performance requirements. They allow a choice between software implementation or implementing directly in a hardware module which leads to an immense range of possibilities for any given project.

In this context, the mobile robotics field is an appropriate domain where to evaluate embedded systems as systems with the capability of managing complex computing tasks. One of the proposed objectives of this project is to build an autonomous mobile robot prototype to study the possibility of a FPGA based architecture in this field. As a way to evaluate how efficient this solution is, the prototype was tested using mapping algorithms. The robotic platform used was the VEX Robotics Kit, after careful study of several other robotic kits. The VEX Robotics kit is robust and its components are flexible (assembly wise) and light.

To validate the prototype the two mapping algorithms used are often used in mobile robotics. The algorithms rely mainly in probabilities and require a lot of processing power. Their implementation is based on a environment occupation grid. This grid is overlaid with an uncertainty model from the sensor to update the probabilities. The chosen sensor to detect obstacles is the SONAR and shaft encoders are responsible for determining the robot's posture.

The end result is a robot in which the Main Processing Unit is embedded in the FPGA, and is responsible for the execution of the mapping and control algorithm. The embedded design is optimized by adding additional hardware modules, like cache memory and Floating Point Unit increasing its real time processing capabilities.

Key words: FPGA; embebbedd systems; probabilistic algorithms; robotics; mobile robots; occupancy grids; mapping;autonomous robot.

Agradecimentos

A aproximar-se o fim de uma etapa tão importante, há que preparar-me para um novo ciclo, que para mim significa poder exercer uma profissão que desde tenra infância me desperta curiosidade. Poderei enfim seguir em frente e deixar o meu contributo na Engenharia.

Nem sempre foi fácil frequentar o curso, mas penso que nunca o é quando o estudo se confronta com outras responsabilidades que se assume na vida.

Ao longo do meu percurso académico, profissional e pessoal contactei com muitas pessoas e situações que me influenciaram e que agora agradeço.

Por se tratar de um trabalho académico começarei por agradecer aos meus Professores, com destaque para o Professor Doutor João Cardoso pela tarefa de me orientar e pelos comentários oportunos em alturas difíceis. Ao Professor Doutor Canas Ferreira e restantes professores da secção de electrónica e sistemas digitais pelos contributos à minha progressão académica. Deixo ainda o meu apreço aos restantes professores, em especial ao Professor Doutor João Tasso, Professor Doutor João Paulo Sousa e Professor Doutor João Martins Ferreira pelo entusiasmo que me incutiram na minha escolha académica.

Agradeço os amigos e colegas da Faculdade de Engenharia da Universidade do Porto, e em especial ao Marc, pelas conversas proveitosas e apoio demonstrado.

Agradeço também aos amigos, o companheirismo e paciência que tiveram ao longo desta jornada.

Experimentei várias mudanças ao longo desta etapa, mas quando se traçam objectivos estas mudanças são necessárias. Deste modo agradeço à minha família e em especial à minha Mãe pelo seu apoio e intervenção em momentos cruciais.

Por último, mas nem por isso menos importante, expresso o meu reconhecimento à Ana pela sua companhia, amor e apoio incondicional que se revelaram determinantes para a realização deste objectivo.

Manuel Reis

*“Education is our passport to the future,
for tomorrow belongs to the people who prepare for it today.”*

Malcom X

Conteúdo

1	Introdução	1
1.1	Contexto	1
1.2	Objectivos	2
1.3	Estrutura da Dissertação	2
2	Mapeamento na Robótica	5
2.1	Introdução	5
2.2	Características e Perspectivas no Mapeamento	6
2.3	Desafios no Mapeamento na Robótica	7
2.4	Probabilidade na Robótica	8
2.5	Grelhas de Ocupação para o Mapeamento	8
2.6	Modelo de Sensor de SONAR	10
2.7	Métodos na Fusão de Leituras de Sensores	11
2.7.1	Bayes	11
2.7.2	Dempster-Shafer	13
2.7.3	Aplicação do Algoritmo	16
2.8	Resumo	18
3	Definição do Protótipo	21
3.1	Sistema	21
3.1.1	Estrutura Física	22
3.1.2	Sensores	22
3.1.3	Energia	23
3.1.4	Controlo e Processamento	23
3.2	Requisitos do Sistema	23
3.2.1	Requisitos Funcionais	23
3.2.2	Requisitos Não Funcionais	24
3.3	Kits Robóticos Comerciais	26
3.3.1	VEX Robotics	26
3.3.2	Mekatronix	26
3.3.3	Mobile Robots	27
3.3.4	Lego Mindstorm NXT	28
3.4	Comparação	28
3.5	Conclusão	29
4	Tecnologias	31
4.1	Kit Robótico VEX	31
4.1.1	Unidade de Controlo	31

4.1.2	Estrutura	38
4.1.3	Sensores	42
4.1.4	Energia	43
4.2	FPGA	44
4.2.1	Microblaze	45
4.2.2	Projecto	47
4.3	Resumo	48
5	Implementação do Protótipo	49
5.1	Subsistema Estrutura	49
5.1.1	Estrutura de Suporte	49
5.1.2	Propulsão	50
5.2	Subsistema Energia	52
5.3	Subsistema Controlo	52
5.3.1	Modos de Funcionamento	55
5.3.2	Lógica de Controlo	55
5.4	Resumo	57
6	Resultados Experimentais	59
6.1	Resultados de Referência no PC	59
6.2	Resultados Obtidos com o Sistema Embebido	61
6.3	Optimizações Específicas	62
6.4	Experiência no Campo	67
6.5	Conclusões	68
7	Conclusões e Trabalho Futuro	71
7.1	Satisfação dos Objectivos	71
7.2	Trabalho Futuro	71
	Referências	73
A	Especificações dos kits robóticos	77
A.1	VEX Robotics	77
A.2	Mekatronix	77
A.3	Mobile Robots	78
A.4	Lego Mindstorm NXT	78
B	Protocolo Simples de Transmissão Paralela	81
C	Protocolo de Transmissão de 4 bits	85
D	Código Fonte de Teste	87
E	Resultados de Profiling	99
E.1	Execução em PC	99
E.2	Execução no Sistema Caracterizado. μ Blaze@50MHz	103
E.3	Optimizações específicas no sistema embebido.	107
F	Bibliografia Adicional	115

Lista de Figuras

2.1	Exemplo de grelha de ocupação.	9
2.2	Modelo de sensor para o SONAR.	10
2.3	Um exemplo de descrição gráfica da Regra de Combinação de Dempster.	15
2.4	Resultado da Combinação de Dempster.	15
2.5	Grelha com exemplo de leitura.	17
2.6	Calculo de função possibilística <i>Bel3</i>	19
3.1	Diagrama de alto-nível do protótipo.	21
3.2	Subsistema estrutura.	22
3.3	Subsistema sensores.	22
3.4	Subsistema energia.	23
3.5	Subsistema de processamento e controlo.	24
3.6	Organização do protótipo.	25
3.7	Montagens possíveis para o kit robótico da VEX Robotics [1].	26
3.8	Kits robóticos disponibilizados pela Mekatronix [2].	27
3.9	Kits robóticos disponibilizados pela Mobile Robots [3].	27
3.10	Kit robótico da Lego [4].	28
4.1	Controlador da VEX Robotics [1].	32
4.2	Configuração interna da Unidade de Controlo.	33
4.3	Estrutura do programa original [5].	34
4.4	Diagrama de operação do programa principal.	35
4.5	Diagrama de fluxo do programa principal.	36
4.6	Diagrama de fluxo da função <i>Process_Data_From_Master_uP</i>	37
4.7	Diagrama de fluxo da função <i>User_Autonomous_Code</i>	38
4.8	Engrenagens e tipos de rodas dentadas.	40
4.9	Variação do <i>duty cycle</i> em PWM	40
4.10	Imagem do motor fornecido com o kit robótico [6].	41
4.11	Princípio de funcionamento de um SONAR [6]	42
4.12	Princípio de funcionamento de um codificador de impulsos [6].	43
4.13	Bateria fornecida com o kit robótico [6].	44
4.14	Arquitectura proposta para o Sistema embebido.	47
5.1	Parte inferior do robô.	50
5.2	p Parte frontal do robô.	51
5.3	Configuração das engrenagens de propulsão.	51
5.4	Circuito eléctrico de alimentação às unidades de controlo.	52
5.5	Conexões Físicas do Protótipo.	53
5.6	Arquitectura Interna do protótipo.	54

5.7	Protótipo implementado.	54
5.8	Modos de funcionamento do protótipo.	56
5.9	Diagrama de alto-nível do sistema de controlo.	56
6.1	Tabela de tempos acumulados (em <i>ms</i>) para para os níveis de optimização testados no PC.	61
6.2	Tabela de tempos acumulados para para os níveis de optimização testados no Microblaze@50MHz.	62
6.3	Exemplo de grelha com submatriz seleccionada.	63
6.4	Comparação de resultados para soluções de hardware diversas e para os algoritmos apresentados.	64
6.5	Tendência de crescimento do tempo de execução considerando dimensões da grelha crescentes.	65
6.6	Tendência de crescimento do tempo de execução considerando dimensões da grelha crescentes para o algoritmo optimizado.	66
6.7	Tendência de crescimento do tempo de execução considerando dimensões da grelha crescentes apenas para a função de actualização do algoritmo optimizado.	66
6.8	Tempos de execução da função de actualização em função da distância ao objecto.	67
6.9	Foto do ambiente a mapear.	68
6.10	Probabilidade de ocupação para cada célula (amarelo - inferior a 50%) (cinzento - superior a 50%) da grelha de ocupação utilizada pelo robô.	69
B.1	Port de E/S genérica do PIC18F8520 [7].	81
B.2	Atribuição de funções e conexões da UC VEX	82
B.3	Diagrama de Comunicação PSTP	83
C.1	Atribuição de funções e conexões da UC VEX.	85
C.2	Diagrama de Comunicação PT4B.	86

Lista de Tabelas

3.1	Tabela de avaliação das especificações de acordo com o Apêndice A	29
A.1	Specifications comparison table according to VEX Robotics website [1].	77
A.2	Specifications comparison table according to Mekatronix website [2].	78
A.3	Specifications comparison table according to Mobile Robots website [3].	79
A.4	Specifications comparison table according to Lego Mindstorm NXT website [4].	79
B.1	Comandos enviados pelo Processador do FPGA para a UC VEX	83
C.1	Comandos <i>non-encoding</i> do CPU do FPGA para a UC VEX.	86
E.1	Tabela de <i>Profiling</i> para dados do tipo <i>Double</i> e otimização O0.	99
E.2	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> e otimização O0.	100
E.3	Tabela de <i>Profiling</i> para dados do tipo <i>Double</i> e otimização O2.	100
E.4	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> e otimização O2.	101
E.5	Tabela de <i>Profiling</i> para dados do tipo <i>Double</i> e otimização O3.	101
E.6	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> e otimização O3.	101
E.9	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> e otimização O2 no sistema embebido.	103
E.10	Tabela de <i>Profiling</i> para dados do tipo <i>Double</i> e otimização O3 no sistema embebido.	104
E.11	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> e otimização O3 no sistema embebido.	105
E.12	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> com <i>FPU</i> e otimização O3 no sistema embebido	106
E.13	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> com memória cache e otimização O3 no sistema embebido.	107
E.7	Tabela de <i>Profiling</i> para dados do tipo <i>Double</i> e otimização O0 no sistema embebido.	108
E.8	Tabela de <i>Profiling</i> para dados do tipo <i>Double</i> e otimização O2 no sistema embebido.	109
E.14	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> com otimização O3 e algoritmo otimizado no sistema embebido.	110
E.15	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> com <i>FPU</i> , otimização O3 e algoritmo otimizado no sistema embebido.	111
E.16	Tabela de <i>Profiling</i> para dados do tipo <i>Float</i> com memória cache, otimização O3 e algoritmo otimizado no sistema embebido.	112
E.17	Tabela comparativa de tempos de <i>profiling</i> e de tempos calculados por contagem de ciclos de relógio para várias configurações e otimização O3.	112
E.18	Tabela de valores de tempo calculados por contagem de ciclos de relógio em função da dimensão da grelha de ocupação.	113

- E.19 Tabela de valores de tempo calculado por contagem de ciclos de relógio em função da distância a um obstáculo (só compreende a função de actualização da grelha). 113

Abreviaturas e Símbolos

E/S	Entrada/Saída
PWM	Pulse Width Modulation
SONAR	SOund Navigation and Ranging
RF	Rádio-Frequência
GPS	Global Positioning System
ASIC	Application Specific Integrated Circuits
PCB	Printed Circuit Board
UC	Unidade de Controlo
IO	Interface de Operador
LED	Light Emiting Diode
SPI	Serial Peripheral Interface
UART	Universal Assynchronous Receiver/Transmitter
GPIO	General Purpose Input/Output
INTC	Interrupt Controler
ALU	Arithmetic and Logic Unit
LMB	Local Memory BUS
FILO	First In Last Out
PC	Personal Computer
FPU	Floating Point Unit

Capítulo 1

Introdução

O desenvolvimento de sistemas embebidos é normalmente uma tarefa difícil. As duas soluções mais comuns passam pela utilização de *Application Specific Integrated Circuits* (ASIC) ou software implementado num processador programável [8].

Apesar da primeira opção (ASIC) fornecer soluções de elevado desempenho e integração, acarreta também um elevado custo de fabrico. A segunda opção, apesar de ser mais económica pode não permitir a satisfação de requisitos.

Actualmente, os sistemas embebidos exigem elevada integração e funcionalidade, baixo consumo energético e dimensão reduzida, para mencionar alguns aspectos importantes. Neste contexto, *Field Programmable Gate Array* (FPGA) surgem com uma solução intermédia que providencia a performance fornecida por sistemas de hardware fixo e a flexibilidade da computação reconfigurável [9, 8].

O potencial de sistemas embebidos implementados em FPGAs é reconhecido e assinalável [10]. Existem projectos de investigação que concluíram que a implementação em módulos de hardware dedicado em FPGAs de funções de software críticas reduzia o tempo de execução e reduzir o consumo de energia [11]. Uma boa partição de software e hardware em sistemas embebidos implementados em FPGAs melhora significativamente a performance do sistema [12, 13, 8].

1.1 Contexto

As plataformas baseadas em FPGAs têm sido estudadas há já alguns anos com o intuito de serem utilizadas como base de sistemas embebidos. Estas têm obtido bons resultados, pois combinam funções de software e módulos de hardware para cumprir o objectivo desejado [14, 15].

O campo da robótica móvel aparece como um domínio apropriado para avaliar os sistemas embebidos como sistemas capazes de suportar e executar tarefas complexas de computação. Por exemplo, a tarefa de navegação compreende algoritmos computacionalmente muito exigentes e portanto adequados à implementação num sistema embebido para testar a sua performance [16].

Desenvolver sistemas embebidos baseados em FPGAs com aplicação à robótica, especificamente protótipos de robôs móveis, é um tema com grandes possibilidades de exploração.

É neste contexto que o trabalho desenvolvido neste projecto apresenta uma plataforma que integra um kit robótico com uma placa de FPGAs.

1.2 Objectivos

Pretende-se com este projecto construir um protótipo de um robô móvel autónomo para implementar e estudar uma arquitectura de sistema embebido baseado em FPGAs.

Especificamente, os objectivos principais deste trabalho são:

- Fornecer um protótipo que permita comandar um robô através de algoritmos implementados em FPGAs;
- Implementar um algoritmo de mapeamento considerado representativo na área da robótica móvel;
- Avaliar o tempo de execução do algoritmo de mapeamento utilizando soluções baseadas num processador RISC embebido no FPGA.

1.3 Estrutura da Dissertação

Ao longo deste documento abordamos um conjunto variado de tópicos. Iniciaremos com a definição dos requisitos do protótipo do robô. Em seguida, descrevemos a sua implementação e consequentes resultados experimentais. Finalizamos com as conclusões retiradas ao longo do nosso trabalho, sem esquecer as possibilidades de trabalho futuro.

Para auxílio na leitura deste documento apresentamos de seguida a sua estrutura.

- No **Capítulo 2** são descritas a teoria e aplicação prática dos algoritmos de robótica que irão pôr à prova as arquitecturas testadas no protótipo;
- O **Capítulo 3** aborda o desenvolvimento de um protótipo funcional de um robô móvel. São definidos requisitos funcionais e não funcionais. Escolheremos entre um conjunto de candidatos, o melhor kit robótico.
- O **Capítulo 4** apresenta as tecnologias escolhidas, assim como todos os componentes envolvidos na realização do robô móvel. É dada a conhecer uma visão de alto nível da implementação;
- No **Capítulo 5** aprofundamos a implementação descrevendo-a com maior pormenor e apresentamos o protótipo final;

- O **Capítulo 6** contém os resultados das experiências realizadas nas arquitecturas propostas com optimizações específicas. Por fim, apresentamos as conclusões retiradas destas experiências.
- Finalmente, no **capítulo 7** apresentamos as conclusões retiradas ao longo do nosso trabalho. Terminamos por fazer uma análise da prossecução dos objectivos determinados previamente e sugerimos ainda vias de acção com vista a melhorar alguns aspectos do projecto em trabalhos futuros.

Capítulo 2

Mapeamento na Robótica

Este capítulo apresenta o tema de mapeamento que serve de base aos algoritmos que irão testar o nosso protótipo. Começamos por realizar uma pequena introdução ao tema de mapeamento na robótica, onde abordamos os tópicos chave deste capítulo. Em seguida definimos e caracterizamos a noção de mapeamento, assim como apresentamos as várias perspectivas existentes acerca do tema. Posteriormente referimos alguns desafios inerentes a este tema e algumas abordagens propostas para a sua resolução. Enquadramos ainda o uso de probabilidades neste campo.

Realizamos também uma apresentação da técnica de grelhas de ocupação para a representação do ambiente, definida como preferencial neste projecto. Associado às grelhas de ocupação discutimos um modelo de incerteza para a actualização de probabilidades da grelha: o modelo do SONAR.

Como métodos de fusão de leituras de sensores, destacamos o método de Bayes e Dempster-Shafer.

Terminamos por sugerir um algoritmo de aplicação à grelha de ocupação.

2.1 Introdução

A investigação na área da problemática do mapeamento em robótica possui uma longa história, pois tem vindo a ser objecto de estudo ao longo de quase três décadas. A problemática do mapeamento é frequentemente designada na especialidade pela questão geradora *Where have I been?* e é vista como de extrema importância para o desenvolvimento de robôs móveis plenamente autónomos.

Os algoritmos de robótica aplicados ao mapeamento actualmente utilizados são predominantemente probabilísticos, devido à incerteza e ao ruído dos sensores usados. Estes são bastante eficazes em ambientes estáticos e de tamanho limitado.

Alguns algoritmos requerem informação exacta sobre a sua posição para construir o mapa ou então usam medidas odométricas. Outros algoritmos são equipados para corresponder dados em

diferentes instantes de tempo, outros ainda requerem símbolos físicos para identificação. Estes algoritmos podem ser executados em tempo real ou realizar várias operações sobre os dados.

Diversos paradigmas estão presentes na resolução da problemática do mapeamento, desde a geométrica *versus* topológica ou a *world-centric versus robot-centric*, que sugerem abordagens diferentes [16] que serão objectivamente resumidas posteriormente neste capítulo .

Uma forma comum e fácil para representação do ambiente é possível através de uma grelha de ocupação. Esta técnica é utilizada para integrar dados de vários sensores num modelo do ambiente.

A esta grelha é sobreposto um modelo de incerteza de sensor para a actualização dos valores de probabilidades. Estes valores são melhor traduzidos com recurso a métodos probabilísticos de fusão de dados de vários sensores numa só grelha de ocupação, como sejam o de Bayes e o de Dempster-Shafer [17].

2.2 Características e Perspectivas no Mapeamento

Segundo S. Thrun, a problemática de mapeamento pode-se definir como o processo de construir um modelo espacial do ambiente de um robô, através da percepção do mundo por via de sensores [16]. Entre os sensores utilizados para este fim encontramos as câmaras, sonares, lasers, tecnologia de infravermelhos e sensores odométricos, como codificadores de impulsos, bússolas e GPS [18, 16].

Todos os sensores que possam ser objecto de uso estão sujeitos a erros ou ruído de medida e são limitados na sua aplicação. Estas limitações exigem que o robô navegue pelo ambiente enquanto constrói o mapa. A sua deslocação está também sujeita a erros, pelo que a postura do robô (localização e orientação) relativamente ao ambiente não será determinada apenas pela odometria.

O mapeamento pode ser abordado através de duas perspectivas: a métrica e a topológica. A perspectiva métrica, capta as propriedades geométricas do ambiente, enquanto a topológica descreve a conectividade dos diferentes locais.

Um dos contributos importantes na perspectiva métrica pertenceu a Elfes e Moravec [18, 16]. Este conceito de mapeamento através de grelhas de ocupação representa o mundo através de células com precisão variável que modelam o espaço livre e ocupado do ambiente.

A perspectiva topológica descreve os ambientes como uma lista de locais que são conectados por arcos. Estes arcos contêm informação de como navegar de um local para o outro. No entanto, esta distinção é algo difusa, visto que ambas as opções dependem de informação geométrica. Como tal, os algoritmos métricos têm uma maior precisão que os topológicos.

Este projecto será desenvolvido em torno da perspectiva métrica, através da utilização de uma grelha de ocupação do ambiente. Este método tem um custo computacional que será abordado na fase experimental.

Além das formas referidas atrás, uma outra forma de classificar a problemática do mapeamento define-se por *world-centric* e *robot-centric*.

Actualmente a perspectiva dominante é a de *world-centric*. Esta caracteriza-se por ser abstracta, sendo o espaço representado por coordenadas globais.

A perspectiva *robot-centric* é mais simples, pois não exige transformação. Isto é, são descritos em medidas do espaço, tendo cada medida, informação que o robô receberia em cada local. Esta perspectiva é preterida, pois em *world-centric* a extrapolação de objectos próximos é relativamente simples e imediata. Na perspectiva de *robot-centric* não existem dados geométricos no espaço de medidas que permita esta correlação[18, 16].

2.3 Desafios no Mapeamento na Robótica

O processo de mapeamento encontra-se rodeado de desafios e complicações. O ruído de medição levanta um desafio ao processo de mapeamento. Este problema seria facilmente solúvel, caso o ruído das diferentes medições fosse estatisticamente independente, o que não é o caso, pois o erro acumula-se ao longo do tempo afectando medidas futuras. Integrar estes erros é determinante para realizar o mapeamento do ambiente, sendo também um factor que complica o processo de mapeamento.

A elevada dimensionalidade do ambiente também causa um constrangimento ao processo de mapeamento. A dimensionalidade define-se como o número de variáveis que são necessárias para descrever o ambiente. Cada variável conta como uma dimensão.

O problema de correspondência é definido pela incerteza de saber se várias medidas dos sensores amostradas em diferentes alturas no tempo correspondem ao mesmo objecto no ambiente [18, 16].

Os ambientes raramente são estáticos, pois sofrem constantemente alterações. Este dinamismo cria um grande desafio ao processo, visto que acrescenta uma nova forma de explicar leituras aparentemente inconsistentes. Virtualmente, não há qualquer algoritmo para mapear eficazmente ambientes dinâmicos. De facto, o paradigma predominante continua a ser o ambiente estático em que o robô é o único elemento variante no tempo[16].

A problemática de mapeamento está intimamente relacionada com a de localização, ou seja, a determinação da postura do robô. De forma intuitiva pode-se considerar que para podermos mapear o ambiente temos de saber a posição do robô.

Em essência, ambas as problemáticas contêm incertezas e se focarmos apenas uma, a outra afectará o processo, como se de ruído sistemático se tratasse. Abordar ambas ao mesmo tempo resulta no facto de tanto o ruído de medida como de controlo se tornarem independentes no que toca às propriedades que estão a ser calculadas [18, 16].

2.4 Probabilidade na Robótica

Como mencionado na Introdução, os algoritmos robóticos de mapeamento são predominantemente probabilísticos, visto serem caracterizados pela incerteza e ruído. Estes algoritmos modelam explicitamente o problema do ruído e os seus efeitos nas medidas efectuadas.

O ruído nas medições é complexo e difícil de englobar num modelo. Esta dificuldade levou a comunidade a adoptar uma única metodologia matemática. O princípio subjacente a esta metodologia de mapeamento é o Teorema de Bayes, o arquétipo da inferência probabilística [16].

Seja $H_1, H_2, \dots, H_k$ uma partição do espaço amostral e s um evento associado a esse mesmo espaço amostral. Então, aplicando a probabilidade condicionada, o Teorema de Bayes é expresso pela equação (2.1):

$$P(H_i|s) = \frac{P(s|H_i)P(H_i)}{\sum_{j=1}^k P(s|H_j)P(H_j)} \quad i = 1, 2, \dots, k \quad (2.1)$$

A equação (2.1) goza da propriedade de probabilidades $P(H|s) + P(\bar{H}|s) = 1.0$.

Suponha-se que se quer descobrir se um dado evento H (**Hipótese**) ocorreu baseado numa medida s de um sensor (e.g., odometria, sensores de distância). A equação (2.1) pode ser simplificada na equação (2.2):

$$P(H|s) = \eta P(s|H)P(H) \quad (2.2)$$

Pela equação anterior [16], o problema pode ser resolvido com base na multiplicação de $P(s|H) \times P(H)$.

O termo $P(s|H)$ indica a probabilidade de uma dada medida s ter ocorrido dependendo da hipótese H e descreve o processo de gerar medidas diferentes baseadas em hipóteses H diferentes. O termo $P(H)$ fornece informação *a priori* e não leva em consideração nenhum dado de leitura s .

Como uma probabilidade é determinada pelo historial completo de movimentação do robô, todas as leituras devem ser consideradas no cálculo da nova probabilidade.

No campo do mapeamento robótico o esquema dominante para integrar dados é conhecido por *Filtro de Bayes* [18, 16]. Estes filtros são recursivos e resultam de uma extensão da regra de Bayes a problemas de estimação que decorrem ao longo do tempo.

Os filtros de Bayes são a abordagem mais simples de inferência probabilística a partir de dados obtidos ao longo do tempo [18, 16].

2.5 Grelhas de Ocupação para o Mapeamento

Para realizar o mapeamento é frequentemente necessário definir uma forma de representação do ambiente. Uma técnica comum desenvolvida por Elfes e Moravec nos anos 80 caracteriza-se pelo uso de grelhas de ocupação com a postura do robô conhecida (localização e orientação) [17, 16].

Uma grelha de ocupação representa o ambiente com um conjunto de células que podem estar ocupadas ou livres e é uma técnica utilizada para integrar dados de vários sensores num modelo do ambiente. A integração é realizada com recurso a algoritmos probabilísticos baseados numa teoria formal de evidência [17].

Os mapas que recorrem à grelha de ocupação não são totalmente precisos, sendo difícil gerar um mapa devido a dados incompletos ou com ruídos dos sensores.

Existe muita ambiguidade nos dados recolhidos pelos sensores. As aplicações mais comuns desta técnica utilizam sensores de detecção de obstáculos e distâncias, tal como SONAR ou *laser range finders*.

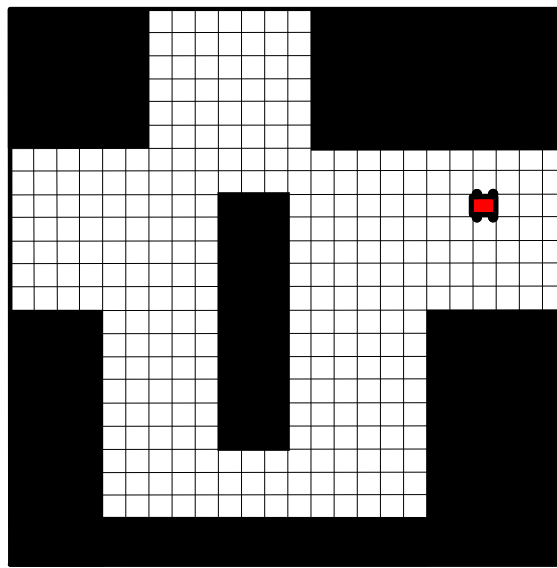


Figura 2.1: Exemplo de grelha de ocupação.

O SONAR cobre um cone inteiro no espaço, em que um obstáculo se pode encontrar algures na abrangência do cone. Além de ser caracterizado pelo ruído, o sonar sofre os efeitos reflexivos dos materiais dos obstáculos e dos ângulos em que estes se encontram.

Um qualquer sensor transmite informação sobre um conjunto de células que podem ser referenciadas sem ser necessário aceder às restantes. Uma leitura é comparada com o mapa, alterando a probabilidade das células visadas. Cada célula é definida pela probabilidade de poder ser atravessada.

Os mapas gerados são probabilísticos derivando na sua versão mais simples, de filtros de Bayes. Esta é uma técnica robusta e fácil de implementar. No entanto, peca por não ter explicitamente um método que aborde a incerteza da pose do robô.

Outra deficiência da técnica de grelha de ocupação é herdada pelo filtro de Bayes que assume ruído independente, isto é evidente quando os valores são integrados enquanto o robô se encontra estacionário. Finalmente, a assumpção de independência entre células contribui como último aspecto negativo a referir desta técnica [16].

2.6 Modelo de Sensor de SONAR

Como já referido, os sonares são sensores comuns na construção de mapas de ambiente em grelhas de ocupação. Este sensor foi objecto de intenso estudo, existindo um modelo de incerteza amplamente utilizado [17].

O modelo mais simples projecta um cone no espaço, caracterizado por um ângulo de vista de β que representa o meio ângulo do cone, e R que define o máximo alcance que este pode detectar. Este cone é projectado numa grelha de ocupação e é dividido em três regiões (ver Figura 2.2):

- **Região I:** Onde as células afectadas se encontram provavelmente ocupadas;
- **Região II:** Onde as células afectadas estão provavelmente vazias;
- **Região III e IV:** Onde a condição das células é desconhecida.

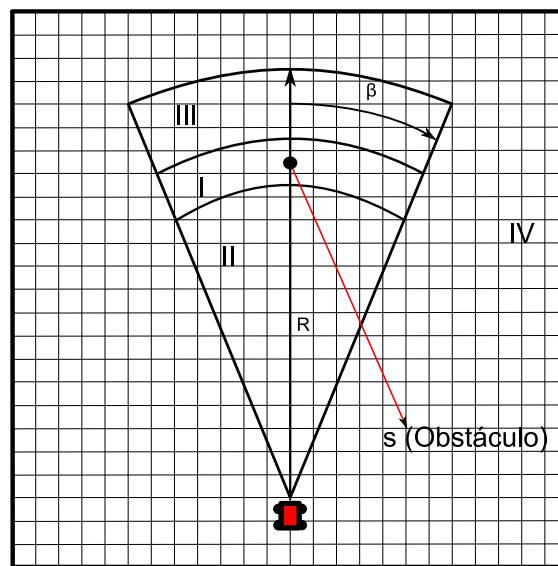


Figura 2.2: Modelo de sensor para o SONAR.

O facto da Região II se estender por várias posições define a sua precisão e reflecte explicitamente a incerteza na leitura. Como tal estabelece-se uma *tolerância*, para fazer face a fenómenos como a refacção e a dispersão.

Da Figura 2.2 pode-se inferir que a probabilidade de uma leitura estar correcta aumentará quando $\beta \rightarrow 0$ e quando $s \rightarrow \text{posição do robô}$.

Pode-se utilizar um método qualquer de natureza probabilística ou semi-evidencial para traduzir as suas leituras em valores [17].

2.7 Métodos na Fusão de Leituras de Sensores

O método ou algoritmo para fusão de dados dos vários sensores nas células de uma grelha de ocupação é uma escolha livre. Como afirmamos anteriormente, o sonar é dos sensores mais utilizados para a tarefa de mapeamento ao obter pequenas representações do ambiente à medida que o robô se desloca.

De seguida apresentamos implicações práticas de duas teorias de evidência na utilização do sonar numa grelha de ocupação: a Bayesiana e a de Dempster-Shafer.

2.7.1 Bayes

O método Bayesiano é dos mais populares na tradução de dados dos sensores em probabilidade e para combinar probabilidades [17].

O resultado do funcionamento de um sonar consiste na verificação da existência de obstáculos no seu trajecto. Este evento, definido por H , pode ser uma de duas hipóteses: **Ocupado** ou **Vazio**. O espaço amostral é então definido por $\Theta = \{Ocupado, Vazio\}$.

Seja $P(H)$, a probabilidade de H ter ocorrido. A este estão associados duas importantes propriedades de probabilidades. A primeira propriedade representa-se por:

$$0 \leq P(H) \leq 1$$

A segunda indica que se conhecermos $P(H)$, então é possível calcularmos $P(\bar{H})$,

$$P(\bar{H}) = 1 - P(H)$$

Neste método, o modelo do sensor gera probabilidades condicionais na forma de $P(s|H)$, sendo posteriormente convertidas em $P(H|s)$ pela regra de Bayes (2.1). O resultado prático da utilização desta teoria permite que duas leituras de sensores no mesmo instante de tempo ou de tempos diferentes sejam combinadas facilmente.

A cada posição numa grelha de ocupação, está associada as probabilidades $P(Ocupado|s)$ e $P(Vazio|s)$.

Segundo R. Murphy [17], para representar a incerteza numa posição da grelha deve-se começar por implementar uma estrutura para guardar as probabilidades associadas, apresentado em baixo.

```
struct grid
    double ocupado
    double vazio
```

```
grid[ROWS][COLUMNS]
```

A mesma autora fornece ainda um conjunto de funções que quantifica uma medida do SONAR em probabilidades para cada posição na grelha.

Relembramos que o modelo de sensor representa a probabilidade $P(s|H)$: a probabilidade do sensor retornar um valor, dado a hipótese de a posição da grelha estar ocupada/livre. Este conjunto de funções apresenta-se de seguida:

Região I

$$P(s|Ocupado) = \frac{(\frac{R-r}{R}) + (\frac{\beta-\alpha}{\beta})}{2} \times Max_{ocupação} \quad (2.3a)$$

$$P(s|Vazio) = 1.0 - P(s|Ocupado) \quad (2.3b)$$

r distância da origem à posição na grelha observada;

α ângulo até à posição da grelha observada;

$Max_{ocupação}$ expressa

Região II

$$P(s|Vazio) = \frac{(\frac{R-r}{R}) + (\frac{\beta-\alpha}{\beta})}{2} \quad (2.4a)$$

$$P(s|Ocupado) = 1.0 - P(s|Vazio) \quad (2.4b)$$

Região III e Região IV Não existem alterações.

Como a probabilidade que pretendemos é $P(H|s)$, utilizamos o Teorema de Bayes para a relacionar $P(s|H)$. Substituindo *Ocupado* por H , temos a seguinte equação:

$$P(Ocupado|s) = \frac{P(s|Ocupado)P(Ocupado)}{P(s|Ocupado)P(Ocupado) + P(s|Vazio)P(Vazio)} \quad (2.5)$$

Os termos $P(Ocupado)$ e $P(Vazio)$ são probabilidades conhecidas *a priori*. Se houver desconhecimento desta informação considera-se que $P(Ocupado) = P(Vazio) = 0.5$. É com estes valores que deve ser inicializada uma grelha de ocupação [17].

Após calcularmos as probabilidades condicionais, é necessário integrá-las com leituras de outros sensores e leituras efectuadas em instantes anteriores.

O filtro de Bayes genérico pode ser utilizado iterativamente para calcular a nova probabilidade em t_n , onde a probabilidade anterior t_{n-1} se torna a probabilidade *a priori*. No entanto, a sua forma recursiva considera o facto dos resultados de leituras pertencerem a diferentes experiências, tornando o cálculo da nova probabilidade muito mais simples.

A forma de actualização de probabilidades associadas a uma posição expressa-se então pela equação (2.6) [17]:

$$P(H|s_n) = \frac{P(s_n|H)P(H|s_{n-1})}{P(s_n|H)P(H|s_{n-1}) + P(s_n|\bar{H})P(\bar{H}|s_{n-1})} \quad (2.6)$$

A regra apresentada na equação anterior é comutativa, tornando-se indiferente a ordem em que duas leituras são processadas.

2.7.2 Dempster-Shafer

Originalmente criada por A. P. Dempster, um Matemático da Universidade de Harvard nos anos 60, e desenvolvida nos anos 80 por Glen Shafer, a teoria de Dempster-Shafer é uma alternativa mais recente do que a de Bayes [17].

Enquanto a regra de Bayes se baseia no facto da evidência ser representada por funções probabilísticas, a teoria de Dempster-Shafer representa a evidência como uma função possibilista. O facto de ser possibilista significa que a função representa uma evidência parcial. Isto é, a possibilidade da evidência pode ser mais significativa do que aquela que foi reportada.

A Regra de Combinação de Dempster é muito diferente da de Bayes, mas com resultados semelhantes [17]. Ao contrário da Teoria de Bayes, na Teoria de Dempster podemos saber onde múltiplas observações entram em conflito que pode ser usado para detectar quando uma grelha de ocupação está sujeita a erros [17].

A função de possibilidade é articulada com a Regra de Combinação de Dempster.

Funções de Possibilidade de Shafer

As funções de possibilidade de Shafer servem o mesmo objectivo que as probabilidades para o método de Bayes. Em vez de medir a probabilidade da proposição, a função de possibilidade mede o denominado grau de confiança (do inglês *mass belief*) m . Cada sensor contribui com um $m = 1.0$, que pode ser distribuído por várias proposições.

Segundo R. Murphy, o resultado deste raciocínio leva a que cada posição de uma grelha de ocupação seja caracterizada por um tuplo com três membros [17]. Esta função de possibilidade pode ser escrita como:

$$Bel = m(Ocupado), m(Vazio), m(Incerteza)$$

Portanto, a estrutura representativa da grelha de ocupação deve ser representada da forma que se segue, para implementação em computação.

```
struct grid
    double ocupado
    double vazio
    double incerteza

grid[ROWS][COLUMNS]
```

Uma propriedade destas funções é que todo o m pode ser atribuído à *incerteza*, isto é, $m(incerteza) = 1.0$. Este valor é utilizado para inicializar uma grelha de ocupação.

As equações que quantificam uma medida do SONAR numa função de possibilidade para cada posição na grelha são:

Região I:

$$m(Ocupado) = \frac{(\frac{R-r}{R}) + (\frac{\beta-\alpha}{\beta})}{2} \times Max_{ocupação} \quad (2.7a)$$

$$m(Vazio) = 0.0 \quad (2.7b)$$

$$m(Incerteza) = 1.00 - m(Ocupado) \quad (2.7c)$$

r distância da origem à posição na grelha observada;

α ângulo até à posição da grelha observada;

Max_{ocupao} expressa

Região II:

$$m(Ocupado) = 0.0 \quad (2.8a)$$

$$m(Vazio) = \frac{(\frac{R-r}{R}) + (\frac{\beta-\alpha}{\beta})}{2} \quad (2.8b)$$

$$m(Incerteza) = 1.0 - m(Vazio) \quad (2.8c)$$

Região III e Região IV: Não existem alterações.

Uma diferença face ao método de Bayes é que qualquer incerteza na leitura conta como m para a *incerteza*.

Regra de Combinação de Dempster

O método mais popular e notavelmente denso para combinar funções de possibilidade é a *Regra de Combinação de Dempster* ou *Soma ortogonal* [17].

A Regra de Dempster combina duas funções *Bel1* e *Bel2* como se eles representassem uma intersecção física, desde que sejam independentes entre si. Significa então, que pode ser aplicada a leituras de sonares realizadas em alturas diferentes.

As duas funções *Bel1* e *Bel2* representam eixos ortogonais formando um quadrado de área 1.0. O interior do quadrado pode ser dividido em sub-regiões representando a confiança no elemento produzido pela intersecção dos dois eixos [17].

O conjunto das intersecções são demonstradas na Figura 2.3. Nesta Figura são visíveis regiões onde acontece a intersecção entre *Ocupado* e *Vazio*. Como estas são proposições mutuamente exclusivas, o seu resultado é o conjunto vazio $\emptyset$. Este factor causa um problema, visto que segundo a definição de uma função de confiança, nenhuma quantidade de confiança pode ser atribuída ao conjunto vazio $\emptyset$.

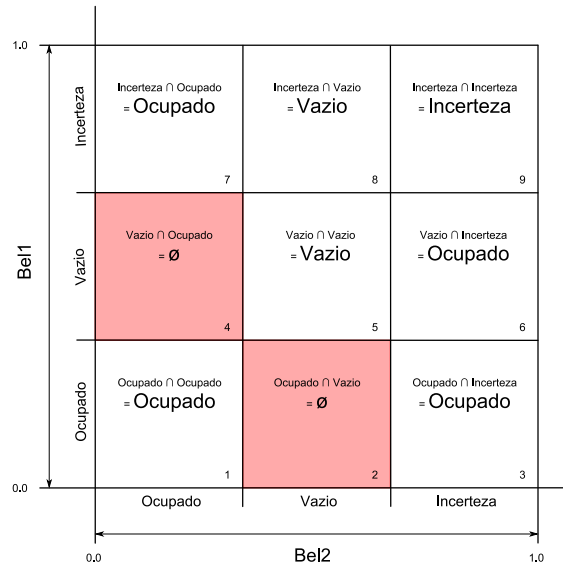


Figura 2.3: Um exemplo de descrição gráfica da Regra de Combinação de Dempster.

A Regra de Combinação de Dempster *normaliza* a função de confiança, distribuindo a área do conjunto vazio $\emptyset$ para as áreas não vazias.

A área de confiança criada pelas intersecções de duas funções de confiança forma uma terceira função de confiança *Bel3*, visível na Figura 2.4.

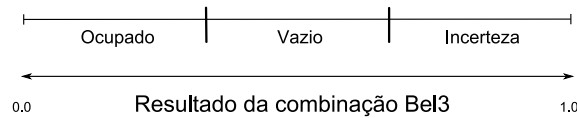


Figura 2.4: Resultado da Combinação de Dempster.

Murphy resume o trabalho de Dempster, apresentando para o caso de uma grelha de ocupação as expressões associadas à Regra:

$$m(\text{Ocupado}) = \frac{\sum_{A_i \cap B_j = \text{Ocupado}} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m(A_i)m(B_j)} \quad (2.9a)$$

$$m(\text{Vazio}) = \frac{\sum_{A_i \cap B_j = \text{Vazio}} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m(A_i)m(B_j)} \quad (2.9b)$$

$$m(\text{Incerteza}) = \frac{\sum_{A_i \cap B_j = \text{Incerteza}} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m(A_i)m(B_j)} \quad (2.9c)$$

Em termos práticos aplicáveis à computação, as equações (2.9) podem traduzir-se nas seguintes:

$$m(Ocupado) = \frac{(Ocupado_{t-1} \cap Ocupado) + (Incerteza_{t-1} \cap Ocupado) + (Ocupado_{t-1} \cap Incerteza)}{1 - [(Vazio_{t-1} \cap Ocupado) + (Ocupado_{t-1} \cap Vazio)]} \quad (2.10a)$$

$$m(Vazio) = \frac{(Vazio_{t-1} \cap Vazio) + (Incerteza_{t-1} \cap Vazio) + (Vazio_{t-1} \cap Incerteza)}{1 - [(Vazio_{t-1} \cap Ocupado) + (Ocupado_{t-1} \cap Vazio)]} \quad (2.10b)$$

$$m(Incerteza) = \frac{(Incerteza_{t-1} \cap Incerteza)}{1 - [(Vazio_{t-1} \cap Ocupado) + (Ocupado_{t-1} \cap Vazio)]} \quad (2.10c)$$

2.7.3 Aplicação do Algoritmo

Como forma de aplicação das teorias anteriores, apresentamos no Algoritmo 2.1, o pseudo-código de uma sugestão de implementação da actualização de leituras dos sensores.

Algoritmo 2.1 Algoritmo de Utilização de uma Grelha de Ocupação.

```

1: initialize occupancy grid
2: procedure UPDATE
3:   for all positions in the grid do
4:     compute distance to origin
5:     compute angle  $\alpha$ 
6:     if position  $\in$  Region I then
7:       compute new probabilities: eq. (2.3) for Bayes or eq. (2.7) for Dempster
8:       update probability for the position: eq. (2.6) for Bayes or eq. (2.9) for Dempster
9:     else if position  $\in$  Region II then
10:      compute new probabilities: eq. (2.3) for Bayes or eq. (2.8) for Dempster
11:      update probability for the position: eq. (2.6) for Bayes or eq. (2.9) for Dempster
12:     else if position  $\in$  ( Region III  $\vee$  Region IV ) then
13:       probabilities remain unaltered
14:     end if
15:   end for
16: end procedure

```

O Algoritmo 2.1 mostra que antes de se poder utilizar a grelha de ocupação para combinar leituras, é necessário inicializá-la de acordo com o método a utilizar. Para o método de Bayes usa-se para cada posição as seguintes probabilidades *a priori*:

$$P(Ocupado) = 0.5$$

$$P(Vazio) = 0.5$$

e para o método de Dempster-Shafer:

$$m(Ocupado) = 0.0$$

$$m(Vazio) = 0.0$$

$$m(Incerteza) = 1.0$$

Após a inicialização da grelha, calcula-se, para cada posição da grelha, as probabilidades associadas a uma nova leitura. As posições que estiverem dentro do cone definido no modelo do sonar actualizarão o novo valor de probabilidade com o anterior da mesma posição.

Para melhor se entender este processo recorremos a um exemplo de duas leituras. Para começar definimos os parâmetros do modelo do sonar:

$$\text{Dimensão da grelha} = 24 \times 24 \text{ posições}$$

$$\text{Alcance do SONAR} = s = 10 \text{ unidades}$$

$$\beta = 15^\circ$$

$$Max_{Ocupação} = 0.98$$

$$\text{Tolerância} = \pm 1 \text{ unidade}$$

Consideremos um primeiro exemplo em que o SONAR retorna uma leitura $r = 10$ e um ângulo $\alpha = 0^\circ$

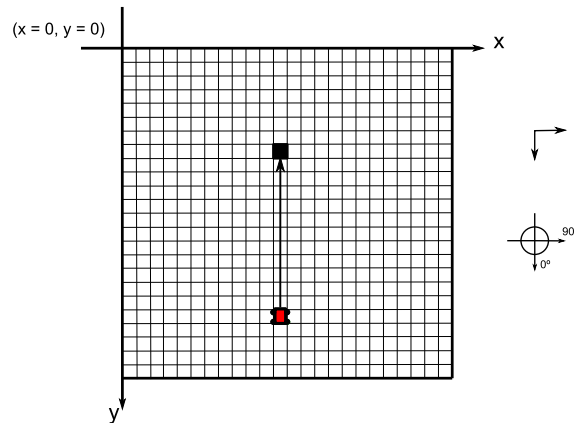


Figura 2.5: Grelha com exemplo de leitura.

Ambos os métodos têm de satisfazer duas condições:

- $|\alpha| \leq |\beta|$, o elemento encontra-se dentro do ângulo de visão do modelo do SONAR;
- $r \leq s + \text{tolerância}$, o elemento encontra-se dentro do limite superior do alcance da medida;

Verifica-se então que $s - \text{tolerância} = 9 \leq r = 10 \leq s + \text{tolerância} = 11$, encontra-se na Região I.

Aplicando ambos os métodos obtemos:

Bayes

$$\begin{aligned} P(s|Ocupado) &= \frac{\left(\frac{10-10}{10}\right) + \left(\frac{15^\circ-0^\circ}{15^\circ}\right)}{2} \times 0.98 = 0.49 \\ P(s|Vazio) &= 1.0 - P(s|Ocupado) = 0.51 \end{aligned}$$

Dempster-Shafer

$$\begin{aligned} m(Ocupado) &= \frac{\left(\frac{10-10}{10}\right) + \left(\frac{15^\circ-0^\circ}{15^\circ}\right)}{2} \times 0.98 = 0.49 \\ m(Vazio) &= 0.0 \\ m(Incertezas) &= 1.00 - m(Ocupado) = 0.51 \end{aligned}$$

Depois de calculadas as probabilidades associadas à nova leitura é altura de actualizar o valor na posição da grelha, ver Figura 2.6.

Bayes

$$P(Ocupado|s) = \frac{0.49 \times 0.50}{0.49 \times 0.50 + 0.51 \times 0.50} = 0.49$$

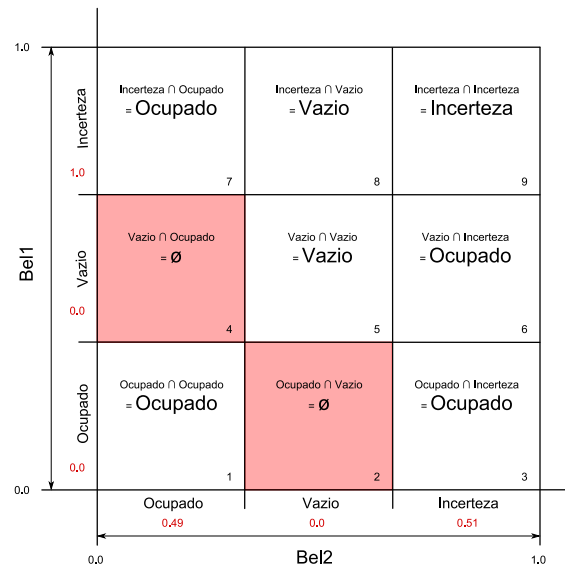
Dempster-Shafer

$$\begin{aligned} m(Ocupado) &= \frac{(0.0 \times 0.49) + (1.0 \times 0.49) + (0 \times 0.51)}{1 - [(0 \times 0.49) + (0 \times 0)]} = 0.49 \\ m(Vazio) &= \frac{(0 \times 0) + (1 \times 0) + (0 \times 0.51)}{1 - [(0 \times 0.49) + (0 \times 0)]} = 0 \\ m(Incerteza) &= \frac{(1 \times 0.51)}{1 - [(0 \times 0.49) + (0 \times 0)]} = 0.51 \end{aligned}$$

A função possibilista *Bel3* é então caracterizada por: $m(Ocupado) = 0.49$, $m(Vazio) = 0.0$ e $m(Incerteza) = 0.51$.

2.8 Resumo

Vimos que à problemática do mapeamento está associado um conjunto de desafios e perspectivas distintas que vão influenciar a abordagem ao problema. A tarefa de mapeamento é predominantemente pautada pelo ruído e incerteza das leituras dos sensores. As grelhas de ocupação apresentam-se como uma técnica adequada à resolução desta problemática, permitindo a representação do ambiente percorrido pelo robô. Como forma de actualizar as células constituintes da

Figura 2.6: Calculo de função possibilística $Bel3$.

grelha de ocupação, utilizam-se métodos probabilísticos que permitem combinar dados de vários sensores amostrados em instantes diferentes. É frequente a utilização de sonares no seguimento da técnica da grelha de ocupação. O modelo de incerteza associado sobrepõe-se, leitura a leitura, a esta grelha. Os métodos probabilísticos preferidos na tradução de leituras em probabilidades são baseados nos filtros de Bayes.

Uma aplicação prática de mapeamento utilizando as ferramentas abordadas realiza os seguintes passos:

- Percorre todas as células da matriz;
- Para cada célula, verifica em que zona do modelo do sensor esta se encontra;
- Actualiza a probabilidade dessa posição, se necessário.

As considerações tomadas neste capítulo constituirão a base do trabalho a realizar neste projecto.

Capítulo 3

Definição do Protótipo

Pretende-se com este projecto montar um protótipo para teste e avaliação de algoritmos de robótica móvel. A sua implementação será baseada em FPGAs. O protótipo consistirá em um ou mais placas de desenvolvimento de FPGA acopladas numa base móvel equipada com sensores adequados à satisfação dos objectivos.

Como forma de avaliar o rendimento desta solução, o protótipo será testado com um ou mais algoritmos de mapeamento. Estes terão que ser computacionalmente exigentes para tirar o máximo rendimento das características do protótipo. Os algoritmos escolhidos serão implementados em um microprocessador embebidos no FPGA.

3.1 Sistema

Após a definição dos objectivos deste projecto no Capítulo 1, é possível então delinear uma representação gráfica do sistema que irá ser desenvolvido. O protótipo será dividido em quatro subsistemas, que passamos a apresentar na Figura 3.1. Nesta figura demonstra-se uma visão de alto nível do protótipo. Nas subsecções seguintes descreve-se cada um dos subsistemas.

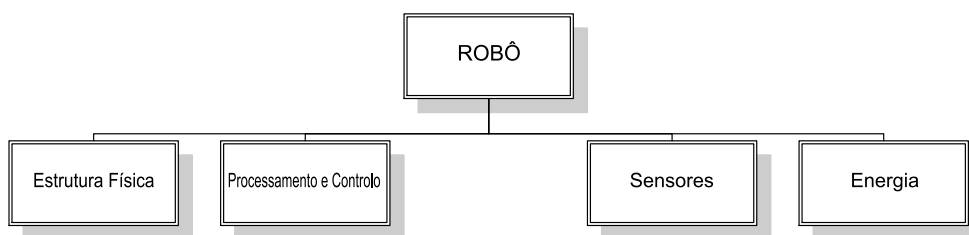


Figura 3.1: Diagrama de alto-nível do protótipo.

3.1.1 Estrutura Física

No subsistema estrutural considera-se a divisão entre a estrutura de suporte do robô e a propulsão. A propulsão contempla os motores e rodas como meio de deslocamento. Este subsistema é representado na Figura 3.2.

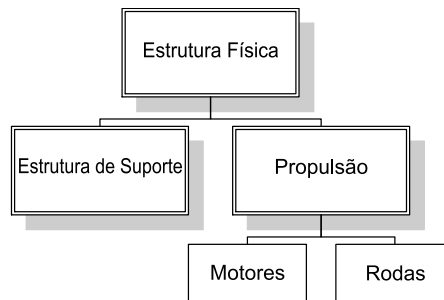


Figura 3.2: Subsistema estrutura.

3.1.2 Sensores

O subsistema dos sensores é um subsistema bastante flexível. Este subsistema é considerado flexível porque, sendo a unidade principal de processamento externa ao kit robótico, é-lhe possível conectar qualquer tipo de sensor ou equipamento adicional, desde que a placa de FPGAs possua portos de entrada/saída suficientes e/ou interfaces adequadas.

Consideramos como componentes mínimos para o cumprimento da missão, os sensores de ultra-sons e os codificadores de impulsos (Figura 3.3). Os primeiros têm como objectivo evitar obstáculos e os segundos servem para auxiliar a determinar a postura do robô (localização e direcção).

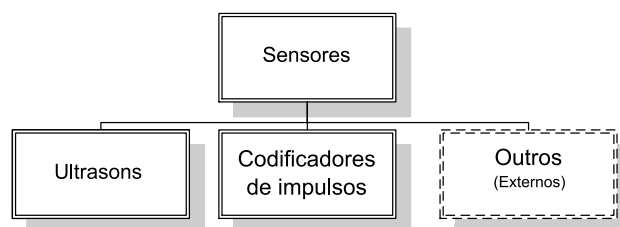


Figura 3.3: Subsistema sensores.

A inclusão de novos sensores ou equipamentos está sujeito à disponibilidade de área na estrutura de suporte, assim como da necessidade para a realização da função.

3.1.3 Energia

O kit robótico deve transportar uma fonte de alimentação. A alimentação energética do kit robótico deve ser suficiente para também fornecer energia aos componentes instalados (incluindo a placa de FPGAs). Em casos excepcionais, e caso seja considerado necessário, pode ser acrescentado uma pequena bateria para um ou mais módulos extra. A Figura 3.4 representa o subsistema energético estabelecendo as necessidades do sistema.

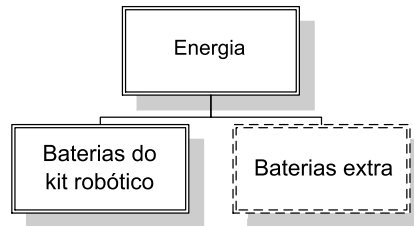


Figura 3.4: Subsistema energia.

3.1.4 Controlo e Processamento

Na unidade computacional existirão duas unidades de processamento (CPU): a CPU do kit robótico e a CPU embebida na placa de FPGA.

A CPU implementada na placa de FPGAs será a principal unidade de processamento controlando inclusive o sistema electrónico do kit robótico. Por outro lado, a CPU do kit terá a função de executar comandos enviados pela FPGA e enviar de volta os dados recebidos dos sensores acoplados. Estes dados são necessários ao processamento dos algoritmos de mapeamento.

Pode ser também necessário implementar alguma lógica adicional na placa de FPGA para responder a necessidades que não possam ser satisfeitas por elementos presentes nos módulos existentes, como seja, implementar protocolos de comunicação.

O subsistema é ilustrado na Figura 3.5 e é, porventura, o mais complexo.

3.2 Requisitos do Sistema

Seguindo as indicações que têm vindo a ser descritas, ilustra-se na Figura 3.6 o sistema alvo a desenvolver.

Para realizarmos com sucesso o esquema apresentado é necessário especificar os seus requisitos. Estes requisitos servirão de base para uma escolha entre kits robóticos já existentes no mercado. Deverão englobar, preferencialmente, uma boa parte do que é exigido do protótipo.

3.2.1 Requisitos Funcionais

É agora possível enumerar quais as funcionalidades que o protótipo deverá realizar. Essas funcionalidades deverão ser as seguintes:

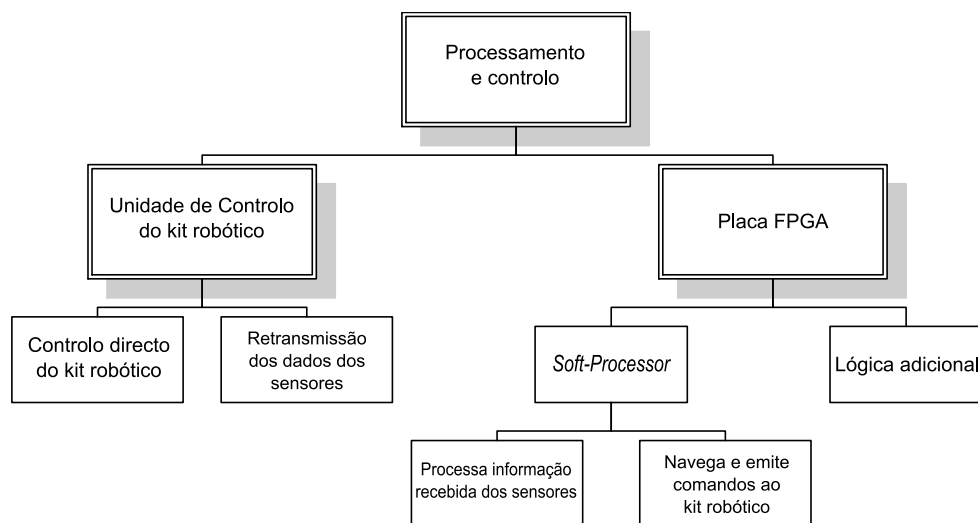


Figura 3.5: Subsistema de processamento e controlo.

1. Construir um mapa do ambiente de acordo com o percurso que vai percorrendo e os dados dos sensores que vai recebendo;
2. Percorrer um ambiente evitando obstáculos;
3. A CPU do FPGA deve comunicar com a CPU do kit para:
 - (a) Enviar comandos decorrentes do algoritmo de navegação;
 - i. Virar à esquerda/direita um número específico de graus;
 - ii. Andar para a frente;
 - (b) Controlar a velocidade instantânea do robô;
 - (c) Parar e arrancar;
 - (d) Receber dados decorrentes das leituras dos sensores directamente acoplados ao controlador do kit.
4. À CPU do FPGA devem poder ser conectados sensores e equipamentos extra que permitam melhorar a performance e implementar uma escolha variada de algoritmos de navegação;
5. A CPU do robô deve ser capaz de enviar leituras dos sensores para a CPU do FPGA e de receber comandos da CPU do FPGA.

3.2.2 Requisitos Não Funcionais

Requisitos de *Design*

1. O protótipo deve ter um peso e dimensões que permita ser transportado e manobrado com relativa facilidade por um operador;

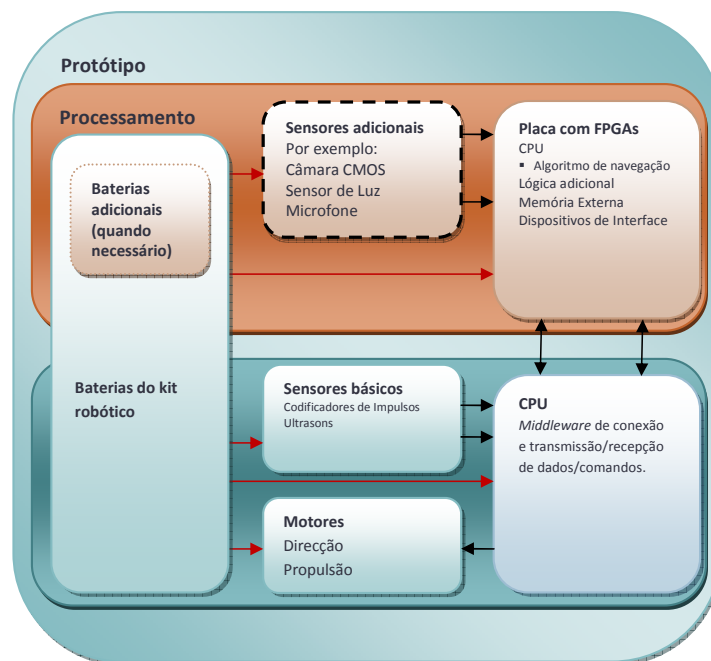


Figura 3.6: Organização do protótipo.

2. As dimensões devem ser tais que permita:
 - (a) Realizar uma inversão de sentido em corredores com 1,50 metros de largura;
 - (b) Alojara uma ou mais placas de FPGA e módulos extra (sensores ou outros).
3. Os materiais de construção da base devem ser robustos e leves (e.g., alumínio);
4. A estrutura de apoio deve permitir a inclusão e fixação de novos módulos de forma fácil e segura (e.g., inclusão de orifícios);
5. Tanto a CPU do kit como do FPGA devem ser reprogramáveis;
6. Tanto a placa da CPU do kit robótico como a placa FPGA devem ter portos de entrada/saída que devem poder ser conectados a partir de fios condutores;
7. Preço do kit de robótica menor ou igual a 1000 €.

Requisitos de Desempenho

1. Ter autonomia para operar durante, pelo menos, uma hora;
2. Deve realizar a sua missão a uma velocidade visivelmente satisfatória sem interrupções nem descontinuidades;
3. Realizar a sua missão no menor período de tempo possível.

3.3 Kits Robóticos Comerciais

Para ir de encontro aos requisitos estabelecidos foi necessário realizar um estudo de mercado, no sentido de encontrar kits de robótica que pudessem ser candidatos ao nosso projecto. Este estudo foi realizado e orientado no sentido de encontrar kits que contivessem já um bom suporte, seja de componentes incorporados como de informação.

Consideraram-se kits que fossem utilizados largamente por campos de investigação na área da robótica e utilizados em salas de aula de universidades de todo o mundo.

De notar que todas as empresas escolhidas estão sediadas nos Estados Unidos da América (EUA). De facto, a grande maioria de fabricantes destes kits robóticos e de robôs humanóides encontram-se localizados nos EUA, Coreia do Sul e Japão.

Na Europa existe um número relevante de fabricantes de produtos que estendem as capacidades dos kits robóticos.

Como resultado da pesquisa enumera-se nas subsecções seguintes os kits que foram considerados como candidatos a este projecto.

3.3.1 VEX Robotics

A VEX Robotics [1] é uma empresa que produz kits robóticos especificamente para fins educacionais. Comercializa ainda módulos extra especialmente adaptados aos seus kits robóticos.

Este kit tem no geral óptimas características. Possui a vantagem de ser bastante flexível, graças aos materiais utilizados mas também ao facto de ser constituído por peças soltas. Este kit possibilita uma grande modularidade pois permite diversas montagens em função da sua missão (ver tabela A.1). Por sua vez, este facto acarreta uma perda de robustez inerente aos diversos pontos de apoio presentes.

O preço encontra-se dentro do limite estabelecido.

De maneira geral cumpre os requisitos estabelecidos anteriormente.

Na Figura 3.7 podemos observar algumas montagens customizadas deste kit.



Figura 3.7: Montagens possíveis para o kit robótico da VEX Robotics [1].

3.3.2 Mekatronix

A Mekatronix [2] produz kits robóticos para investigação, que são usados em muitas universidades.

Dentro dos kits que comercializam seleccionamos para candidato o Talrik Jr. Pro TM.

Este kit tem uma estrutura de apoio cilíndrica, construída com materiais robustos. Além dos kits robóticos, produzem também uma variedade razoável de extras e upgrades que potenciam as capacidades deste kit.

O preço encontra-se muito próximo do estabelecido. No entanto, a sua forma e dimensões deixam muito a desejar, tanto no que toca à sua portabilidade como em termos de flexibilidade (ver tabela A.2).

Podemos contemplar o robô Talrik Jr. Pro TM na Figura 3.8, o segundo a contar da esquerda.

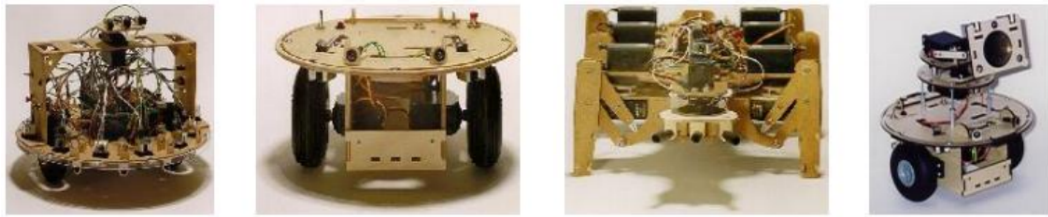


Figura 3.8: Kits robóticos disponibilizados pela Mekatronix [2].

3.3.3 Mobile Robots

A Mobile Robots [3] é uma empresa que comercializa soluções robóticas para variadas áreas de intervenção: investigação, desenvolvimento e empresarial/institucional.

A par das suas soluções *standard*, é possível escolher de uma gama variada, módulos que complementam os kits robóticos fornecidos. Para o efeito considerou-se como candidato o modelo Pioneer 3-AT TM.

O Pioneer 3-AT TM possui uma estrutura de apoio bastante robusta, contemplando módulos de controlo sofisticados e com um suporte consolidado, quer dos seus produtos quer em termos de documentação, facto que leva a ser utilizado nas áreas referidas. Apesar destes pontos positivos, a sua base é extremamente pesada e pouco flexível, não permitindo muitas alterações, caso os objectivos da missão sejam redefinidos (ver tabela A.3).

O preço deste kit robótico é muito superior ao valor limite.

O modelo Pioneer 3-AT TM é o primeiro da esquerda, na Figura 3.9.



Figura 3.9: Kits robóticos disponibilizados pela Mobile Robots [3].

3.3.4 Lego Mindstorm NXT

O último kit considerado como possível candidato é fabricado pela Lego [4] e designa-se por Lego Mindstorm NXT. Dentro deste kit a nossa opção recaiu no robô Tribot. Trata-se de um kit muito interessante pelas suas características sendo utilizado como um kit de iniciação à robótica.

Na análise comparativa destaca-se pelo seu baixo custo e por possuir uma unidade de controlo sofisticada de 32 bits com interfaces com usb e bluetooth, permitindo uma elevada flexibilidade de controlo e conectividade.

É um produto leve e bastante portátil, fácil de transportar e movimentar com relativa facilidade. No entanto, apresenta algumas vulnerabilidades. A estrutura de apoio e os materiais de construção são muito frágeis afectando a robustez do produto (ver tabela A.4).

Uma imagem do Tribot encontra-se presente na Figura 3.10.



Figura 3.10: Kit robótico da Lego [4].

3.4 Comparação

Definidos os candidatos a este projecto e com o apoio da Tabela A.1 elabora-se uma análise comparativa dos vários kits robóticos.

Pode-se observar através da tabela que em termos de preço, o kit da Lego seria a opção mais em conta. No entanto, as restantes opções também se encontram dentro do limite estabelecido como requisito.

Comparando o nível do controlo e sensorial todos se equiparam com ligeira desvantagem para o kit da Lego. Todos possuem microcontroladores reprogramáveis e com bastantes funcionalidades incorporadas e pinos de E/S permitindo a inclusão de módulos especificamente desenhados para os kits, como módulos de sensores extra. Para o kit da Lego, apesar da boa unidade de controlo, é necessária a inclusão de módulos de comunicação extra para contactos com FPGAs e sensores que não acompanhem o kit.

Os kits da Mekatronix e da Mobile Robots são bastante robustos nos seus materiais e na sua forma de construção (cilíndrica), perdendo terreno para os restantes no peso/dimensão e na flexibilidade de montagem.

Tabela 3.1: Tabela de avaliação das especificações de acordo com o Apêndice A

<i>Características</i>	Tabela de comparação			
	VEX Robotics	Mekatronix	Mobile Robots	Lego
Preço	**	**	*	***
Estrutura de Apoio	***	**	**	*
Baterias	***	**	**	**
Microcontrolador	***	***	***	***
Software de Programação	**	***	***	***
Pinos de E/S	***	***	***	**
Inclusão de módulos	***	***	***	**
Existência de Módulos para o kit	***	***	***	**
Peso	***	**	*	***
Dimensões	***	*	*	***
Materiais de Construção	***	***	***	*
Robustez	**	***	***	*
Flexibilidade	***	*	*	**

- * Não se adequa aos requisitos
 ** Suficiente para cumprir os requisitos propostos
 *** Indicado para cumprir os requisitos propostos

De todos, o kit da VEX Robotics é o que apresenta a melhor relação robustez/dimensões. Apesar de não ser tão robusto como os anteriores, as suas peças soltas construídas em alumínio conferem-lhe flexibilidade na montagem e um baixo peso, características necessárias para satisfazer os requisitos.

3.5 Conclusão

De acordo com a secção anterior, é possível verificar que o kit da VEX Robotics, de uma forma geral, aparenta ser a melhor solução de entre os kits candidatos.

A sua forma de montagem confere-lhe flexibilidade permitindo configurações completamente personalizadas e adequadas às suas missões e funcionalidades. Estas configurações e os materiais envolvidos permitem que seja manobrado facilmente por um operador.

É possível adaptá-lo facilmente para ultrapassar dificuldades de manobrabilidade e movimento.

Os sensores que acompanham o kit permitem uma variedade de funcionalidades e o microcontrolador é reprogramável possuindo portas de E/S disponíveis ao utilizador. Este último aspecto é fundamental para a comunicação com a placa de FPGAs. Vem ainda equipado com vários tipos de rodas para diferentes terrenos.

Em suma, este kit contém as características que melhor se adequam ao que é pedido para o sistema a um preço acessível.

Capítulo 4

Tecnologias

Depois de definida a organização do nosso sistema, é agora altura de fazer a introdução das tecnologias escolhidas para criar o robô móvel. Falaremos acerca dos sensores, do controlador acoplado e do esqueleto do nosso robô que formará a sua estrutura física e aparência. Abordaremos ainda todos os aspectos críticos do desenho físico.

A informação existente sobre o kit robótico é pouca e encontra-se dispersa, de forma que a informação que apresentaremos neste capítulo é em grande parte resultado de um estudo aprofundado das características dos diversos componentes do kit.

Mencionaremos finalmente o cérebro da operação: o FPGA e o sistema que nela será embebido.

4.1 Kit Robótico VEX

A presente secção aborda as propriedades associadas ao kit robótico e os aspectos da montagem que consideramos pertinentes para a realização deste projecto.

4.1.1 Unidade de Controlo

A Unidade de Controlo (UC) do kit robótico fornece ao utilizador 16 portas de E/S digitais ou até 16 portas de entrada analógicas, entre as quais se incluem pinos T_x e R_x , sendo esta escolha configurável. Disponibiliza igualmente mais 6 portas digitais que vêm configuradas como interrupções do controlador e ainda 8 portas configuradas para PWM [6]. Além disto existem ainda as seguintes interfaces:

Porta série: Permite ao utilizador comunicar com a UC;

Portas de Recepção R_{x1} e R_{x2} São utilizadas para conectar receptores de RF para os controlos remotos, também designados como Interface de Operador (IO). Desta forma, dois operadores

podem controlar partes diferentes do robô ao mesmo tempo. Este modo de funcionamento requer dois conjuntos de cristais diferentes, um para cada operador;

Conector da bateria: Conector para uma bateria de 7.2V. A UC fornece energia aos componentes a si conectados.

LED R_{x1} e R_{x2} : Luzes indicadoras da comunicação com os respectivos receptores;

LED PGRM: Luz indicadora de programação a decorrer;

LED Bateria: Indicação de bateria fraca. Fica verde se a bateria estiver entre níveis aceitáveis e vermelho se a bateria estiver a necessitar de ser recarregada.

Na Figura 4.1 observa-se a configuração física da UC.

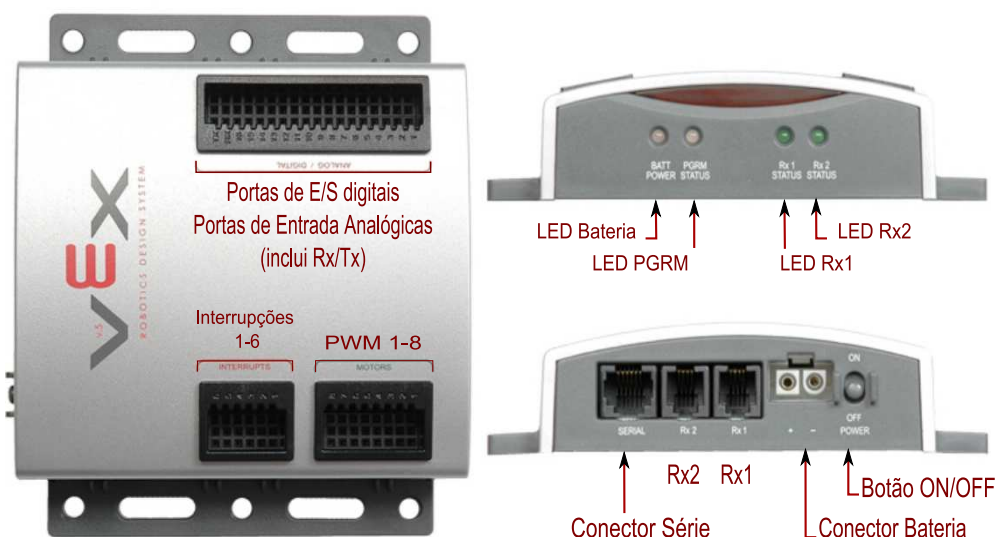


Figura 4.1: Controlador da VEX Robotics [1].

Na sua operação normal, a UC está conectada ao Interface de Operador (IO), recebendo os seus dados. De acordo com os dados recebidos e os dados recolhidos dos sensores, executa as funções programadas. A IO estabelece uma via de comunicação entre um utilizador e a UC, através de comunicação rádio e protocolo RS-422, respectivamente.

Internamente, a UC é composta por dois Microchip PIC18F8520 que trocam dados entre si e interagem com os portos de E/S.

O microcontrolador MASTER vem pré-programado com *firmware* para gerir as operações internas na UC. Não é permitido acesso para o reprogramar, no entanto, o utilizador consegue aceder às portas PWM que este microcontrolador controla.

O microcontrolador USER, como o nome indica, está acessível ao utilizador para ser reprogramado. Através deste microcontrolador tem-se acesso directo às portas analógicas e digitais disponíveis na UC. O utilizador pode reprogramar o microcontrolador USER criando funções

customizadas para as mais variadas tarefas, que incluem funções autónomas. Esta configuração interna da UC é visível na Figura 4.2.

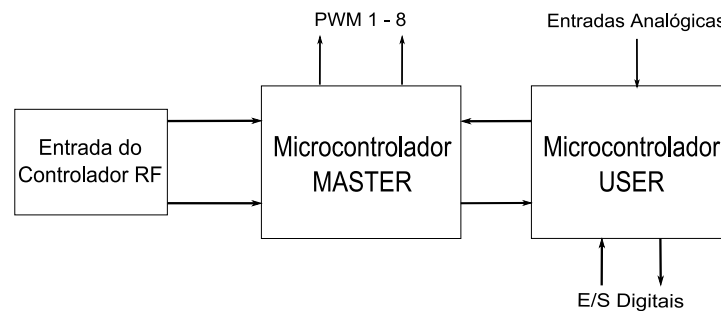


Figura 4.2: Configuração interna da Unidade de Controlo.

A UC vem de fábrica pré-programado com um *firmware* genérico de controlo do robô através do Interface de Operador de um controlo remoto de rádio-frequência (RF). Ambos os microcontroladores USER e MASTER podem ser programados partir da porta série e é importante que se conheça as funções pré-programadas e as estruturas pré-definidas antes de o fazer, pois estas auxiliam bastante a relação com a UC. O código fonte disponível no *site* da VEX Robotics ¹ encontra-se escrito em Linguagem C, com trechos em linguagem Assembly e é um ponto de partida para a customização de código, escrito preferencialmente em Linguagem C. Para reprogramar é aconselhável utilizar as seguintes ferramentas:

- MPLAB©IDE. Programa para Windows de utilização gratuita para microcontroladores PIC;
- O compilador a utilizar sugerido é o MPLAB C18 v2.40;
- IFI Loader para realizar o download do programa. Ferramenta disponibilizada no *website* da VEX Robotics [1].

Para garantir que a UC funcione correctamente é recomendado que se mantenha a estrutura do código exemplo fornecido. A Figura 4.3 retrata a estrutura de ficheiros providenciada pelo código exemplo. Após uma observação atenta, constata-se que o programa consiste num número significativo de ficheiros.

O modo de funcionamento geral do código fonte que vem pré-programado de raiz encontra-se retratado na Figura 4.3. O programa inicia em *main.c*. Esta começa com a função *IFI_Initialization* que realiza a configuração inicial de parâmetros do sistema como os registos e os *buffers* R_x e T_x do *Serial Peripheral Interface* (SPI). É uma função que não deve ser alterada. De seguida é invocada a função *User_Initialization*. Esta sim, pode ser alterada de acordo com as necessidades do utilizador. Nesta função existe a possibilidade de configurar os pinos: digitais de entrada ou saída, ou analógicos de entrada [5].

¹<http://www.vexrobotics.com/vex-robotics-downloads.shtml>

Após a inicialização, o programa entra na rotina principal. Esta rotina interage com funções presentes em vários ficheiros.

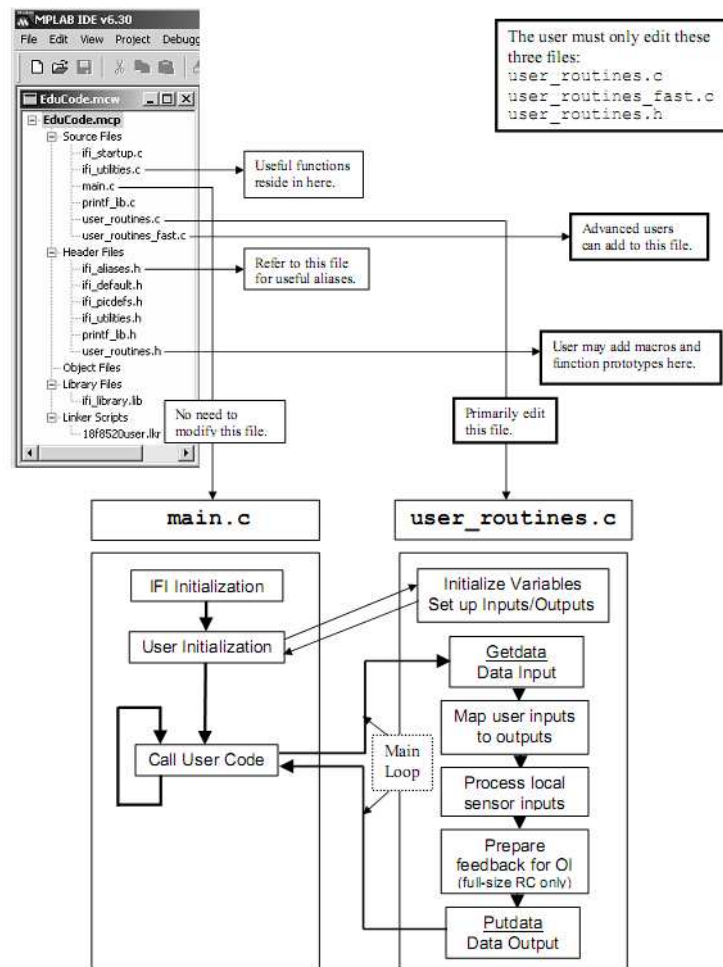


Figura 4.3: Estrutura do programa original [5].

Na Figura 4.4 observa-se a operação genérica do programa pré-instalado. É aconselhável manter esta estrutura no desenvolvimento de novas funções/rotinas. De maneira geral o microcontrolador MASTER comunica com a Interface de Operador através das portas R_{x1} e R_{x2} recebendo as indicações do controlo remoto e retransmitindo-as para o microcontrolador USER. Este executa as funções programadas e utilizar os dados recebidos do microcontrolador MASTER. Após a conclusão das suas funções, o microcontrolador USER reenvia para o MASTER dados possivelmente alterados dos portos PWM que o MASTER irá mapear. Todo o processo descrito se repetirá até que o sistema seja desligado.

Apesar de sermos aconselhados a manter a estrutura, temos a possibilidade de alterar os ficheiros sem interferir com funções específicas da Unidade de Controlo.

- user_routines.c

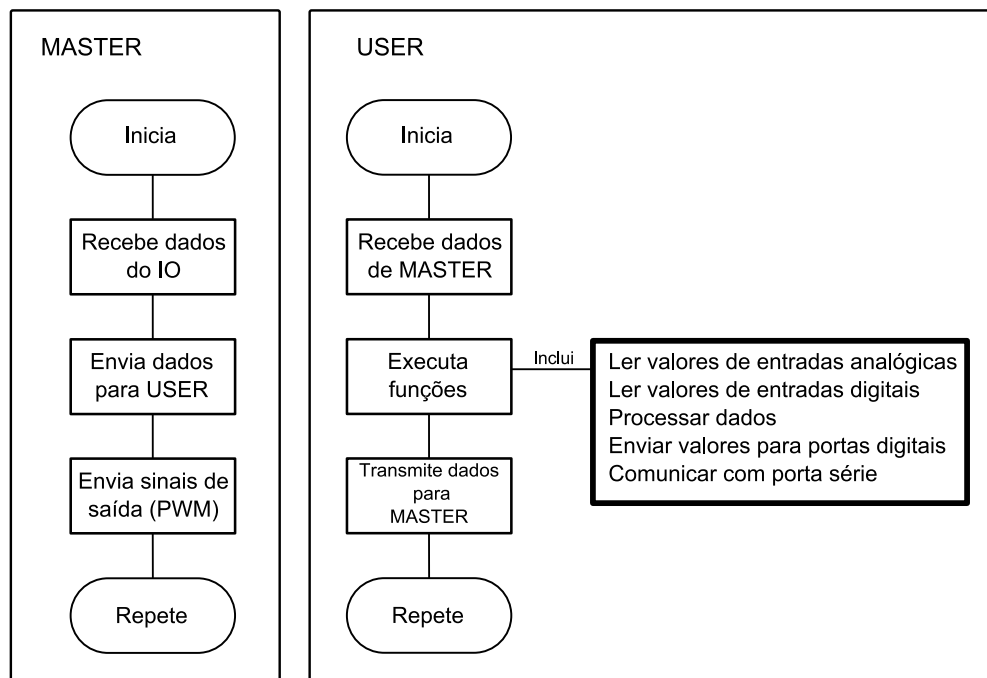


Figura 4.4: Diagrama de operação do programa principal.

- `user_routines_fast.c`
- `user_routines.h`

Como já tínhamos referido, a função `main.c` inicia com a execução das funções de inicialização `IFI_Initialization` e `User_Initialization`. Ao entrar no `loop` infinito, caracterizado por `while(1)`, a rotina vai verificar continuamente se a `flag Statusflag.NEW_SPI_DATA = 1`. Esta `flag` é enviada pelo microcontrolador MASTER para avisar o microcontrolador USER que existem novos dados para serem enviados, dados que contêm informações do controlo remoto.

Se `Statusflag.NEW_SPI_DATA = 0`, a rotina executa a função `Process_Data_From_Local_IO`, disponível no ficheiro `user_routines_fast.c`.

Se por outro lado `Statusflag.NEW_SPI_DATA = 1`, facto que nesta UC acontece de 18.5ms em 18.5ms, então a rotina irá executar a função `Process_Data_From_Master_uP`. Dentro deste ciclo, verifica ainda a `flag autonomous_mode = 1`. Se assim for entra na função `user_autonomous_code`, disponível no ficheiro `user_routines_fast.c`. A `flag autonomous_mode` é definida nas configurações e indica se a UC irá executar funções de modo autónomo [5]. Na Figura 4.5 está presente, na forma de um diagrama de fluxo, o funcionamento do algoritmo principal contido em `main.c`.

O Algoritmo 4.1 apresenta sob a forma de pseudo-código as etapas realizadas em `main.c`.

Vimos atrás que a determinada altura na função principal `main.c`, existia uma invocação da função `Process_Data_From_Master_uP`, contida no ficheiro `user_routines.c`. Esta função é chamada de 18.5ms em 18.5ms, porque é a este ritmo que o microcontrolador MASTER envia novos dados para o microcontrolador USER. Nesta função a primeira invocação ocorre para a função

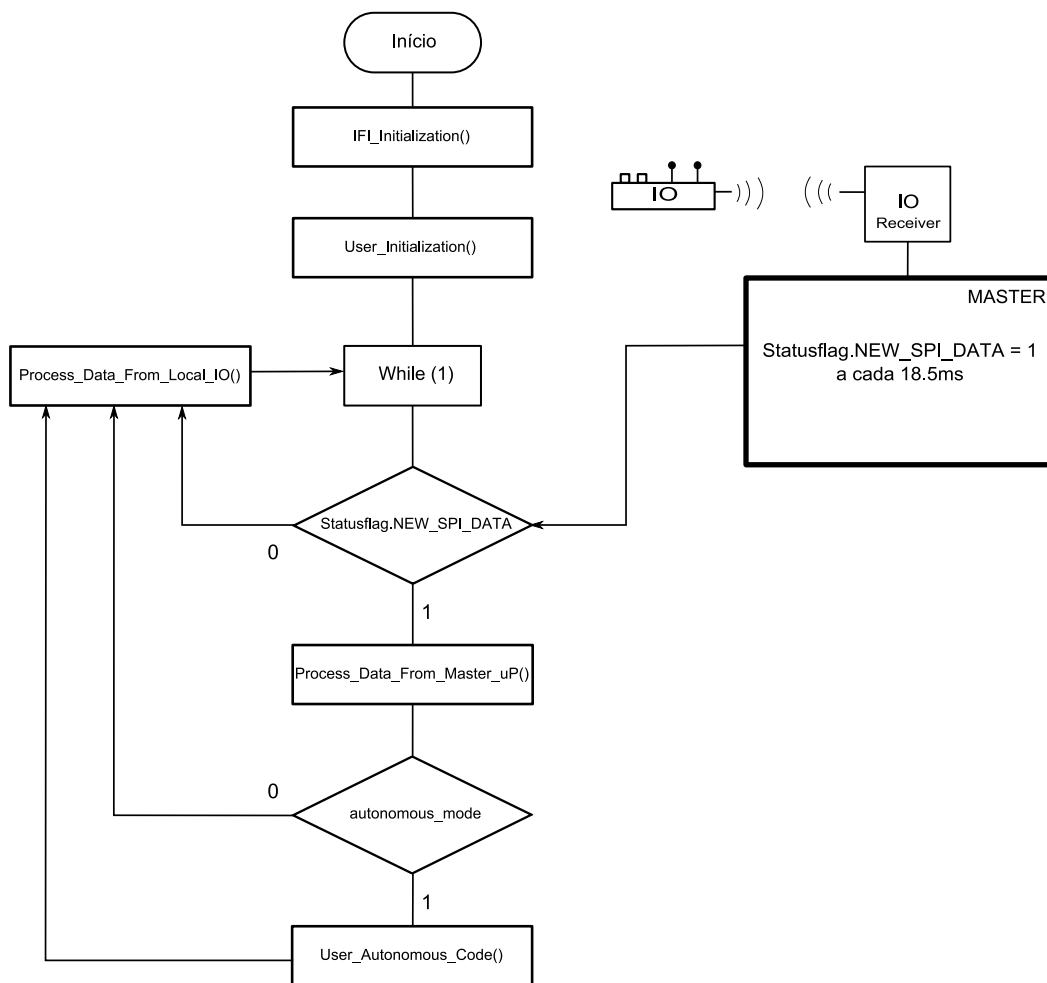


Figura 4.5: Diagrama de fluxo do programa principal.

Algoritmo 4.1 main.c

```

1: IFI_Initialization()
2: User_Initialization()
3:
4: procedure MAIN
5:   while 1 do
6:     if Statusflag.NEW_SPI_DATA then
7:       Process_Data_From_Master_uP()
8:     if autonomous_mode then
9:       User_Autonomous_Code()
10:    end if
11:  end if
12:  Process_Data_From_Local_IO()
13: end while
14: end procedure

```

Getdata, que é responsável por fazer a leitura de fluxo de dados enviados em série pelo microcontrolador MASTER. É aconselhável manter esta função na localização original, para manter o correcto funcionamento da UC [5].

Após a execução da função *Getdata*, pode-se executar uma qualquer função customizada, que para efeitos de exemplo designaremos por *user_routine*. Por omissão, esta função contém as funcionalidades básicas do robô em modo de controlo remoto. Esta função pode ser alterada ou até apagada.

Para concluir a função *Process_Data_From_Master_uP* é invocada a função *Putdata*. Nesta função é enviado para o microcontrolador MASTER um fluxo de dados em série. Dados esses que o microcontrolador MASTER irá usar para activar portas PWM e monitorizar o estado do microcontrolador USER. Esta função não deve ser alterada, movida ou apagada, visto que, se o microcontrolador USER não executar esta função a cada 18.5ms, o microcontrolador MASTER desactiva o microcontrolador USER e a UC não funcionará. O LED PGRM ficará vermelho assinalando a ocorrência da desactivação do microcontrolador USER [5, 6].

As funções customizadas não deverão conter ciclos infinitos e deverão ser suficientemente rápidas por forma a não atrasarem a execução da função *Putdata*.

Pode-se observar um diagrama de fluxo simples na Figura 4.6.

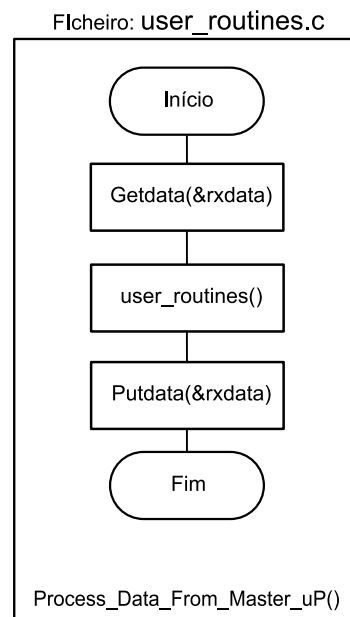


Figura 4.6: Diagrama de fluxo da função *Process_Data_From_Master_uP*.

Uma função muito importante para o desenvolvimento do nosso projecto é a função disponível ao utilizador *User_Autonomous_Code*. Nesta função poderemos desenvolver funcionalidades que dão autonomia ao nosso robô indo de encontro aos objectivos definidos.

O modo de funcionamento da função *User_Autonomous_Code* é muito semelhante à realizada na função *main.c*. Ao entrar nesta função todos os valores relativos às portas PWM são colocados

num estado neutro (127, para um motor significa permanecer estacionário sem rotação). Entra de seguida num ciclo que só termina quando a *flag autonomous_mode* for alterada por um qualquer processo para *autonomous_mode = 1*.

Dentro do ciclo principal continua a existir comunicação com o microcontrolador MASTER, para que a UC permaneça funcional. São invocadas as funções de comunicação entre microcontroladores já abordadas podendo ser executadas quaisquer funções customizadas desde que o período de comunicação de 18.5ms entre microcontroladores seja observado (e.g., *User_Function*). A Figura 4.7 representa o modo de funcionamento na forma de um diagrama de fluxo e o Algoritmo 4.2 o pseudo-código da função *User_Autonomous_Code*.

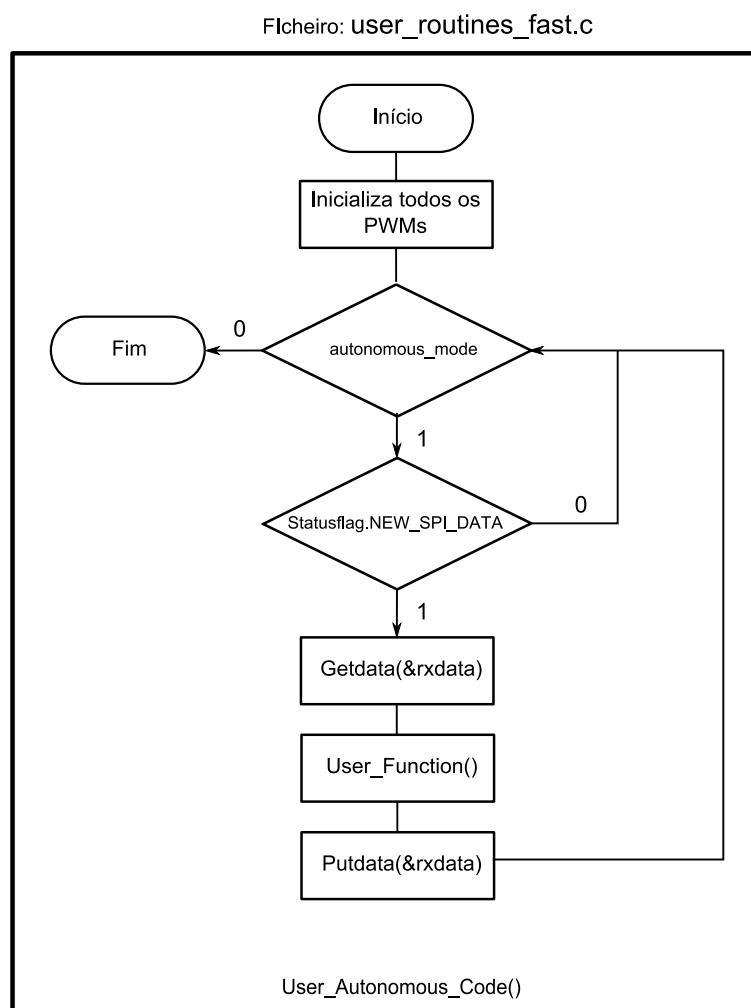


Figura 4.7: Diagrama de fluxo da função *User_Autonomous_Code*.

4.1.2 Estrutura

A estrutura física é o esqueleto do robô. Dá-lhe forma e suporta todos os componentes necessários à realização da missão a que é proposto. Necessitamos que a montagem tenha uma base

Algoritmo 4.2 main.c

```

1: PWM Initialization
2:
3: procedure User_Autonomous_Code()
4:   while autonomous_mode do
5:     if Statusflag.NEW_SPI_DATA then
6:       Getdata(&rxdata)
7:       User_Routine()
8:       Putdata(&txdata)
9:     end if
10:  end while
11: end procedure

```

plana para suportar a UC, a bateria e a placa de FPGAs. Os motores propulsão o robô. E aos veios que os motores movem devem ser acoplados os codificadores de impulsos para obter com o mínimo erro o número de rotações do motor.

Engrenagens

As engrenagens podem ser componentes vantajosos numa qualquer montagem mecânica. Uma combinação de rodas dentadas pode ser vista como um multiplicador de torque e um divisor de velocidade [6].

Entre as várias configurações possíveis com um conjunto de rodas dentadas, destacamos as funções:

- **Engrenagem condutora:** transmite força para girar outra Engrenagem. Esta é usualmente ligada ao veio de um motor;
- **Engrenagem Conduzida:** é accionada pelo força transmitida pela Engrenagem condutora;
- **Engrenagem de ligação:** encontra-se entre uma Engrenagem condutora e uma Engrenagem conduzida. O seu efeito na razão de velocidade e força é anulado. Gira no sentido oposto ao do movimento do robô. Também é utilizado para transmitir força ao longo de uma distância.

Estes conceitos podem ser visualizados na Figura 4.8. Algumas conclusões podem ser retiradas ao observar a figura. Podemos concluir que um número ímpar de rodas dentadas imprime movimento no sentido oposto de rotação do motor (consequentemente da Engrenagem condutora), enquanto que com um número par o sentido do movimento é o mesmo. O sentido de rotação das rodas dentadas alterna entre cada engrenagem.

A razão da força (Força que uma máquina consegue exercer dada a força que lhe é aplicada), pode ser calculada através do número de dentes que uma engrenagem possui (4.1).

$$\text{Razão de Força} = \frac{\text{Nº de dentes da engrenagem conduzida}}{\text{Nº de dentes da engrenagem condutora}} = \text{Torque} \quad (4.1)$$

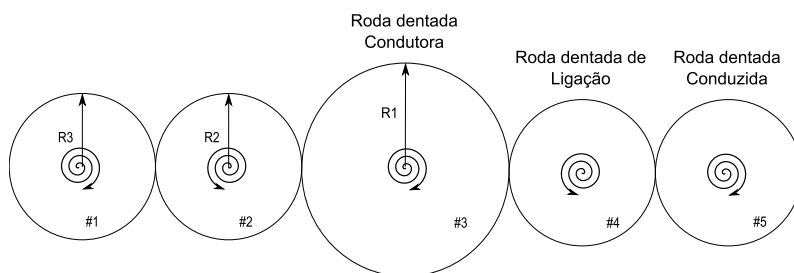


Figura 4.8: Engrenagens e tipos de rodas dentadas.

A força adicional transmitida por uma determinada configuração de engrenagens sacrifica a velocidade. A razão da velocidade é o inverso da razão de força (4.2)

$$\text{Razão de Velocidade} = \frac{1}{\text{Razão de Força}} \quad (4.2)$$

Actuadores

Os motores DC deste kit são controlados através de técnicas *Pulse Width Modulation* (PWM). Esta técnica permite o controlo de circuitos analógicos por meio de uma saída digital. O princípio subjacente é abrir e fechar um interruptor em intervalos de tempo uniformes. Utiliza uma onda quadrada cuja largura do impulso é modulada resultando na variação do valor médio da forma de onda (ver Figura 4.9).

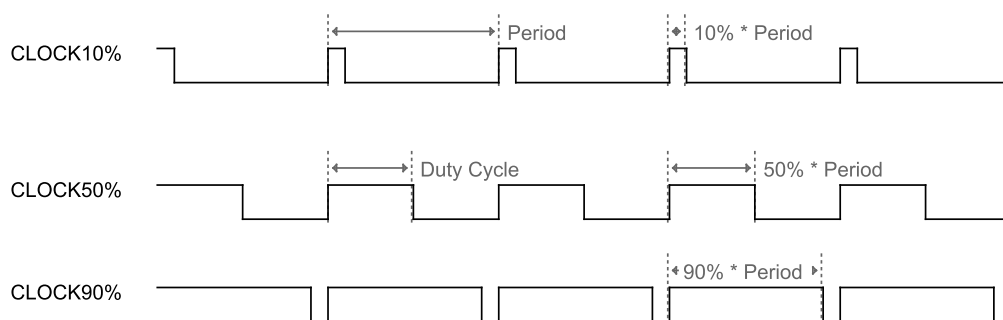


Figura 4.9: Variação do *duty cycle* em PWM

Se considerarmos uma onda quadrada $v(t)$, um valor mínimo V_{Min} , um valor máximo V_{Max} e um *duty cycle* D , o valor médio da forma de onda durante um período de tempo T é dado por (4.3):

$$\langle v(t) \rangle_T = \frac{1}{T} \int_T v(t) dt \quad (4.3)$$

com:

$$v(t) = \begin{cases} V_{Max} & \text{para } 0 \leq t \leq D \cdot T \\ V_{Min} & \text{para } D \cdot T \leq t \leq T \end{cases}$$

A expressão (4.3) desenvolve na seguinte (4.4):

$$\langle v(t) \rangle_T = \frac{1}{T} \left(\int_0^{DT} V_{Max} dt + \int_{DT}^T V_{Min} dt \right) \quad (4.4)$$

Se $V_{Min} = 0$ então $\langle v(t) \rangle_T = D \cdot V_{Max}$. O valor médio para $v(t)$ é directamente proporcional ao *duty cycle* da modulação.

Os motores fornecidos com o kit robótico são controlados através do software programado na Unidade de Controlo (ver Figura 4.10). No código pré-programado, é o microcontrolador MASTER que controla todas as portas PWM de acordo com os valores armazenados nas variáveis associadas às portas (*pwm0x*), que em linguagem de programação C, são variáveis *unsigned char* cobrindo a gama de valores $[0 : 255]$. Estes valores são enviados aquando da invocação da função *Putdata(&txdata)*. Assim os motores são alterados a cada comunicação de $18.5ms$, que é quando o microcontrolador MASTER recebe novos dados do microcontrolador USER.

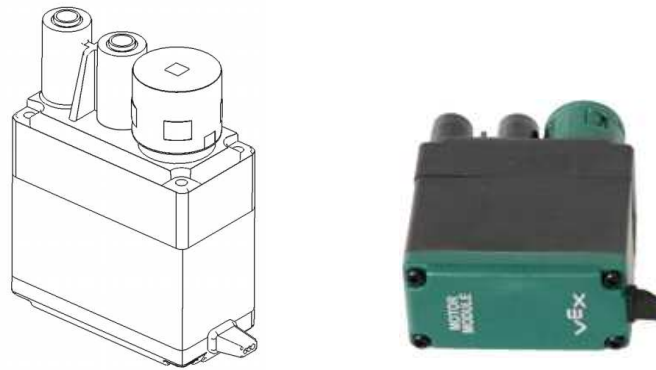


Figura 4.10: Imagem do motor fornecido com o kit robótico [6].

Existe a possibilidade de ser o microcontrolador USER a controlar as portas PWM. Para tal, é necessário definir esta escolha na função *User_Initialization*. Se o microcontrolador USER estiver configurado para controlar as portas PWM pode alterar o seu valor a qualquer altura. Esta funcionalidade, no entanto, não se encontra bem documentada.

Sendo o valor dos portos PWM alterado a cada $18.5ms$, este valor será o período da onda que controla as portas. O *duty cycle* é definido em software através de um valor de 8 bits, que toma valores no intervalo $[0 : 255]$, em que:

- **[255:128]**, sentido positivo de rotação, em que o extremo 255 corresponde à velocidade máxima;
- **127**, valor que para a rotação do motor;

- [0:126], sentido negativo de rotação, em que o valor 0 corresponde à velocidade máxima;

4.1.3 Sensores

Os sensores fazem medições do ambiente medições que auxiliam a realização de certas funções. Dão ao robô a capacidade de se mover num ambiente sem interacção humana, ajustando o seu comportamento ao elemento ambiental. Abordamos de seguida os sensores escolhidos como essenciais para a realização dos objectivos: o SONAR e os codificadores de impulsos.

SONAR

Ao uso de ondas ultra-sónicas para navegar e determinar a distância a objectos chama-se *SOund Navigation and Ranging* (SONAR). O SONAR emite uma onda de alta-frequência (tipicamente perto de $40kHz$) que alerta o robô para os obstáculos no seu percurso. O transmissor converte um impulso eléctrico numa onda de som através do *transducer* transmissor que é enviada através do ar.

Se a onda encontrar um objecto no seu caminho, vai reflectir de volta ao *transducer* receptor (ver Figura 4.11) que converte a onda de volta para um impulso eléctrico que pode ser interpretado por um microcontrolador.

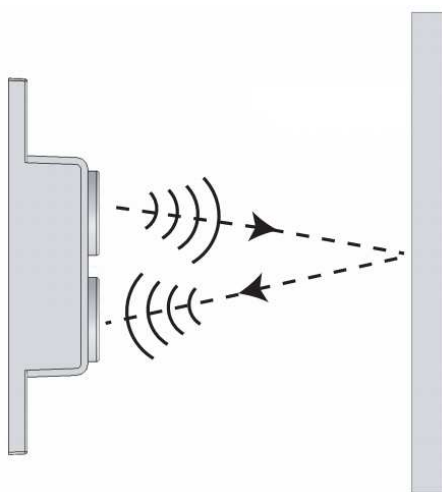


Figura 4.11: Princípio de funcionamento de um SONAR [6]

O tempo que demora desde o sinal transmitido a receber o eco (onda reflectida) permite determinar a distância ao objecto. Esta distância é calculada tendo por base a equação (4.5)

$$\text{distância ao objecto} = \text{velocidade do som} \times \text{tempo de ida e volta} \times \frac{1}{2} \quad (4.5)$$

De acordo com o valor aceite actualmente para a velocidade do som ($344m \cdot s^{-1}$ [19]), a equação (4.5) torna-se em:

$$\text{distância ao objecto} = 172 \text{ m} \cdot \text{s}^{-1} \times \text{tempo de ida e volta} \quad (4.6)$$

Codificadores de impulsos

Os codificadores de impulsos são vulgarmente utilizados para determinar movimento.

O princípio de funcionamento é o seguinte: uma luz de infravermelhos passa através de ranhuras equidistantes ao longo do perímetro de um disco que gira dentro do sensor. À medida que o disco gira, a luz é interrompida num padrão regular. Estas interrupções são processadas para determinar a quantidade de movimento do disco (ver Figura 4.12).

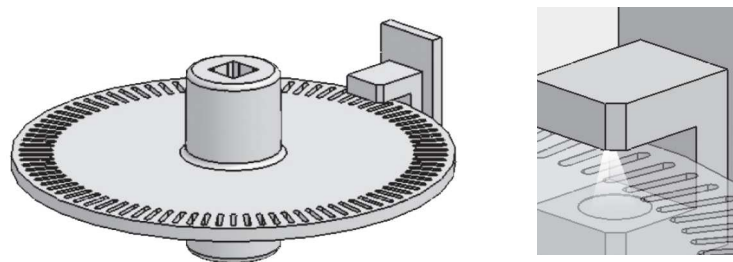


Figura 4.12: Princípio de funcionamento de um codificador de impulsos [6].

Se o codificador de impulsos estiver conectado directamente ao veio de uma roda de um robô, é possível determinar a distância que o robô percorreu tendo por base o perímetro da roda e o número de revoluções que o disco realizou.

Para calcular a quantidade de movimento de uma roda considera-se que a distância percorrida é igual a duas vezes o raio da roda vezes o número de revoluções completas do disco do codificador de impulsos (ver equação (4.7)) em que uma revolução equivale ao número de ranhuras presentes no disco do codificador de impulsos.

$$\text{distância percorrida} = 2 \times \text{raio} \times \text{número de revoluções} \quad (4.7)$$

No nosso caso, o codificador de impulsos do kit da VEX tem 90 ranhuras em cada disco e por isso uma revolução equivale a 90 ranhuras percorridas.

De acordo com a informação presente no manual de utilizador da VEX [6] o codificador de impulsos pode detectar até 1700 impulsos por segundo, o que corresponde a 18.9 revoluções por segundo e 1133 revoluções por minuto.

4.1.4 Energia

A fonte de energia utilizada para este projecto é uma bateria recarregável NiCd (ver Figura 4.13). A bateria utilizada fornece significativamente mais energia que um conjunto de pilhas NiCd do tipo AA.

A bateria fornecida apresenta as seguintes características:

- **Tensão 7.2V**
- **Carga 2000mAh**
- **Gama de Tensão com carga 7.1V a 7.8V**



Figura 4.13: Bateria fornecida com o kit robótico [6].

4.2 FPGA

A particularidade da tecnologia FPGA (Field Programmable Gate Array) [20] consiste na possibilidade de configurar o circuito integrado para um qualquer comportamento, por outras palavras um circuito dedicado a uma função específica. Assim sendo é possível implementar no FPGA qualquer tipo de lógica combinacional ou síncrona para o mais diverso tipo de aplicações.

A placa de desenvolvimento utilizada para este projecto possui um FPGA proveniente da Xilinx [21], mais especificamente o modelo equipado com o integrado XC3S200 [22]. Este FPGA permite a integração de até 200 mil portas lógicas.

No sentido de facilitar o desenvolvimento de sistemas integrados no FPGA são disponibilizadas ferramentas tal como é o caso do EDK (Embedded Development Kit) [23]. Este software permite o desenvolvimento no FPGA de sistemas embebidos muito completos e com diversos blocos funcionais, tal como microprocessadores, barramentos de comunicação e blocos desenvolvidos à medida.

Para este projecto em concreto utilizaremos o FPGA da placa de desenvolvimento para criar um sistema embebido que controle o robô. Nesse sistema utilizaremos o microprocessador incluído no EDK, Microblaze [24].

Embora facilmente se possa encontrar no mercado microcontroladores a preços bem mais reduzidos, a grande vantagem de utilizar esta arquitectura embutida num FPGA reside na possibilidade única de implementação de blocos IP (*Intellectual Property*) auxiliares, também embebidos no próprio FPGA. Estes blocos podem ser combinacionais e/ou síncronos com qualquer sinal

ou relógio disponível no FPGA. Assim abre-se um conjunto de possibilidades tais como isolar o controlo e atendimento dos módulos exteriores (motores ou sensores) ou mesmo redireccionar cálculos complexos. Deste modo permite a libertação do microprocessador para acções de coordenação e outras aplicações mas também reduz o tempo de obtenção dos resultados nomeadamente dos cálculos complexos, pois usando um módulo dedicado permite a realização do algoritmo com menores tempos de execução.

4.2.1 Microblaze

O Microblaze [24] é um microprocessador RISC (*Reduced Instruction Set*). É disponibilizado sob a forma de *softcore* e por isso permite seleccionar os blocos a incluir no sistema.

Um sistema embebido num FPGA permite a opção de além de incluir o microprocessador, conectar a este vários outros *IP cores* (e.g. GPIO, UART, INTC) conforme a necessidade e as portas lógicas disponíveis.

O Microblaze possui uma arquitectura de Harvard de 32 bit com barramentos de 32 bit para dados e instruções que executam programas e acedem a dados na memória interna e externa. Não faz a distinção entre acessos às E/S e à memória, pois acede às E/S através de mapeamento em memória. O Microblaze encontra-se optimizado para implementação em FPGAs da Xilinx [24].

A base desta arquitectura é constituída por:

- um sistema de *pipelining* de 3 ou de 5 estágios;
- um banco de registos com 32 registos de uso genérico;
- uma ALU (*Arithmetic Logic Unit*);
- *Barrel shifter*;
- dois níveis de interrupção

Além da configuração base, o microprocessador pode ser apetrechado com outras unidades para colmatar alguma insuficiência consoante as necessidades de um sistema embebido:

- divisor;
- multiplicador;
- uma unidade de FPU
- memória Cache.

Estes são alguns exemplos de blocos passíveis de serem acrescentados.

Como o Microblaze é um microprocessador *soft-core*, quaisquer blocos não utilizados não serão implementados no FPGA, libertando recursos para outras funcionalidades

O sistema de *pipeline* mais simples do Microblaze é paralelo e dividido em três estágios: Busca, Decodificação e Execução. Cada estágio demora, por norma, um ciclo de relógio a completar.

O Microblaze é um processador que só pode realizar as operações *load/store* de e para a memória. Não pode fazer operações em dados directamente na memória, logo os dados são transportados para dentro do microprocessador e colocados em registos de uso genérico para realizar qualquer operação.

Para pilha, o Microblaze utiliza uma metodologia FILO (*First In Last Out*). O primeiro valor a ser inserido na pilha é o último a sair. Os valores são inseridos na memória numa posição invertida, isto é, o primeiro valor é guardado na posição de memória mais alta, o seguinte na posição de memória anterior e assim sucessivamente. A stack cresce inversamente com o decréscimo da posição de memória [24].

Para suportar múltiplos pedidos de interrupções é necessário adicionar um controlador de interrupções (INTC) e a *flag* IE (Interrupt Enable) deverá estar activa $IE = 1$. Além de interrupções o Microblaze suporta outras funcionalidades como excepções.

Para tirar partido do microprocessador é conveniente acoplar algum tipo de periféricos, sejam eles fornecidos por uma entidade (e.g., Xilinx) ou desenhados especificamente para um fim, para haver algum tipo de interacção com o mundo exterior.

Descrevemos sucintamente os blocos de hardware considerados importantes para a implementação do protótipo: a *UART* e a *FPU*.

UART

O bloco UART serve para transmitir informação de depuração durante o desenvolvimento e será a forma preferencial de comunicação para transmitir os dados referentes à função a que este projecto se destina: o mapeamento. Isto é, tratará de reportar as respectivas probabilidades associadas à grelha no final da sua missão.

Single-Precision Floating Point Unit FPU

A unidade de FPU (Floating Point Unit) opera com aritmética de vírgula flutuante de precisão simples sendo compatível com a norma IEEE-754. Integrar uma unidade FPU num sistema embebido associado ao Microblaze aumenta consideravelmente a performance deste sistema na execução de cálculos com dados em vírgula flutuante. Está optimizado para ter um baixo custo e uma elevada performance [24].

A FPU realiza operações aritméticas comuns, tendo como menor latência de 4 ciclos para adição, subtração e multiplicação, 28 ciclos para a divisão e 1 ciclo para operações de comparação. Como é parte configurável do Microblaze não ocupa espaço extra se não for utilizado.

4.2.2 Projecto

Como já referimos, usamos para a fase de testes e de desenvolvimento a placa de desenvolvimento Spartan 3 Starter Kit [22]. Para a realização do projecto consideramos a utilização de núcleos IP que estejam otimizados para funcionar com o Microprocessador escolhido.

Como software de desenvolvimento para esta etapa utilizamos:

- Xilinx ISE Design Suite, versão 10.1;
- Embedded Development Kit, versão 10.1;
- Xilinx Platform Studio, versão 10.1;
- Core Generator, versão 10.1.

A arquitectura proposta para este projecto de engenharia conta, além do Microblaze, com módulos GPIO para interface com o exterior, sejam através de LEDs, DIP *switches* e pinos dos conectores de expansão, memória externa para albergar o programa principal e os dados, módulos de controladores de interrupções e temporizadores.

Na Figura 4.14 apresenta-se uma arquitectura proposta para implementação que contempla alguns dos blocos já referidos.

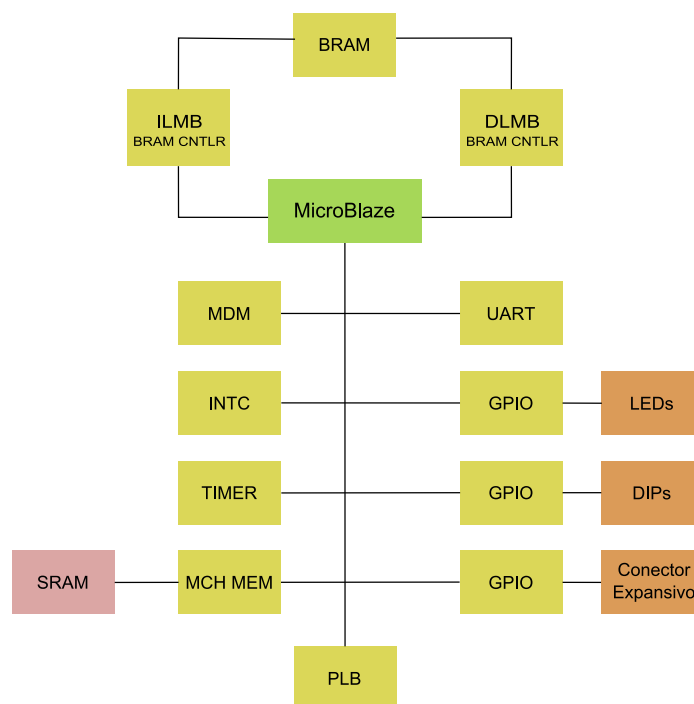


Figura 4.14: Arquitectura proposta para o Sistema embebido.

4.3 Resumo

Neste capítulo foram abordadas as tecnologias envolvidas na realização do projecto.

Dado que o kit robótico adquirido não tem uma boa base documental, foi só após pesquisa e estudo intenso de sistemas complementares e alternativos possível realizar com pormenor esta descrição, assim como cumprir os objectivos que tinham sido propostos.

Caracterizamos com pormenor o funcionamento interno da UC e apresentamos a estrutura de código fonte a utilizar para manter o bom funcionamento da UC.

Descrevemos alguns conceitos associados à montagem física do robô, assim como os métodos de funcionamento e respectivas conexões dos sensores de interesse para este projecto.

Toda a descrição realizada neste capítulo teve em vista a apresentação sob a alçada dos subsistemas definidos no Capítulo 3, para realizar a ponte entre o que era desejado e o que realmente irá ser implementado.

Por fim, apresentamos uma tecnologia com amplo suporte de documentação: o FPGA, o microprocessador Microblaze e as ferramentas de desenvolvimento.

Capítulo 5

Implementação do Protótipo

Este capítulo apresenta o produto final resultante das considerações que têm vindo a ser feitas até então. O resultado é um robô móvel com possibilidade de controlo manual e autónomo através de um sistema embebido integrado num FPGA.

Nas próximas secções demonstramos a interligação dos vários subsistemas e a implementação de código fonte e configurações físicas protótipo.

5.1 Subsistema Estrutura

Nesta secção apresentamos o resultado final do aspecto físico do robô, das suas ligações físicas e posição de sensores e motores.

5.1.1 Estrutura de Suporte

Levando em consideração os aspectos propostos no Capítulo 4, o robô foi montado com o intuito de ser o mais simétrico possível e com o peso distribuído uniformemente.

As respectivas dimensões do robô são:

Altura	=	12 <i>cm</i>
Largura	=	20 <i>cm</i>
Comprimento	=	24 <i>cm</i>
Distância entre eixos	=	10 <i>cm</i>
Peso	=	<i>Aproximadamente 2 Kg;</i>

Na Figura 5.1, demonstra-se a parte inferior da montagem efectuada para o robô. Nesta figura consegue-se observar diversos componentes numerados que apresentamos de seguida:

1. Motor esquerdo;
2. Codificador de impulsos;
3. Roda de 2.75" ou aproximadamente 7 cm;
4. Roda dentada de 60 dentes;
5. Roda dentada de 36 dentes;
6. Interface com o utilizador. Receptor RF.

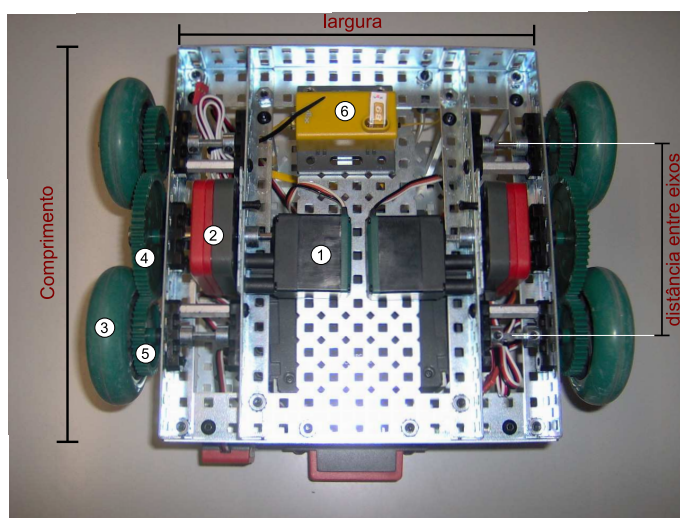


Figura 5.1: Parte inferior do robô.

Ainda, na Figura 5.1, demonstra-se a parte frontal da montagem efectuada para o robô. É possível observar os seguintes elementos:

1. SONAR colocado a cerca de 11 cm acima do solo;
2. Unidade de Controlo do kit robótico;
3. Roda de 2.75" ou aproximadamente 7 cm;
4. Sensor de luminosidade.

5.1.2 Propulsão

O robô tem uma engrenagem de três rodas dentadas de cada lado como vemos na Figura 5.1 e que estão na base do método de propulsão. O motor está localizado no centro do robô e está conectado à engrenagem do meio: a engrenagem condutora. Esta tem 60 dentes. Os eixos das rodas encontram-se conectadas às engrenagens de 36 dentes (engrenagens conduzidas).

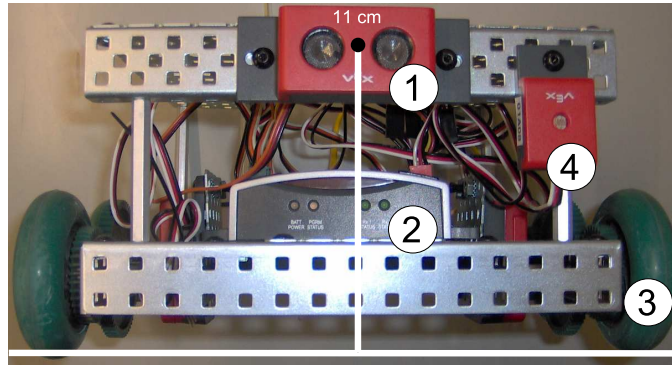


Figura 5.2: p Parte frontal do robô.

Na Figura 5.3 encontra-se um esquema de como está efectuada esta ligação. Como podemos observar da Figura, a engrenagem condutora movimenta as rodas conduzidas no sentido oposto ao seu.

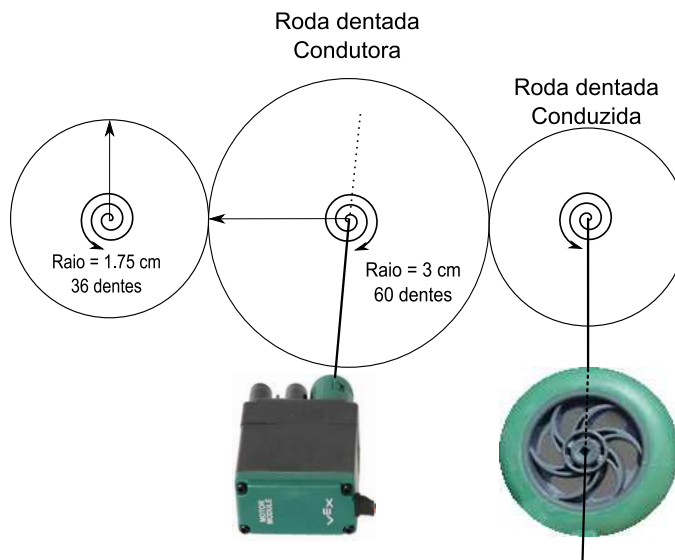


Figura 5.3: Configuração das engrenagens de propulsão.

A montagem apresentada implica que o robô tenha tracção às quatro rodas. A velocidade linear do robô pode ser modificada alterando o raio das rodas. Usando rodas maiores permite que com o mesmo número de rotações do motor, o robô percorra mais terreno. Seja o raio da engrenagem condutora $R_1 = 3 \text{ cm}$, o raio da engrenagem conduzida $R_2 = 1.75 \text{ cm}$ e o raio da roda $R_3 = 3.5 \text{ cm}$. A distância percorrida torna-se possível de calcular através da equação (5.1)

$$\text{distância} = GR \times 2 \times \pi \times R_3 \frac{n^\circ \text{ impulsos}}{90 \text{ impulsos}}$$

com,

$$GR \text{ (Razão das Engrenagens)} = \frac{60}{36}$$

Na configuração de 60 : 36 a velocidade máxima do robô é de $2.0fps \simeq 60cm/s$ [25].

Para realizar uma curva é determinado que os motores girem na mesma proporção em sentidos opostos para que o deslocamento do seu centro geométrico seja aproximadamente nulo. Isto é, gira sobre o seu próprio eixo.

5.2 Subsistema Energia

Para realizar a alimentação dos componentes do nosso protótipo utilizou-se uma bateria de NiCad com capacidade de 2000mAH, ligada à UC e à placa de FPGAs da forma que é representada na Figura 5.4.

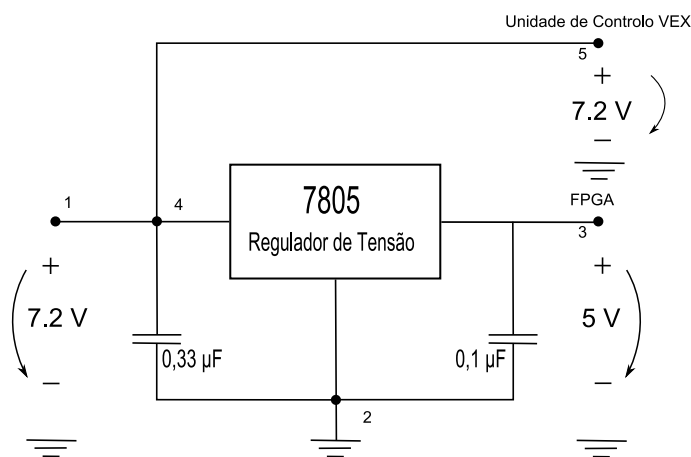


Figura 5.4: Circuito eléctrico de alimentação às unidades de controlo.

5.3 Subsistema Controlo

O sistema mais complexo é sem dúvida o subsistema de controlo, aliás como já tivemos oportunidade de referir. É, no entanto um desafio, explorar as formas possíveis de conexão e comunicação entre sistemas diferentes de forma a se obter a melhor solução.

Na Figura 5.5, apresenta-se aquela que é a ligação física realizada entre a placa de FPGAs e a UC. Observa-se que os codificadores de impulsos e a saída do SONAR estão ligados às portas de interrupção da UC. Visto que uma interrupção ocorre a uma cadência muito menor que a frequência de funcionamento da UC, este é o melhor método para realizar as contagens e diminuir o erro, especialmente nos codificadores de endereços.

As ligações entre os dois componentes é realizada apenas por fios condutores. Em termos lógicos, a sua comunicação poderia ser um qualquer protocolo suportado por ambos. No entanto,

dado que só temos acesso a algumas portas do microcontrolador USER, que nem são seguidas na sua lógica interna, a conexão directa com fios condutores é a solução imediata e até preferível, no sentido de obter um melhor controlo sobre a comunicação.

Para estabelecer a comunicação entre os dois dispositivos foi implementado um protocolo simples de transmissão de 4 bit de cada vez e que se encontra disponível para consulta no Apêndice C. Sendo a conexão realizada apenas por fios condutores, o protocolo a implementar pode ser alterado se as circunstâncias assim o exigirem.

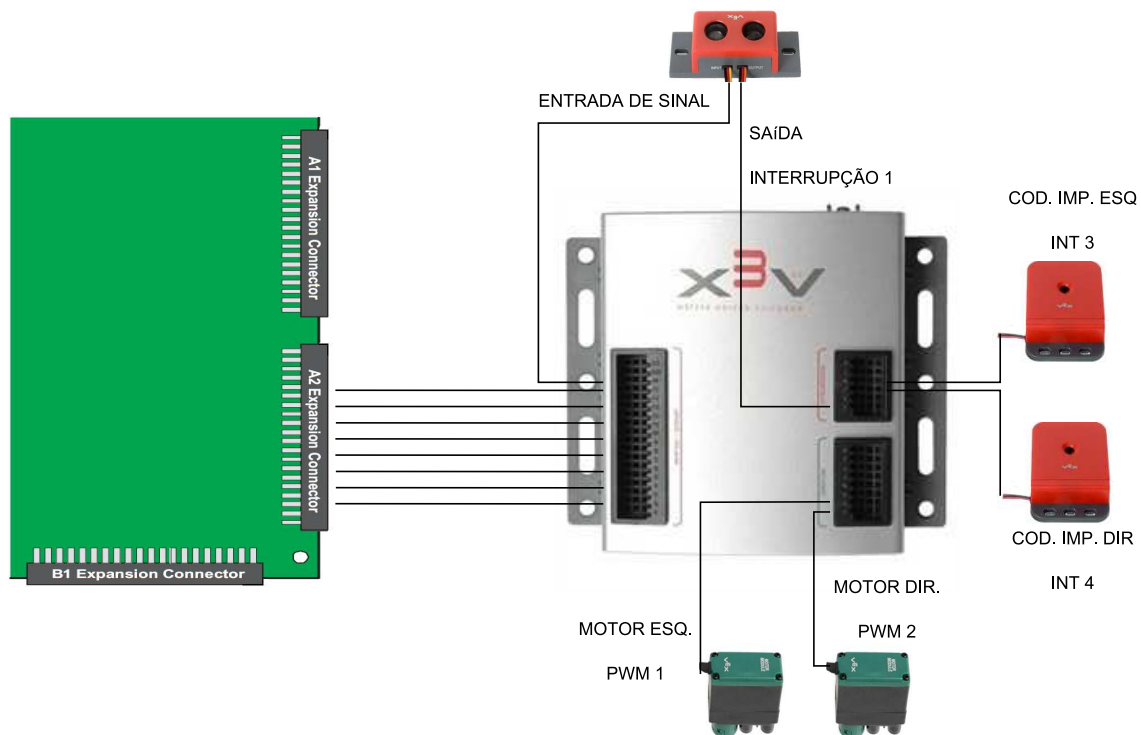


Figura 5.5: Conexões Físicas do Protótipo.

Após conhecermos as conexões físicas realizadas, mostramos na Figura 5.6 a arquitectura interna do protótipo. Vemos do lado direito um esquema já nosso conhecido do Capítulo 4. O Microblaze com os seus periféricos e barramentos. Neste caso ao Microblaze estão conectados periféricos que accionam LEDs, DIPs e ainda pinos de um conector expansivo. Ao conector expansivo estão ligados os fios condutores que estabelecem a comunicação com a UC do kit robótico. Vemos também um bloco referente à memória SRAM, que no nosso caso é onde está alojado o programa principal e os dados, por não haver memória interna (*Block RAMs*) suficientes nesta placa de desenvolvimento.

Por último, o bloco UART é o método de comunicação externa preferencial para depuração de código e para quando a missão de mapeamento estiver concluída transmitir as probabilidades da grelha de ocupação.

Do lado direito da Figura 5.5 podemos ver a arquitectura interna da UC já conhecida e a forma como conecta internamente com os restantes elementos. Como referido os fios condutores estão

ligados às portas de E/S disponíveis na UC que interagem directamente com o microcontrolador USER. Este comunica internamente com o microcontrolador MASTER que por sua vez recebe os comandos enviados por controlo remoto de um operador.

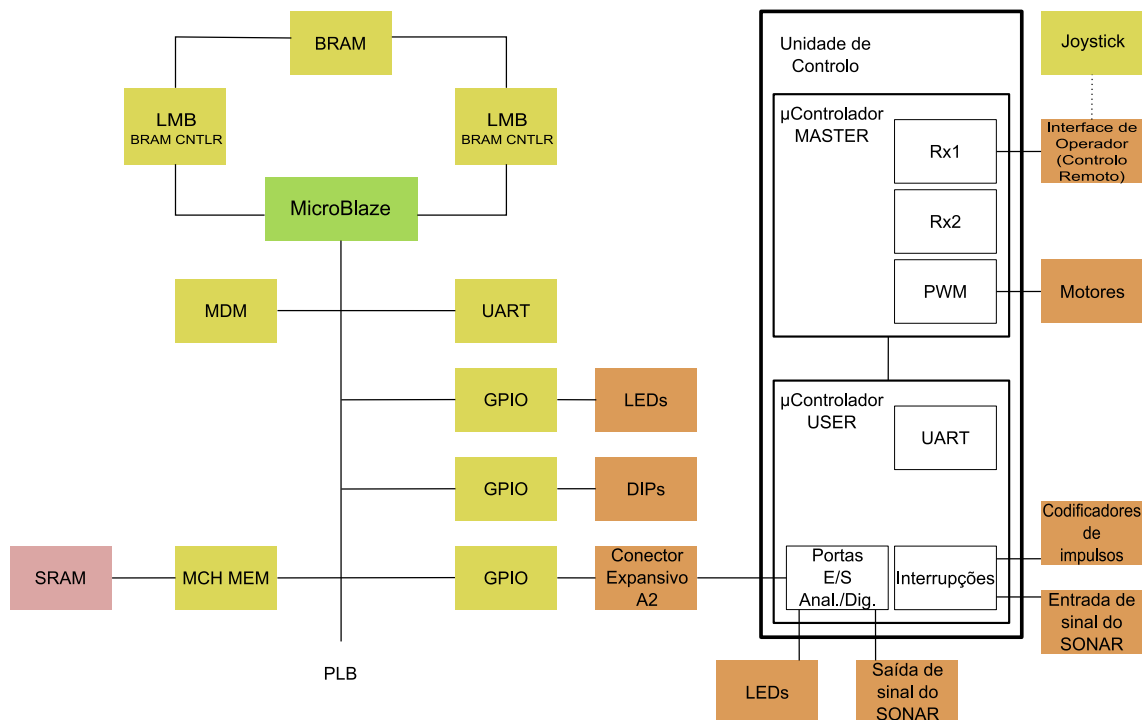


Figura 5.6: Arquitectura Interna do protótipo.

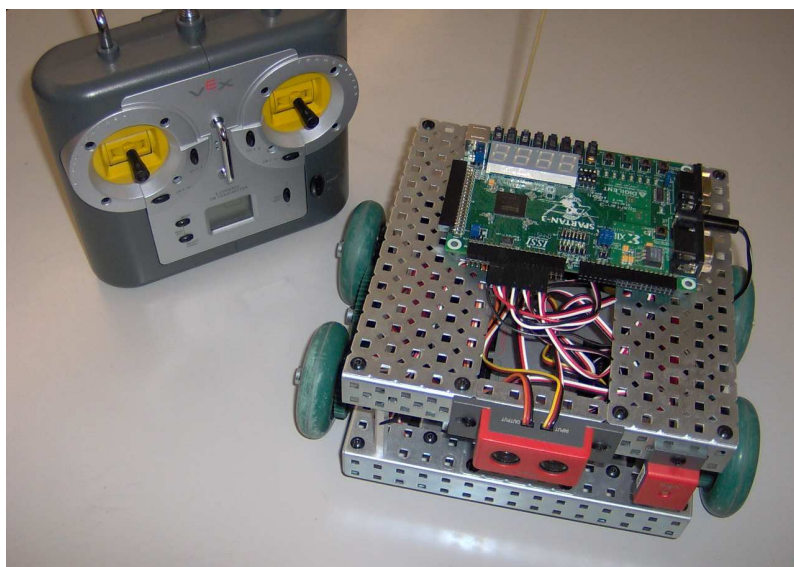


Figura 5.7: Protótipo implementado.

5.3.1 Modos de Funcionamento

Agora que já conhecemos a arquitectura lógica e física do nosso protótipo é altura de conhecermos os três modos de funcionamento implementados:

- **Modo 1:** É o modo normal de funcionamento de um carro remoto. É controlável por controlo remoto e o caminho percorrido não é contabilizado para efeitos de mapeamento. Pode ser utilizado para realizar pequenas correcções na trajectória ou para mover o protótipo para um ponto específico;
- **Modo 2:** Ao pressionarmos o botão referente ao Canal 5 do controlo Remoto, entramos num modo que apesar de se continuar a controlar o movimento do robô, a sua trajectória e obstáculos encontrados no ambiente são contabilizados para efeitos de mapeamento. Isto é, à medida que conduzimos o robô, o sistema de controlo realiza a sua tarefa de mapeamento. Os comandos enviados pelo Microblaze aos actuadores são ignorados;
- **Modo 3:** Se, enquanto no Modo 2, pressionarmos o botão referente ao canal 6, o robô entra no modo autónomo realizando as suas funções e comunicando com o Microblaze, enviando dados dos sensores para o Microblaze construir a grelha de ocupação e recebendo comandos para os actuadores.

Na Figura 5.8 encontramos um diagrama exemplificativo dos modos de funcionamento.

5.3.2 Lógica de Controlo

De uma forma genérica, a UC apenas retransmite os dados provenientes dos sensores para posterior processamento pelo Microblaze. Após a recepção destes dados, o Microblaze inicia um conjunto de tarefas necessárias à realização da sua missão. Começa por calcular a sua pose actual dados os novos valores odométricos. Depois deste cálculo realiza o preenchimento da grelha de ocupação levando em consideração os novos valores do SONAR, também transmitidos pela UC.

Para acelerar o teste do protótipo, optou-se por implementar um controlador reactivo no nosso robô. A tarefa de mapeamento é sempre realizada, no entanto, não é levada em conta no controlo de movimento do robô. Este ao detectar um obstáculo ou os limites da sua fronteira, realiza uma viragem para a esquerda para procurar novos caminhos. É um percurso algo aleatório, dependendo dos obstáculos que for encontrando. Se efectivamente encontrar um obstáculo, então é dada ordem de paragem ao robô, após a confirmação da imobilização é dada nova ordem para virar. Quando o ângulo de viragem equivaler ao submetido para evitar o obstáculo, o robô pára novamente para prosseguir em frente. Todos estes comandos são transmitidos na altura da comunicação entre a UC e o Microblaze.

Toda esta operação deve ser concluída em menos de 18 ms, devido ao facto de que se esse tempo for ultrapassado o microcontrolador MASTER da UC desactiva o microcontrolador USER e o nosso robô deixa de ser controlável. Existe portanto uma restrição temporal a observar.

A Figura 5.9 representa o modo de controlo do protótipo que temos vindo a descrever.

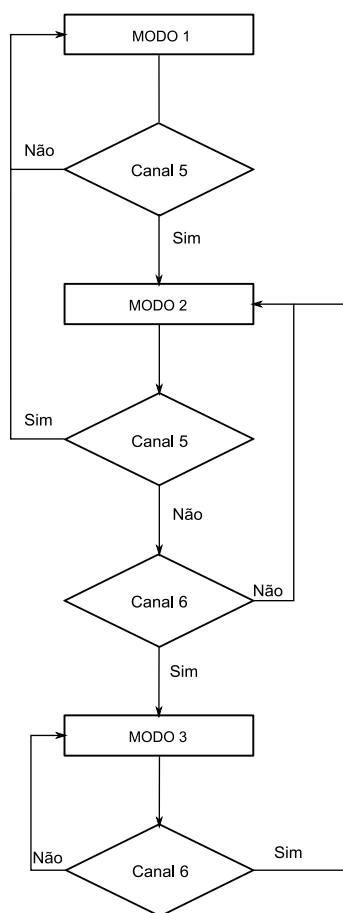


Figura 5.8: Modos de funcionamento do protótipo.

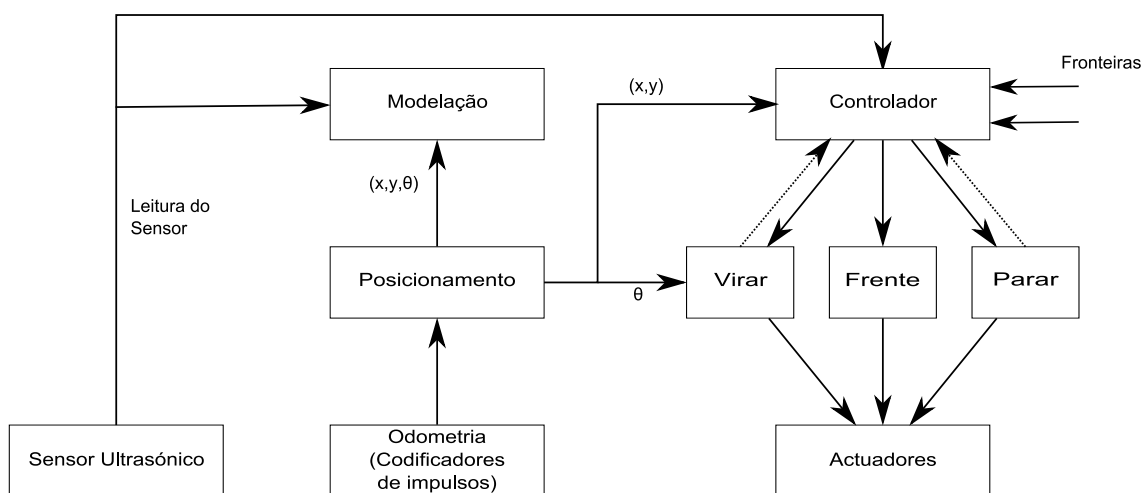


Figura 5.9: Diagrama de alto-nível do sistema de controlo.

No Algoritmo 5.1 encontramos as operações genéricas executadas no microprocessador Microblaze.

Algoritmo 5.1 Algoritmo de controlo presente no FPGA

```

1: procedure control
2:   Get sensor values from VEX UC
3:   Calculate current position from new data
4:   Verifies control of the robot (issues control commands)
5:   Update occupancy grid from sensor model
6: end procedure

```

Para testar efectivamente o FPGA como plataforma credível para sistemas embutidos, implementamos um algoritmo probabilístico de mapeamento que pusesse à prova o Microblaze enquanto *cérebro* da operação. Este algoritmo toma a forma de função apresentada na linha 5 do Algoritmo 5.1.

O código fonte disponível no apêndice D percorre todas as posições da matriz para calcular a probabilidade correspondente. Tendo já disponível a pose do robô, o algoritmo segue os seguintes passos:

1. Calcula o vector central para $\alpha = 0$;
2. Com base na posição actual do robô, calcula para cada posição o vector correspondente;
3. Calcula norma dos dois vectores (o central e o da nova posição)
4. Calcula o ângulo interno dos dois vectores;
5. Calcula a distância entre os dois pontos;
6. De acordo com a distância estabelece se há obstáculo dentro dos limites máximos do sensor ou não;
7. Actualiza a grelha de ocupação com o algoritmo probabilístico escolhido.

5.4 Resumo

Apresentamos neste capítulo a implementação dos vários subsistemas do protótipo e a relação entre estes vários subsistemas. Desta implementação abordamos alguns aspectos práticos considerados relevantes para a montagem do protótipo, tal como as relações entre engrenagens na determinação da distância percorrida e a localização dos codificadores de impulsos. Ficamos ainda a saber que em resultado da montagem efectuada, o protótipo tem tracção às quatro rodas e gira sobre o seu eixo, à semelhança de um tanque.

Demonstramos as ligações físicas dos diversos componentes e sua localização na estrutura, com uma fotografia do protótipo implementado.

Além da arquitectura física, apresentamos ainda a arquitectura interna e a sua organização lógica e de controlo. Demonstramos os vários modos de funcionamento do protótipo: por controlo remoto, por controlo remoto com mapeamento e movimentação autónoma com mapeamento.

Na lógica de controlo abordamos a estrutura da rotina implementada e as suas tarefas no controlo e construção da grelha de ocupação.

Capítulo 6

Resultados Experimentais

Este capítulo apresenta resultados experimentais, obtidos pela aplicação dos algoritmos e arquitecturas já apresentados.

Iniciamos a análise da performance dos nossos algoritmos de mapeamento com o *profiling* da implementação de Bayes no PC. Esta implementação do algoritmo de mapeamento servirá como referência para os testes iniciais de *profiling* no PC e no sistema embebido. São realizados testes para diversos níveis optimizações de compilação. Após discussão de resultados concluímos qual o nível com melhor performance e passamos a implementar outras formas de optimização, como a utilização de dados do tipo *float* e módulos adicionais de hardware. Além do *profiling*, realizamos também medição do tempo de execução das funções principais utilizando um temporizador. A comparação destes resultados permite-nos determinar a melhor opção em termos de tipos de dados a efectuar e a melhor arquitectura a usar, para tornar a execução o mais rápida possível.

Por fim, tendo determinado a melhor opção de hardware e de tipo de dados a utilizar comparamos ambas as implementações do algoritmo de mapeamento (Bayes e Dempster-Shafer) e discutimos as suas diferenças no que toca a tempos de execução.

Todos os testes foram efectuados num computador pessoal com as seguintes características:

- **Placa principal** ASUSTeK Computer INC. P5VD2-MX SE;
- **Processador** Intel(R) Core(TM)2 CPU 6320 @ 1.86GHz;
- **Frequência dos núcleos** 1600.0 MHz;
- **Memória instalada** DDR2 2048 MBytes at 266.7 MHz;
- **Sistema Operativo** Microsoft Windows XP Professional Service Pack 2 (Build 2600);

6.1 Resultados de Referência no PC

Para o desenvolvimento dos algoritmos, foram utilizadas as seguintes ferramentas:

- **Pacote de desenvolvimento** MingW, versão 5.1.4;
- **Compilador** gcc, versão 3.4.5;
- **Profiler** gprof, versão 2.18.50.20080625.

Para melhor podermos observar a evolução de optimizações iniciamos os testes do algoritmo de referência num PC. Os resultados do *profiling* do algoritmo de referência reflectem um ciclo de 10000 iterações do ciclo principal do programa, para uma matriz de 100×100 de dimensão e compilado com vários níveis de optimização de compilação. O algoritmo de referência refere-se à implementação de *Bayes*, para uma obstáculo localizado à distância média de 6 unidades. Os testes realizados no PC são posteriormente executados no sistema embebido. No decurso dos testes verificou-se que à distância máxima escolhida não era possível obter resultados por parte da ferramenta de *profiling* do EDK. A distância de 6 unidades é o valor que permite obter a generalidade dos resultados correctamente. Nesta implementação o algoritmo percorre cada uma das células da matriz 100×100 verificando se se encontra dentro da área de acção do cone para posteriormente executar cálculos que determinem a sua probabilidade. O Algoritmo 6.1 demonstra os passos mais importantes dos algoritmos de teste utilizados para a obtenção de resultados.

Algoritmo 6.1 Algoritmo de Referência

```

1: procedure main
2:   Initialize Grid           ► Bayes or Dempster-Shafer
3:   Update Matrix
4: end procedure

```

Os resultados de *profiling* para o algoritmo de referência no PC podem ser consultados no Apêndice E na forma de tabelas, para vários níveis de optimização. Visto que os totais acumulados nas tabelas se referem a 10000 iterações, apresentamos um resumo dos resultados totais na Figura 6.1, na forma de gráfico, para cada iteração.

Numa primeira observação constata-se que, curiosamente, os valores obtidos nestas simulações demonstram que a implementação realizada para a precisão *double* obteve tempos de execução mais baixos do que para a execução com precisão *float*. Este factor é deveras interessante e pode dever-se ao facto de se usarem funções específicas para a precisão que utilizamos, isto é, usamos certas funções optimizadas para a precisão *double* e alterá-mo-las para outras funções optimizadas para a optimização *float*.

O facto de usarmos um PC equipado com um Processador da Intel, indica que independentemente da precisão escolhida o processador e especificamente a sua unidade de vírgula flutuante guarda os valores nos seus registos sempre com precisão dupla estendida (80 bits) [26], sugerindo que para a precisão *float* existirão operações de conversão de *float* para *double extended*. Comparando as tabelas E.1 e E.2 nota-se um aumento preponderante das funções optimizadas para *float* *sqrtrf* e *acosf*. Nota-se ainda na tabela E.2 o aparecimento da função *pow* quando esta não teria sido explicitamente usada, indicando que terá sido invocada por outra função. Estes aspectos e o facto de para todos os testes a compilação com *float* ocupar mais de 200 *bytes* na secção de código

que as congéneres em *double* dão relevo à justificação de que existe de facto operações que em precisão *float* são adicionadas para a conversão de precisão simples para precisão dupla estendida.

A adição de cerca de 3.5Kb para auxílio no *profiling* em todos os casos contribui para o atraso na execução e para a diferença dos tempos relatados.

Outro fenómeno inesperado é o facto de ao utilizarmos optimizações na compilação (-O2 e -O3) estas obtêm piores resultados do que o nível -O0. Por breve inspecção de ficheiros Assembly gerados na fase de testes (para O0 e O2), verifica-se que funções que em O0 apareciam como funções de *store/load* da FPU, aparecem em O2 como *movl*. Esta situação pode dever-se ao facto de para o nível de optimização O2 e O3 a cache disponível não ser suficiente e passar então a utilizar acessos à memória, muito mais penalizadores em termos de ciclos de relógio consumidos.

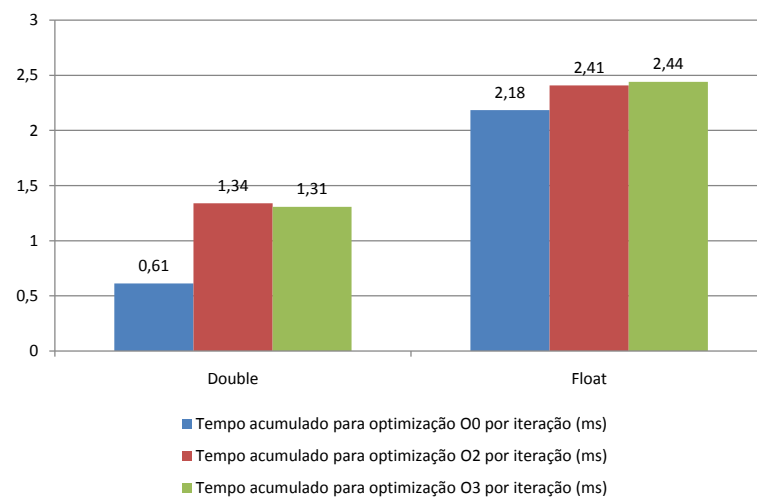


Figura 6.1: Tabela de tempos acumulados (em *ms*) para para os níveis de optimização testados no PC.

Face a estes resultados, num computador pessoal obteríamos melhores resultados com maior precisão e em menos tempo, o que é até o desejável, pois alia-se velocidade de execução a precisão de resultados.

6.2 Resultados Obtidos com o Sistema Embebido

Não obstante os resultados obtidos na secção anterior, realizamos um estudo à performance dos algoritmos de mapeamento naquele que será o sistema a utilizar, o sistema embebido contendo um Microblaze a operar a 50 MHz.

Seguimos uma metodologia que evoluísse do pior caso possível para o melhor. Iniciamos com uma implementação que percorre todas as células da matriz, independentemente do seu tamanho. Todos os testes foram realizadas para o algoritmo de referência com implementação de *Bayes* para uma matriz de dimensões 100×100 executada uma vez e com um obstáculo à distância média de 6 unidades.

Os resultados de *profiling* obtidos a partir do sistema embebido podem ser consultados no Apêndice E.

Iniciamos os testes retirando resultados de *profiling* para para precisão *double* e várias optimizações para ver a progressão do tempo de execução anunciado. Aqui, ao contrário do que acontece no PC, as optimizações melhoram a performance do sistema. A optimização O3 melhora em 1.4 vezes o resultado de referência sem optimização (O0). Por o software de *profiling* (EDK) não ter sido capaz de produzir resultados para a precisão *float* sem nível de optimização (O0) não nos é possível apresentar este dado.

No sistema embebido era esperado que a simples passagem de *double* para *float* melhorasse substancialmente os resultados. Isso de facto aconteceu obtendo, para uma compilação com optimização O2, uma redução no tempo de cerca de metade (696,08 para *double* em contraste com 335,00 para *float*). Continuamos o nosso estudo, desta vez com optimização de compilação O3. Globalmente melhoramos ambos os tempos obtidos para a optimização O2. Mas, a simulação com precisão simples (*float*) continua a obter os melhores resultados reduzindo novamente em cerca de 1,8 vezes o melhor tempo de precisão simples obtido para O2 (335,00 para *float* com optimização O2 em contraste com 186,90 para *float* com optimização O3). Estes valores encontram-se visíveis na Figura 6.2 na forma de um gráfico.

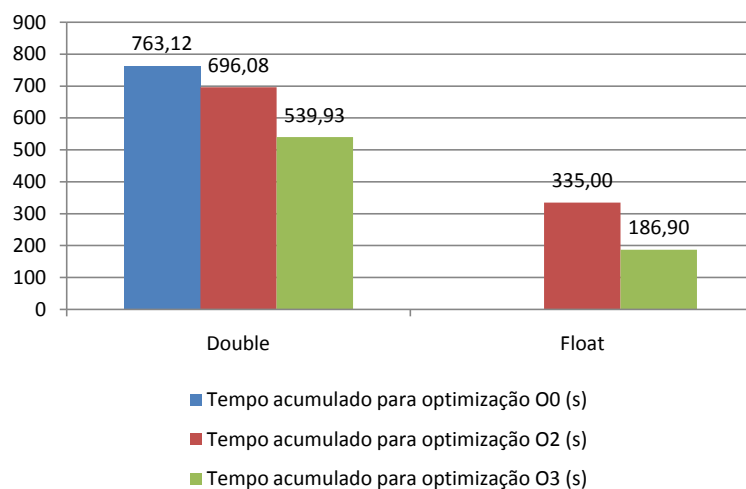


Figura 6.2: Tabela de tempos acumulados para para os níveis de optimização testados no Microblaze@50MHz.

Chegamos então à conclusão que a precisão simples obtém melhores *performances* que a precisão dupla no sistema embebido.

6.3 Optimizações Específicas

Sabendo já que a precisão simples é o método mais rápido procedemos à averiguação de tempos de execução quando se inclui módulos de hardware extra, tal como a unidade de vírgula flutu-

ante (*FPU*) e a inclusão de memória *cache*. Além dos resultados obtidos por *profiling* realizamos também testes para determinar o tempo real aproximado da função testada nos vários contextos, utilizando um temporizador. Estes resultados em conjugação com os de *profiling* são objecto de análise daqui em diante.

À inclusão de módulos adicionais, especificamente *FPU* e memória *cache*, realizamos também testes com a versão otimizada do algoritmo. Esta versão ao invés de percorrer todas as células da matriz, percorre apenas uma submatriz que contém a área de intervenção do cone associado ao modelo do SONAR, executando muito menos cálculos (ver Figura 6.3).

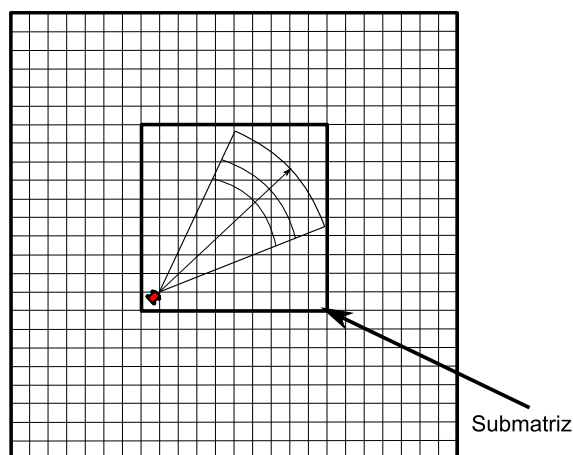


Figura 6.3: Exemplo de grelha com submatriz seleccionada.

Os resultados de *profiling* e do tempo de execução medido através de temporizador obtidos no sistema embebido podem ser consultados no Apêndice E.

Na Figura 6.4, visualiza-se um gráfico em escala logarítmica que compara os vários tempos de *profiling* e de medição com temporizador para várias alternativas de hardware e para ambas as opções do algoritmo (referência e otimizado, respectivamente). Todos foram obtidos com precisão simples e optimização de compilação O3. Desta Figura é possível constatar que para o *profiling*, a solução memória *cache* obtém a melhor performance (26,96s) para o algoritmo de referência. Esta diferença dilui-se quando comparamos os resultados obtidos por temporizador, em que tanto a solução com memória *cache* como a solução com *FPU* obtêm resultados muito semelhantes (11,03 para a solução com *cache* e 11,55 para a solução com *FPU*) com ligeira vantagem para a solução com *cache*. As soluções com blocos de hardware adicionais são aproximadamente 6,78 vezes mais rápidas na sua execução do que a solução simples.

Os resultados obtidos para o algoritmo de referência indicam que a inclusão de módulos de hardware extra auxiliam na obtenção de melhores *performances*. Para explorarmos ainda mais esta relação, realizamos os mesmos testes com a versão do algoritmo otimizado.

Apesar de nos testes de *profiling* todos os resultados serem aproximados (cerca de 21s), o tempo medido através do temporizador mostra uma realidade diferente. Das três soluções destaca-se a solução com memória *cache*, com um tempo cerca de 7,4 vezes melhor ao da pior solução, a solução sem blocos adicionais (0,222s para a solução simples e 0,030s para a solução com memória *cache*). Esta parece ser a melhor solução de entre as escolhidas para a implementação. Por fim, constata-se que para a placa de desenvolvimento escolhida *Spartan 3 Starter Kit* com o integrado XC3S200 a ocupação de portas lógicas se encontra próximo do máximo (entre 93% para a solução simples e 99% para a solução com *FPU*). Este facto não permitiu que fosse possível determinar nesta placa de FPGAs uma solução que combinasse memória *cache* e unidade de vírgula flutuante *FPU*.

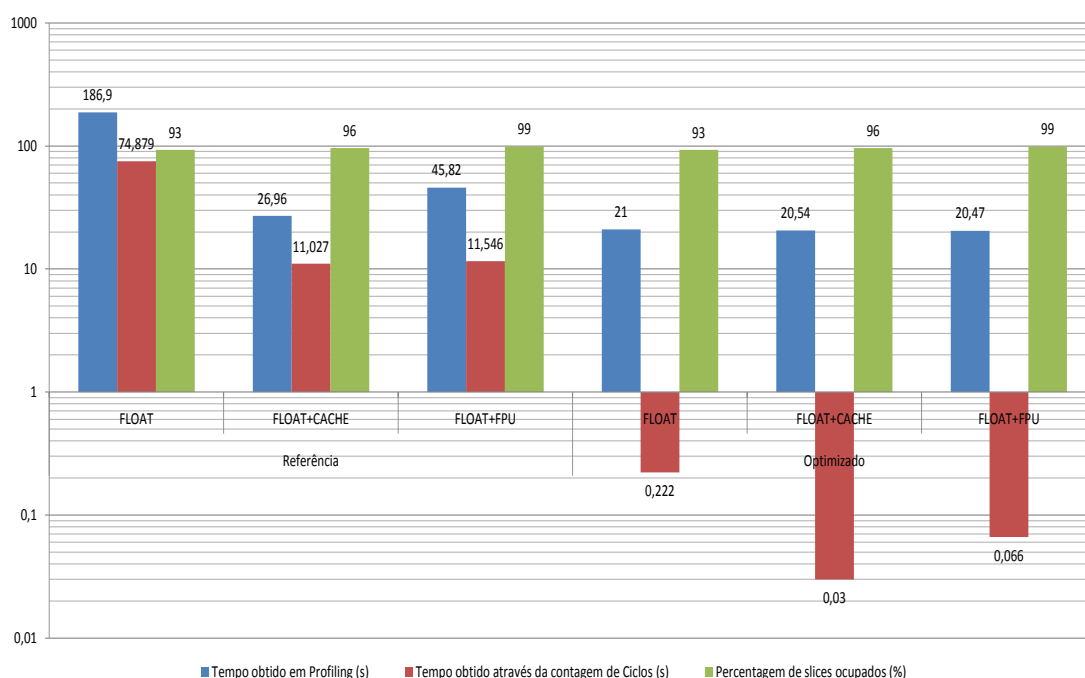


Figura 6.4: Comparação de resultados para soluções de hardware diversas e para os algoritmos apresentados.

Dos resultados observados até então, concluímos que a solução com memória *cache* era a opção que apresenta a melhor performance para o algoritmo em questão. Deste modo os resultados demonstrados a partir deste momento são baseados nesta solução.

A partir deste momento, como estabelecemos que a solução de arquitectura com memória *cache* a executar o algoritmo optimizado com dados do tipo *float* obtém uma velocidade de execução superior, efectuamos os testes com a inicialização da matriz e uma actualização da mesma tendo em conta um objecto à distância máxima permitida pelo SONAR (já não existem constrangimentos por parte do software).

A Figura 6.5 retrata a tendência dos tempos necessários em função da dimensão das grelhas usadas. Estes resultados são agora comparados entre os dois algoritmos de mapeamento referenciados: o de *Bayes* e o de *Dempster-Shafer*.

Observa-se que, à medida que a dimensão da grelha de ocupação aumenta, aumenta também exponencialmente o tempo necessário para completar a rotina, para ambos os algoritmos de mapeamento. O algoritmo de *Dempster-Shafer* tem no entanto uma razão de aumento de tempo superior ao de *Bayes*.

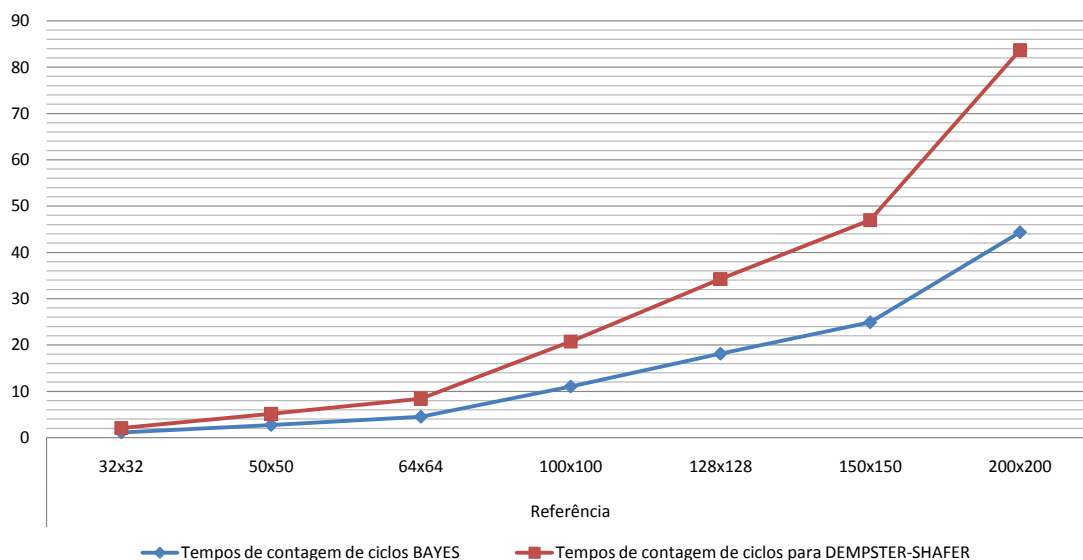


Figura 6.5: Tendência de crescimento do tempo de execução considerando dimensões da grelha crescentes.

Realizando a mesma comparação, desta vez com a versão otimizada de actualização na matriz, observa-se na Figura 6.6 que os resultados de tempo variam muito pouco com o aumento das dimensões da grelha. De facto para um aumento de 40 vezes o número de células, o tempo utilizado para a realização da rotina aumenta apenas cerca de *1ms* para cada caso. O algoritmo de *Dempster-Shafer* continua a necessitar de mais tempo para a actualização da matriz.

O aumento de *1ms* na execução da rotina leva-nos a considerar que este se deve ao tempo necessário para inicializar a matriz, visto que para o algoritmo otimizado a actualização da grelha só está relacionado com a distância ao objecto. Assim, realizamos a mesma contabilização de tempos medidos por temporizador apenas para a função de actualização da grelha. A inicialização da grelha não entra para os próximos tempos calculados.

Da Figura 6.7 observa-se que independentemente da dimensão da grelha o tempo de execução mantém-se quase inalterado para ambos os algoritmos de mapeamento, confirmando que o algoritmo otimizado não depende da dimensão da grelha. O algoritmo de mapeamento de *Bayes* continua a executar mais rapidamente que o algoritmo de *Dempster-Shafer*.

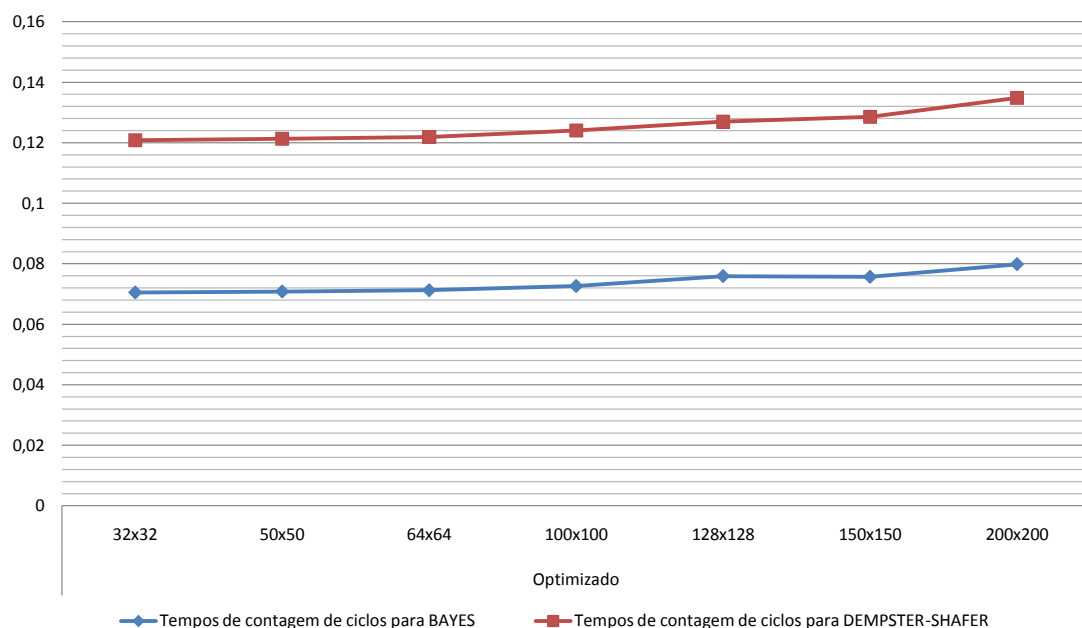


Figura 6.6: Tendência de crescimento do tempo de execução considerando dimensões da grelha crescentes para o algoritmo otimizado.

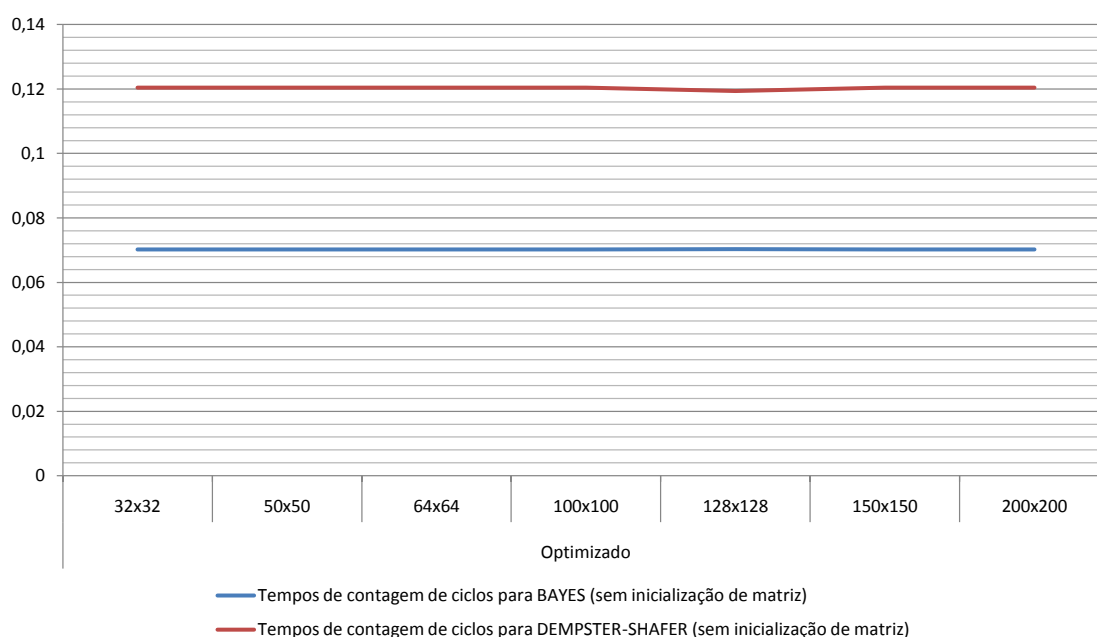


Figura 6.7: Tendência de crescimento do tempo de execução considerando dimensões da grelha crescentes apenas para a função de actualização do algoritmo otimizado.

Os dados obtidos até então indicam que o factor que influenciará o tempo de execução do algoritmo otimizado será a distância ao objecto. A Figura 6.8 apresenta graficamente os tempos

de execução para ambos os algoritmos de mapeamento em função da distância a um obstáculo. Consta-se que o número de células a calcular aumenta muito pouco até ao ponto em que a distância é 10 unidades. Notando-se neste caso um considerável aumento de tempo. O algoritmo de mapeamento de *Dempster-Shafer* continua a ser o mais exigente em termos computacionais, originando tempos de execução sempre superiores aos do algoritmo de *Bayes*.

Tendo em conta que a distância máxima admitida para uma leitura de SONAR do nosso protótipo é 10 unidades, o pior tempo de execução é 70ms para o algoritmo de *Bayes* e 120ms para o algoritmo de *Dempster-Shafer*.

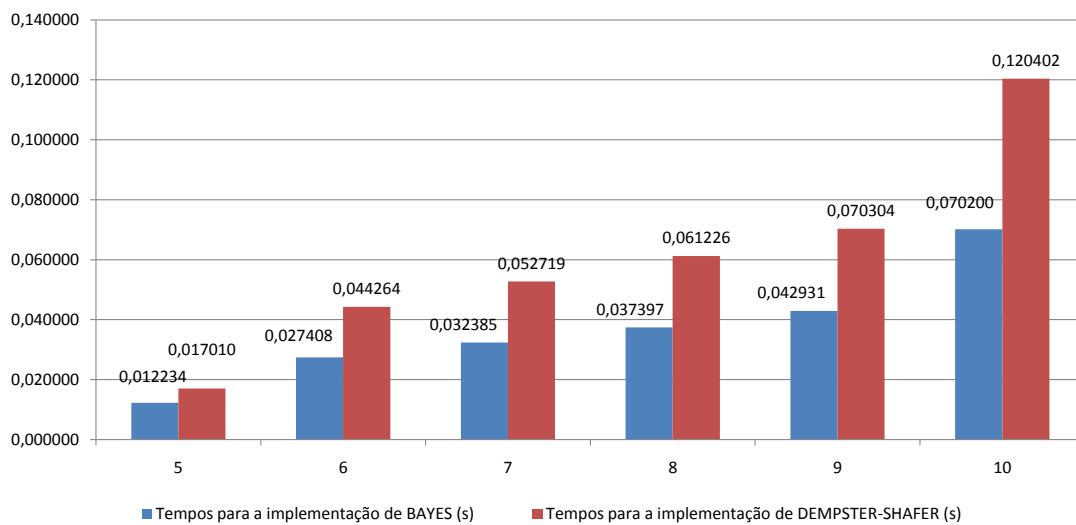


Figura 6.8: Tempos de execução da função de actualização em função da distância ao objecto.

6.4 Experiência no Campo

Após termos chegado à conclusão de que a arquitectura do sistema embebido que providencia os melhores tempos de execução inclui memória cache e dados do tipo *float* realizamos uma experiência no campo com o protótipo.

Construímos um ambiente com cerca de $1m^2$ de área e com obstáculos. O robô deveria desviar-se dos obstáculos e realizar o mapeamento do ambiente.

A grelha de ocupação do robô continha 30×30 unidades, o equivalente a $9m^2$. O robô considerava a posição inicial no centro do mapa (posição $x = 15$ e $y = 15$, neste exemplo).

Na Figura 6.9 pode-se observar o ambiente que o robô teria que percorrer e a linha a tracejado representa o caminho efectivamente percorrido. Verifica-se que o robô evitou os obstáculos realizando viragens à direita (estas viragens são aleatórias). O trajecto apresentado foi realizado pelo robô em menos de 10 segundos.

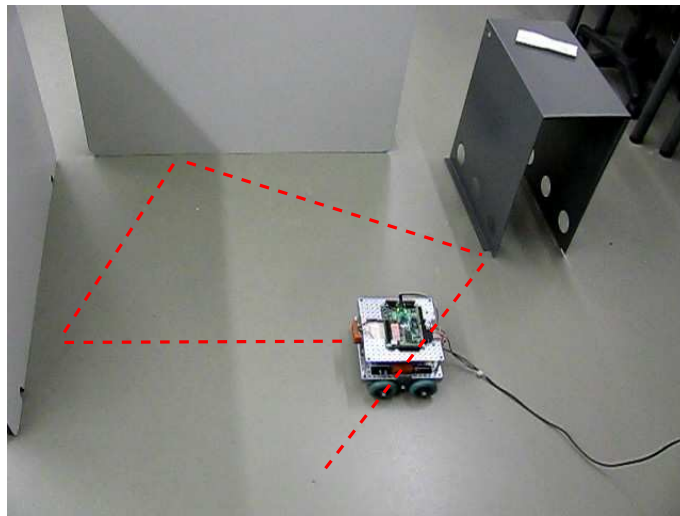


Figura 6.9: Foto do ambiente a mapear.

No fim desta experiência, a grelha de ocupação contruída pelo algoritmo é a representada pela Figura 6.10. Nesta Figura encontram-se também ilustradas as áreas consideradas como obstáculos e o provável percurso percorrido pelo robô.

Numa primeira observação é perceptível pelas células a cinzento (células com uma probabilidade de estar ocupadas superior a 0.50) os limites dos obstáculos presentes no ambiente. A indefinição representa a incerteza imposta pelo modelo do SONAR aplicado ao mapa. Na Região I as células são consideradas como provavelmente ocupadas e na Região II como provavelmente vazias. A fusão de várias leituras de uma mesma célula faz variar as probabilidades associadas a essa célula, daí a gama variada de valores de probabilidade na proximidade de um obstáculo.

Em suma, o protótipo realizou satisfatoriamente e com um tempo de execução curto a função de mapeamento.

6.5 Conclusões

As análises efectuadas com base no *profiling* dos algoritmos, permitiram elaborar uma linha de acção que determinasse o melhor tempo possível para a placa de FPGAs utilizada. Realizaram-se melhorias constantes que iniciaram na simplificação da precisão, alterando de precisão dupla para precisão simples, passando pela inclusão de módulos de hardware (memória cache e FPU) e terminando em alterações à rotina que actualiza a grelha.

Conseguiu-se uma diminuição de tempo de execução de cerca de 2 vezes na passagem da precisão dupla para precisão simples. Para o algoritmo de referência e dados do tipo *double* compilados com optimização O2 obteve-se 696,08s em contraste com 335,00s do algoritmo de referência e dados do tipo *float* compilados com optimização O2. Conseguiu-se ainda uma diminuição em cerca de 1,8 vezes quando comparando as melhores opções de optimização de compilação obtendo-se 335,00s compilados com optimização O2 em comparação com 186,90s com

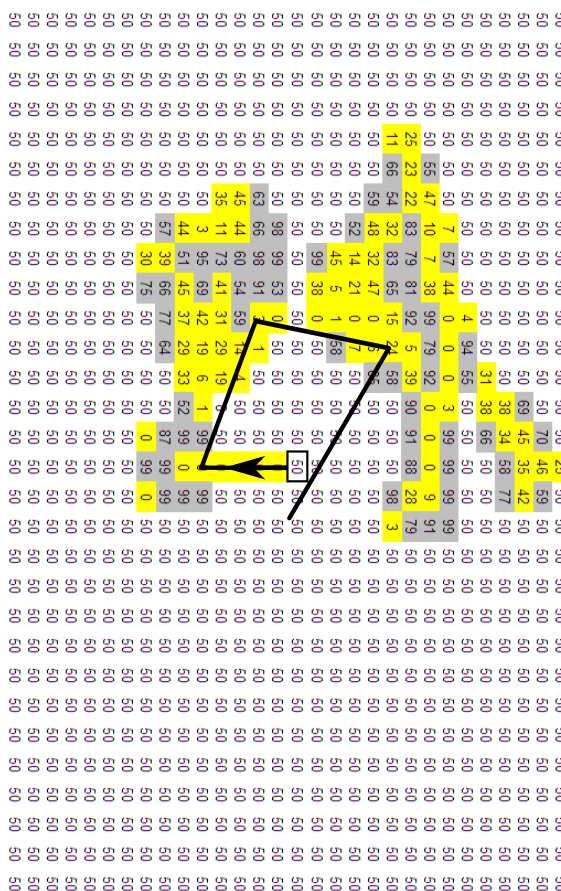


Figura 6.10: Probabilidade de ocupação para cada célula (amarelo - inferior a 50%) (cinzento - superior a 50%) da grelha de ocupação utilizada pelo robô.

compilação otimizada O3, ambos para dados do tipo *float*).

Globalmente o tempo de *profiling* para um objecto à distância média de 6 unidades melhorou em cerca de 28 vezes quando se optimizou o tipo de dados e acrescentou módulos de hardware, especificamente memória cache. Também para *profiling*, a execução do algoritmo optimizado na arquitectura com memória cache permitiu melhorar o tempo de execução obtido anteriormente para o algoritmo de referência com dados do tipo *double* em 36 vezes.

Tomando como exemplo a medição de tempos com o temporizador conseguiu-se uma redução em cerca de 2500 vezes o tempo de execução para um obstáculo à distância média de 6 unidades, obtendo-se 74,879s para o algoritmo de referência em contraste com os 30ms obtidos para o algoritmo otimizado executado num projecto com a inclusão de memória cache. Ambos os tempos referem-se a algoritmos que utilizam dados do tipo *float* e compilados com optimização O3.

As experiências efectuadas no terreno permitiram validar o funcionamento do algoritmo de mapeamento, assim como o funcionamento do protótipo.

Capítulo 7

Conclusões e Trabalho Futuro

Neste projecto foi implementada uma arquitectura que realiza a interacção de dois sistemas distintos (kit robótico e FPGA). Para testar o protótipo foi implementado um algoritmo de mapeamento tendo em vista a construção do mapa do ambiente explorado pelo robô. Foram estudadas alterações à arquitectura interna do sistema embebido, especificamente com a inclusão de módulos de hardware adicionais, para estudar o seu impacto no desempenho da execução do algoritmo em tempo real.

7.1 Satisfação dos Objectivos

Os objectivos traçados originalmente foram genericamente alcançados.

Implementamos e estudamos um algoritmo de mapeamento executado num processador RISC embebido num FPGA. Tivemos a oportunidade de estudar diferentes soluções optimizadas dessa arquitectura.

No FPGA encontra-se o algoritmo de controlo que recebe dados da Unidade de Controlo da VEX Robotics e com base nesses dados controla as movimentações do robô. Utiliza também esses dados para actualizar as probabilidades da grelha de ocupação.

Por restrições de tempo, implementamos um algoritmo reactivo para exploração do ambiente não utilizando a informação do mapa para tomar decisões.

7.2 Trabalho Futuro

A realização deste projecto deixa em aberto um conjunto de possibilidades para trabalho futuro.

O ponto actual deste projecto sugere optimizações do sistema. Especificamente, modificar o código fonte e a arquitectura do robô para aumentar a sua eficiência/eficácia, face aos objectivos

propostos. De facto, as optimizações ao código fonte utilizado deverão centrar-se em melhoramentos relacionados com o varrimento das células a serem actualizadas na grelha de ocupação, visto ter sido esta a etapa a causar constrangimentos na execução do algoritmo.

Outras funções no código fonte que apresentem algum congestionamento podem ser substituídas por módulos IP dedicados.

A sugestão de trabalho futuro no que toca à arquitectura pode ser a eliminação da componente de processamento de baixo nível do kit robótico substituindo-o por blocos dedicados que estabeleçam uma interface para controlarem os sensores e motores do kit.

Outro desafio poderia consistir na implementação de um processo funcional em que, com base no mapa obtido, o robô planeia o seu próximo movimento, como por exemplo, a exploração não aleatória (ao invés da movimentação reactiva utilizada neste projecto) e implementação de *path planning*.

A optimização do protótipo deve considerar abordagens para minimizar o erro de localização, utilizando algoritmos específicos para este efeito.

Referências

- [1] VEX robotics design system. <http://www.vexrobotics.com/>.
- [2] Mekatronix homepage. <http://www.mekatronix.com/>.
- [3] Intelligent robots for commercial & research applications. <http://www.mobilerobots.com/>.
- [4] LEGO.com MINDSTORMS NXT home. <http://mindstorms.lego.com/>.
- [5] *2004 Programming Reference Guide*.
- [6] *2008 VEX Inventor's Guide*.
- [7] *PIC18F6520/8520/6620/8620/6720/8720 Data Sheet*.
- [8] Katherine Compton e Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, 2002.
- [9] G. Rubin, M. Omieljanowicz, e A. Petrovsky. Reconfigurable FPGA-Based hardware accelerator for embedded DSP. Em *Mixed Design of Integrated Circuits and Systems, 2007. MIXDES '07. 14th International Conference on*, páginas 147–151, 2007.
- [10] Zoran Salcic. PROTOS— a microcontroller/FPGA-based prototyping system for embedded applications. *Microprocessors and Microsystems*, 21(4):249–256, Dezembro 1997.
- [11] D.F. Wolf, J.A. Holanda, V. Bonato, R. Peron, e E. Marques. An FPGA-Based mobile robot controller. Em *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, páginas 119–124, 2007.
- [12] Karlheinz Weiß, Thorsten Steckstor, Gernot Koch, e Wolfgang Rosenstiel. Exploiting FPGA-features during the emulation of a fast reactive embedded system. Em *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, páginas 235–242, Monterey, California, United States, 1999. ACM.
- [13] Yanbing Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, e J. Stockwood. Hardware-software co-design of embedded reconfigurable architectures. Em *Design Automation Conference, 2000. Proceedings 2000. 37th*, páginas 507–512, 2000.
- [14] P. Lysaght. FPGAs as meta-platforms for embedded systems. Em *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on*, páginas 7–12, 2002.
- [15] M. Borgatti, F. Lertora, B. Foret, e L. Cali. A reconfigurable system featuring dynamically extensible embedded microprocessor, FPGA and customisable I/O. Em *Custom Integrated Circuits Conference, 2002. Proceedings of the IEEE 2002*, páginas 13–16, 2002.

- [16] Sebastian Thrun. Robotic mapping: A survey. *Exploring Artificial Intelligence in the New Millenium*, 2002.
- [17] Robin R. Murphy. *An Introduction to AI Robotics*. The MIT Press, 1st edição, Novembro 2000.
- [18] Sebastian Thrun, Wolfram Burgard, e Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [19] NIST national institute of standards and technology: Time and frequency division.
- [20] Scott Hauck. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, 1 edição, Dezembro 2007.
- [21] Xilinx. <http://www.xilinx.com/>.
- [22] *Spartan-3 FPGA Starter Kit Board User Guide*.
- [23] *EDK Concepts, Tools, and Techniques—A Hands-on Guide to Effective Embedded System Design*.
- [24] *Microblaze Processor Reference Guide*.
- [25] VEX robotics motor kit. <http://www.vexrobotics.com/vex-robotics-motor-kit.shtml>.
- [26] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture.
- [27] Adam Milstein. *Motion Planning*, chapter Occupancy Grid Maps for Localization and Mapping. InTech Education and Publishing, First edição, 2008.
- [28] F. Mayer-Lindenberg. Design and application of a scalable embedded systems' architecture with an FPGA based operating infrastructure. Em *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, páginas 189–196, 2006.
- [29] Matthew Charles Deans. *Bearings-only localization and mapping*. Tese de doutoramento, Carnegie Mellon University, 2005. Ph.D.
- [30] U. Reinsalu, A. Arhipov, e P. Ellervee. Architectural exploration tasks for on-chip embedded systems. Em *Electronics Conference, 2008. BEC 2008. 11th International Biennial Baltic*, páginas 171–174, 2008.
- [31] Kristopher R. Beevers. *Mapping with limited sensing*. Tese de doutoramento, Rensselaer Polytechnic Institute, 2007. Ph.D.
- [32] M. Gschwind, V. Salapura, e D. Maurer. FPGA prototyping of a RISC processor core for embedded applications. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(2):241–250, 2001.
- [33] I. Gonzalez, E. Aguayo, e S. Lopez-Buedo. Self-Reconfigurable embedded systems on Low-Cost FPGAs. *Micro, IEEE*, 27(4):49–57, 2007.
- [34] Roland Siegwart e Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, Abril 2004.

- [35] Scott Hauck. The future of reconfigurable systems. in *5th Canadian Conference on Field Programmable Devices*, 1998.
- [36] Andre' DeHon. Reconfigurable architectures for General-Purpose computing. Relatório té AITR-1586, 1996.
- [37] S. Sanchez-Solano, A.J. Cabrera, I. Baturone, F.J. Moreno-Velo, e M. Brox. FPGA implementation of embedded fuzzy controllers for robotic applications. *Industrial Electronics, IEEE Transactions on*, 54(4):1937–1945, 2007.
- [38] Momotaz Begum. *Robotic mapping using soft computing methodologies*. Tese de doutoramento, Memorial University of Newfoundland (Canada), 2005. M.Eng.
- [39] J.J. Rodriguez-Andina, M.J. Moure, e M.D. Valdes. Features, design tools, and application domains of FPGAs. *Industrial Electronics, IEEE Transactions on*, 54(4):1810–1823, 2007.
- [40] Thomas Braunl. *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Springer, 1 edição, Agosto 2004.
- [41] Tun Yang e V. Aitken. Evidential mapping for mobile robots with range sensors. *Instrumentation and Measurement, IEEE Transactions on*, 55(4):1422–1429, 2006.
- [42] Scott Alan Hauck. *Multi-FPGA systems*. Tese de doutoramento, University of Washington, 1995. Ph.D.
- [43] S.A. Ito e L. Carro. A comparison of microcontrollers targeted to FPGA-based embedded applications. Em *Integrated Circuits and Systems Design, 2000. Proceedings. 13th Symposium on*, páginas 397–402, 2000.
- [44] Jason G. Tong. *Software profiling for an FPGA-based CPU core*. Tese de doutoramento, University of Windsor (Canada), 2007. M.A.Sc.
- [45] Gregory Dudek e Michael Jenkin. *Computational principles of mobile robotics*. Cambridge University Press, New York, NY, USA, 2000.
- [46] Ulrich Nehmzow. *Mobile Robotics: A Practical Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

Anexo A

Especificações dos kits robóticos

Neste apêndice estão patentes as características associadas a cada kit robótico considerado como candidato a ser utilizado neste projecto. Estas informações foram transcritas de documentação e consulta dos respectivos sites.

A.1 VEX Robotics

“The VEX Protobot Robot Kit contains over 300 parts and pieces which can be used to build the VEX Protobot, the VEX Tumbler, or some creation of your own! ” [1]

Tabela A.1: Specifications comparison table according to VEX Robotics website [1].

<i>Characteristics</i>	VEX Robotics Features
	Protobot Robot Kit
Price	Under \$1000.00 with full add ons
Support Structure	Square
Batteries	7.2V Rechargeable NiCad batteries. Capacity 1000 2000 mAh
Microcontroller	Microchip PICmicro® PIC18F8520 10 MIPS
Programming Software	easyC Versions from \$99.00 to \$149.00
I/O Pins	6 Interrupt inputs, 16 Digital I/O with no analog and vice versa
Inclusion of extra modules	Yes
Extra modules for the kit	Modules for motion, power, structure, sensors, logic and control
Weight	Under 2,5Kg
Dimmensions	Drive Base Width 12.375 in. Drive Base Length 9.125 in.
Construction materials	Aluminum and plastic
Robustness	Reasonable
Flexibility	Good

A.2 Mekatronix

“TJ Pro™ offers a sophisticated, expandable, programmable, autonomous, mobile robot with features found on units costing twice as much. This robot uses the same

body as the TJ robot. TJ Pro™ is available in an aircraft plywood or black ABS plastic body. It rides on two 2.75 inch Du-Bro wheels and a rear skid driven by a pair of servos, hacked as gearhead motors.” [2]

Tabela A.2: Specifications comparison table according to Mekatronix website [2].

<i>Characteristics</i>	Mekatronix Features
	Talrik JR Pro
Price	Around \$1000.00 with full add ons
Support Structure	Cylindrical
Batteries	Six AA rechargeable NiCad (600 ma-hr), 5.4-7.2 volts
Microcontroller	MC68HC11 microcontroller
Programming Software	MC68HC11 Assembly Language, C, or BASIC
I/O Pins	Eight digital outputs. Three digital input ports
Inclusion of extra modules	Yes
Extra modules for the kit	Modules for Control, Sensors and Power
Weight	Around 5Kg
Dimensions	7 inch diameter and 3 1/2 inch height
Construction materials	ABS Plastic
Robustness	Good
Flexibility	Bad

A.3 Mobile Robots

“PIONEER 3-AT is a highly versatile all-terrain robotic platform, software-compatible with all MobileRobots robots, chosen by many DARPA grantees and others requiring a high-performance robot with plenty of real estate for customization. Powerful, yet easy to use; reliable, yet flexible, P3-AT is a popular team performer for outdoor or rough-terrain projects.” [3].

A.4 Lego Mindstorm NXT

“TriBot is a flexible and fast 3-wheeled driving robot! It takes advantage of all four sensors to do the job you program it to do. TriBot can grab a ball when you give it a sound command (through its Sound Sensor), can be programmed to follow a line with its Light Sensor , can feel objects with its Touch Sensor , and can see with its Ultrasonic Sensor.” [4]

Tabela A.3: Specifications comparison table according to Mobile Robots website [3].

Mobile Robots Features	
<i>Characteristics</i>	Pioneer 3-AT
Price	More than €9000.00
Support Structure	Cylindrical
Batteries	Three 12V sealed, lead-acid batteries
Microcontroller	Hitachi H8S
Programming Software	ARIA Robotics API
I/O Pins	8-bit external I/O bus w/ up to 16 devices
Inclusion of extra modules	Yes
Extra modules for the kit	Modules for Control, Sensors and Power
Weight	Around 12Kg
Dimensions	50cm x 49cm x 26cm
Construction materials	Aluminum
Robustness	Very Good
Flexibility	Bad

Tabela A.4: Specifications comparison table according to Lego Mindstorm NXT website [4].

Lego Mindstorm NXT Features	
<i>Characteristics</i>	Tribot
Price	Around €200.00
Support Structure	Unstructured
Batteries	6 AA batteries
Microcontroller	32-bit ARM7 microcontroller
Programming Software	Lego Mindstorm NXT software
I/O Pins	4 input ports. 3 output ports
Inclusion of extra modules	Difficult
Extra modules for the kit	Modules for Control, Sensors and Power
Weight	Around 5 Pounds
Dimensions	4.6 x 15 x 15.1 inches
Construction materials	Plastic
Robustness	Bad
Flexibility	Good

Anexo B

Protocolo Simples de Transmissão Paralela

O Protocolo Simples de Transmissão Paralela (PSTP) foi desenvolvido no decurso do projecto com vista a estabelecer uma comunicação rápida entre as portas de saída do FPGA e a Unidade de Controlo (UC) da VEX Robotics.

Cada porto do PIC18F8520 tem três registos que definem o seu modo de funcionamento, como explicado no seu manual [7]

- Registo TRIS: que define a direcção dos dados;
- Registo PORT: permite a leitura dos níveis dos pinos;
- Registo LAT: permite operações de leitura/escrita de valores considerando os pinos de E/S.

A operação genérica de um porto de E/S é exemplificada pela Figura B.1

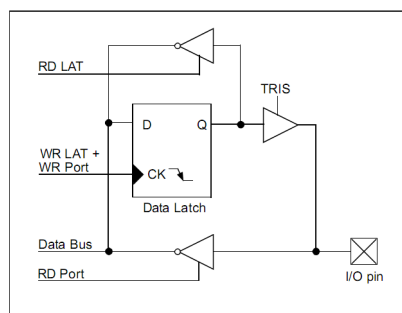


Figura B.1: Port de E/S genérica do PIC18F8520 [7].

A UC VEX tem 16 pinos de E/S disponíveis. Para o objectivo desta comunicação, a Figura B.2 representa os pinos conectados e a forma como são utilizados.

A comunicação é iniciada pela CPU que se encontra no FPGA (no nosso caso o Microblaze ou o PowerPC) e espera até que a UC VEX fique disponível. A par deste pedido é enviado um

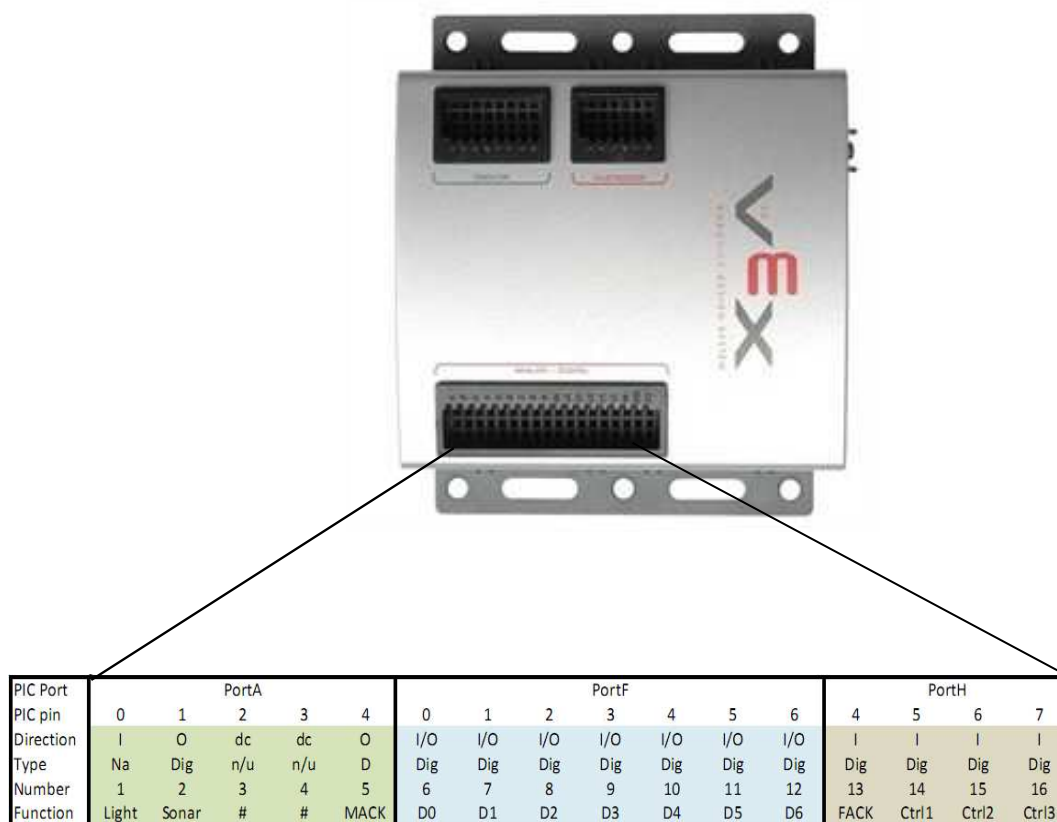


Figura B.2: Atribuição de funções e conexões da UC VEX

sinal de controlo com o comando da informação desejada. Os comandos existentes encontram-se sumariados na Tabela B.1.

Para a realização desta tarefa existe a necessidade de um *level-shifter* para transformar a saída de 5V da UC VEX para os 3.3 V requeridos pelas entradas do FPGA.

Se o Processador do FPGA necessitar de enviar comandos para os motores, então os valores presentes nos pinos de controlo alertarão a UC VEX que deve alterar o estado dos seus pinos de entrada para saída. Um ciclo de escrita é então iniciado.

Ambos os casos são apresentados no seguinte diagrama de comunicação (Figura B.3).

Tabela B.1: Comandos enviados pelo Processador do FPGA para a UC VEX

<i>Codificação</i>	Comandos Comando
000	Terminar conexão
001	Dados do SONAR
010	Dados do codificador de impulsos esquerdo
011	Dados do codificador de impulsos direito
100	Dados do sensor de luz
101	<i>Disponível</i>
110	<i>Disponível</i>
111	Notifica que quer enviar comandos

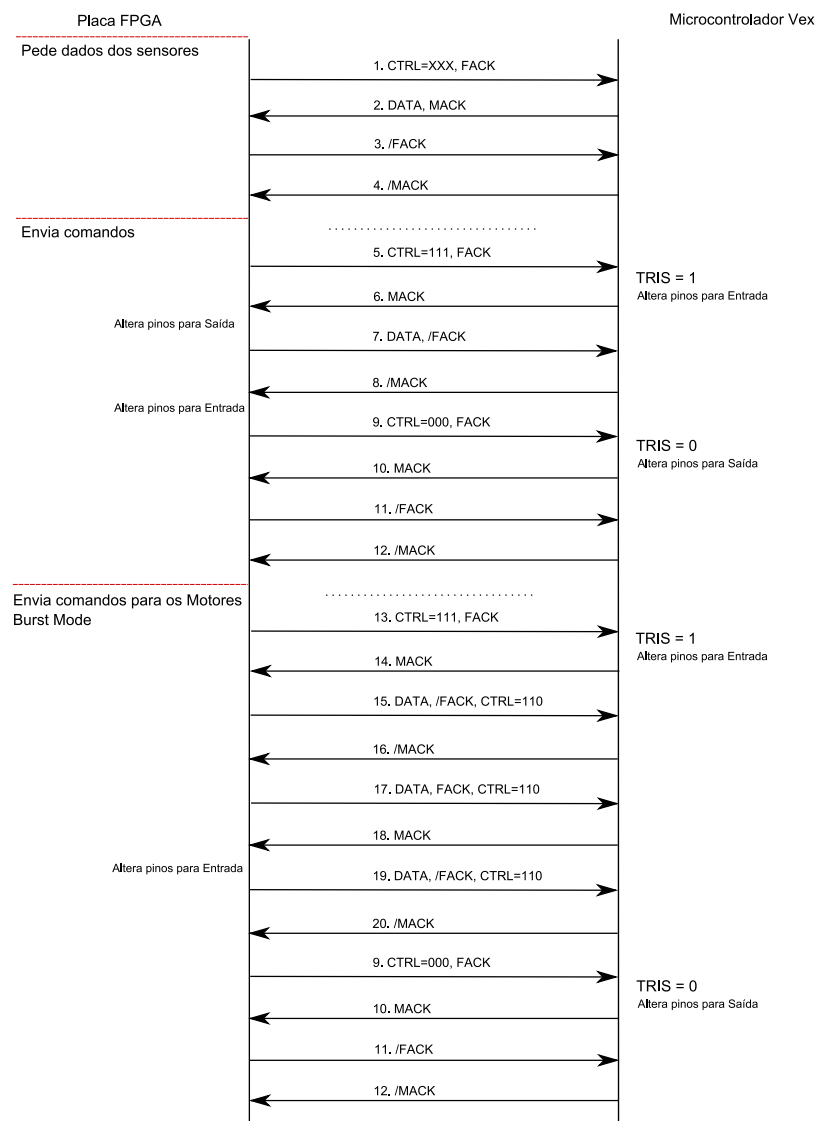


Figura B.3: Diagrama de Comunicação PSTP

Anexo C

Protocolo de Transmissão de 4 bits

O Protocolo de Transmissão de 4 bits (PT4B) foi desenvolvido como uma alternativa ao PSTP. Tem vantagem sobre este, pelo facto de ser mais fácil de implementar. Com apenas umas ligações directas com fios condutores pode-se estabelecer uma transmissão funcional.

Podem-se utilizar quaisquer portos de *General Purpose Input Output* (GPIO) do FPGA. O diagrama de conexão à unidade de controlo (UC) da VEX Robotics é apresentado na Figura C.1.

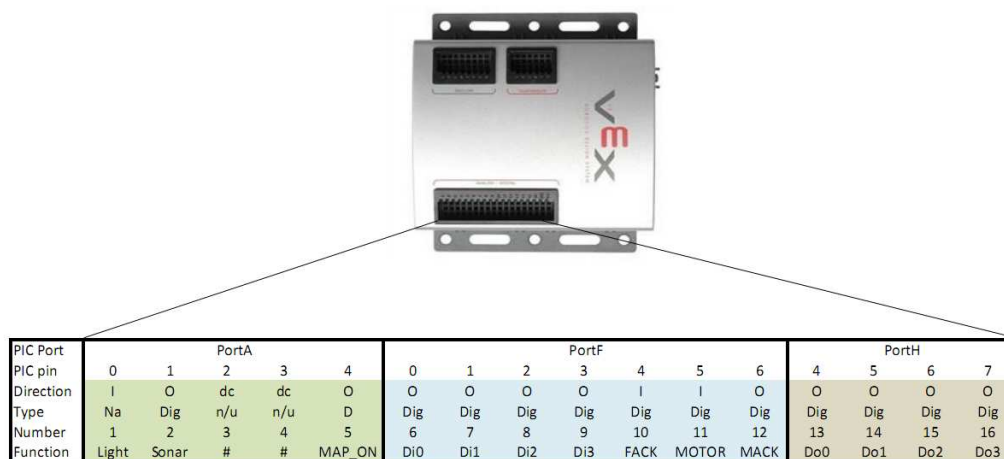


Figura C.1: Atribuição de funções e conexões da UC VEX.

Neste protocolo o FPGA inicia o pedido de comunicação, enviando um comando *non-encoding* identificando os sensores para os quais necessita de efectuar leituras. Mostra-se na Tabela B.1 a informação enviada pelo processador do FPGA.

Mostra-se na Tabela C.1 a informação enviada pelo processador do FPGA.

Os dados solicitados são enviados pela UC VEX, 4 bits de cada vez. Após a transmissão de dados deve haver a confirmação de recepção por parte do CPU do FPGA. Após a conclusão da transmissão, o FPGA inicia o envio de comandos para os motores, usando o pino MOTOR para indicar o sentido de rotação do motor. Um diagrama de comunicação é apresentado na Figura C.2.

Tabela C.1: Comandos *non-encoding* do CPU do FPGA para a UC VEX.

<i>Codificação</i>	Comandos Comando
0001	Dados do Sonar
0010	Dados do codificador de impulsos esquerdo
0100	Dados do codificador de impulsos direito
1000	Dados do sensor de luz

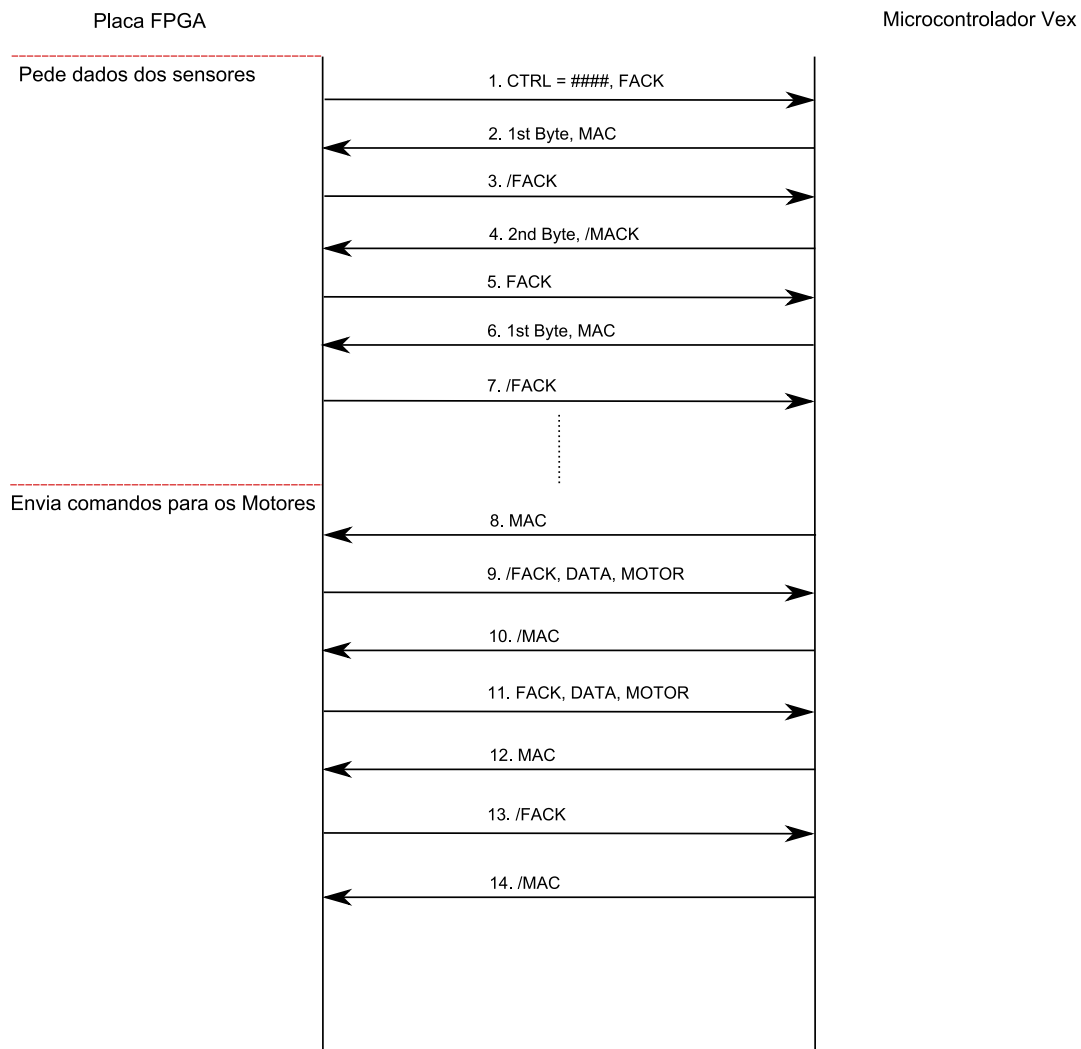


Figura C.2: Diagrama de Comunicação PT4B.

Anexo D

Código Fonte de Teste

Neste Apêndice é apresentado o código fonte do algoritmo de mapeamento utilizado no protótipo e no estudo de optimização da arquitectura do FPGA. Neste algoritmo existe a possibilidade mediante uma configuração inicial de seleccionar qual a teoria de mapeamento a utilizar e definir o tipo de dados, assim como escolher entre o algoritmo de referência ou o algoritmo optimizado.

```

/*****
 * File:    teste.c
 * Author:  Manuel Reis 010503078 FEUP PORTUGAL
 *
 * Created on 8 de Junho de 2009, 3:37
 *****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*****
                                ALIAS DEFINITIONS
 *****/

#define PI          3.14
#define RADIUS      1.2    //dm
#define TICK_COUNT  90     //90 ticks = 1 sensor revolution
#define ROWS        100    /* Number of rows in the super-grid */
#define COLUMNS    100    /* Number of columns in the super-grid */
#define MAXOCCUPIED 0.98   /* Expresses the assumption that a reading
                             of occupied is never fully believable */
#define TOLERANCE   1      /* In dm. Often, the resolution of the sonar.
                             It has the impact of making Region I wider,
                             thereby covering more grid elements. */

#define BETA        PI/12  /* The half angle representing the width of the cone */
#define RANGE       10     /* In dm. Maximum range detected. */
#define MAXSTEP     1      /* Size of the cell in dm */
#define PROB_OCC    0.5    /* A PRIORI probability - OCCUPIED */
#define PROB_EMP    0.5    /* A PRIORI probability - EMPTY */
#define APRIORIPROB 0.5

```

```

/*****
CONFIGURATION PARAMETERS
*****/
#define PRECISION 0      /* Choose Precision (0 for float) (1 for double) */
#define OPTIMIZED 0      /* Choose Optimized (0 for None) (1 for yes) */
#define GRID_TYPE 0      /* Choose type of struct (0 for BAYES)
                        * (1 for BELIEF) */

#if PRECISION
    typedef double      datatype;
    #define power      pow
    #define cosine      cos
    #define sine        sin
    #define arcs        acos
    #define roundtoint  rint
    #define squareroot  sqrt

#else
    typedef float       datatype;
    #define power      powf
    #define cosine      cosf
    #define sine        sinf
    #define arcs        acosf
    #define roundtoint  rintf
    #define squareroot  sqrtf
#endif      /* Precision for the calculations of probabilities */

typedef struct
{
    datatype occupied;
    datatype empty;
    #if GRID_TYPE
        datatype dontknow;
    #endif
} GRID;

typedef struct
{
    int x;
    int y;
} point;

typedef struct
{
    int i;
    int j;
} vector;

GRID super_grid[ROWS][COLUMNS];

```

```

/*****
                                FUNCTIONS PROTOTYPES
*****/
datatype return_distance(int x1,int y1, int x2, int y2);
datatype return_y(datatype distance, datatype angle);
datatype return_x(datatype distance, datatype angle);
datatype calc_norma(vector vector);
int internal_product(vector vector1, vector vector2);
vector find_vector(int x0,int y0,int x1,int y1);

void compute_sonar_matrix(int x, int y, datatype s, datatype angle);
void matrix_region(int x, int y, datatype r, datatype angle, int obstacle);

#if !GRID_TYPE
    void init_grid_bayes(void);
    void printf_out_grid_bayes(void);
    datatype bayes_region_I(datatype s, datatype alpha);
    datatype bayes_region_II(datatype s, datatype alpha);
    datatype bayes_update(datatype Ps1_0,datatype PO_s0 );
#else
    void init_grid_belief(void);
    void printf_out_grid_belief(void);
    GRID belief_update(GRID prob, int i, int j);
    datatype belief_region_I(datatype s, datatype alpha);
    datatype belief_region_II(datatype s, datatype alpha);
#endif

/*****
* FUNCTION NAME: main
* PURPOSE:      main function
*****/

int main(int argc, char** argv) {

    int i = 0;

    #if !GRID_TYPE
        // BAYES
        for(i = 0; i < 1; i++)
        {
            init_grid_bayes();
            compute_sonar_matrix(3,3,10,((0*PI)/2));
        }
    #else
        // DEMPSTER-SHAFER
        for(i = 0; i < 1; i++)
        {
            init_grid_belief();

```

```

        compute_sonar_matrix(3,3,1,((0*PI)/2));
    }
#endif

    exit(0);
}

/*****
* FUNCTION NAME: return_distance
* PURPOSE:      returns distance between two points
*****/
datatype return_distance(int x1,int y1, int x2, int y2)
{
    datatype distance = squareroot(power((x2-x1),2)+power((y2-y1),2));
    return distance;
}

/*****
* FUNCTION NAME: compute_sonar_matrix
* PURPOSE:      Calls function to compute probabilities. Indicates there was
*               obstacle
*****/
void compute_sonar_matrix(int x, int y, datatype s, datatype angle)
{
    if (( s < (RANGE+TOLERANCE) ) && (s > 0))
    {
        matrix_region(x, y, s, angle,1);
    }
}

/*****
* FUNCTION NAME: calc_norma
* PURPOSE:      Calculates the norm of a vector
*****/
datatype calc_norma(vector vector)
{
    datatype result = squareroot(power(vector.i,2) + power(vector.j,2));
    return result;
}

/*****
* FUNCTION NAME: internal_product
* PURPOSE:      Calculates the internal product between two vectors
*****/
int internal_product(vector vector1, vector vector2)
{
    int result;
    result = vector1.i*vector2.i + vector1.j*vector2.j;
}

```

```

    return result;
}

/*****
* FUNCTION NAME: find_vector
* PURPOSE:      Given two points, calculates the vector
*****/
vector find_vector(int x0,int y0,int x1,int y1)
{
    vector vector;
    vector.i = x1-x0;
    vector.j = y1-y0;
    return vector;
}

/*****
* FUNCTION NAME: matrix_region
* PURPOSE:      updates probabilities in the occupancy grid
*****/
void matrix_region(int x, int y, datatype r, datatype angle, int obstacle)
{
    vector center,tempvector;
    int i,j,tempint = 0;
    datatype normal1,norma2,result,newangle,newdistance;
    datatype Ps_O = 0, Prob = 0;

    #if GRID_TYPE == 1
        GRID probability;
    #endif

    long int count = 0;

    center.i = roundtoint(return_x(r,angle));
    center.j = roundtoint(return_y(r,angle));
    newdistance = return_distance(x,y,(x + center.i),(y + center.j));
    #if OPTIMIZED
        point min, max, temp;
        min.x = max.x = x;
        min.y = max.y = y;

        newangle = - (BETA);

        for(i = 0; i < 3; i++)
        {
            temp.x = x + roundtoint(return_x(r,(newangle+ angle)));
            temp.y = y + roundtoint(return_y(r,(newangle+ angle)));

            if (temp.x < 0)
                temp.x = 0;

```

```

    else if (temp.x > COLUMNS)
        temp.x = COLUMNS;

    if (temp.y < 0)
        temp.y = 0;
    else if (temp.y > COLUMNS)
        temp.y = COLUMNS;

    if (temp.x < min.x)
    {
        min.x = temp.x;
    }
    else if (temp.x > max.x)
    {
        max.x = temp.x;
    }

    if (temp.y < min.y)
    {
        min.y = temp.y;
    }
    else if (temp.y > max.y)
    {
        max.y = temp.y;
    }

    newangle += (BETA);
}

for (j=min.y; j<max.y; j++) // Y
{
    for(i=min.x; i<max.x; i++) // X
    {
#else
        for (j=0; j<ROWS; j++) // Y
        {
            for(i=0; i<COLUMNS; i++) // X
            {
#endif

#if !GRID_TYPE
                if ((j == y)&&( i == x))
                {
                    result = return_distance(x,y,i,j);
                    if (result <= newdistance)

```

```

        {
            Ps_O = bayes_region_II(result, 0);
            Prob = bayes_update(Ps_O, super_grid[j][i].occupied);
            super_grid[j][i].empty = 1 - Prob;
            super_grid[j][i].occupied = Prob;
        }
    }
    else
    {
        tempvector = find_vector(x,y,i,j);
        tempint = internal_product(center,tempvector);
        normal = calc_norma(tempvector);
        norma2 = calc_norma(center);
        result = tempint / (normal*norma2);
        newangle = arcos(result);
        if ((newangle <= (BETA)))
        {
            result = return_distance(x,y,i,j);
            if (obstacle)
            {
                if ((result >= newdistance - TOLERANCE)&&
                    (result <= newdistance + TOLERANCE))
                {
                    Ps_O = bayes_region_I(result, newangle);
                    Prob = bayes_update(Ps_O, super_grid[j][i].occupied);
                    super_grid[j][i].empty = 1 - Prob;
                    super_grid[j][i].occupied = Prob;
                }
                else if (result < newdistance - TOLERANCE)
                {
                    Ps_O = bayes_region_II(result, newangle);
                    Prob = bayes_update(Ps_O, super_grid[j][i].occupied);
                    super_grid[j][i].empty = 1 - Prob;
                    super_grid[j][i].occupied = Prob;
                }
            }
            else if (result < newdistance)
            {
                Ps_O = bayes_region_II(result, newangle);
                Prob = bayes_update(Ps_O, super_grid[j][i].occupied);
                super_grid[j][i].empty = 1 - Prob;
                super_grid[j][i].occupied = Prob;
            }
        }
    }
}

#else

if ((j == y)&&( i == x))
{

```

```

        result = return_distance(x,y,i,j);
        if (result <= newdistance)
        {

        }
    }
else
{
    tempvector = find_vector(x,y,i,j);
    tempint = internal_product(center,tempvector);
    normal = calc_norma(tempvector);
    norma2 = calc_norma(center);
    result = tempint / (normal*norma2);
    newangle = acos(result);
    if ((newangle <= (BETA)))
    {
        result = return_distance(x,y,i,j);
        if (obstacle)
        {
            if ((result >= newdistance - TOLERANCE)&&
                (result <= newdistance + TOLERANCE))
            {
                probability.occupied = belief_region_I(result,newangle);
                probability.empty = 0;
                probability.dontknow = 1 - probability.occupied;
                probability = belief_update(probability,i,j);
                super_grid[j][i] = probability;
            }
            else if (result < newdistance - TOLERANCE)
            {
                probability.occupied = 0;
                probability.empty = belief_region_II(result,newangle);
                probability.dontknow = 1 - probability.empty;
                probability = belief_update(probability,i,j);
                super_grid[j][i] = probability;
            }
        }
        else if (result < newdistance)
        {
            probability.occupied = 0;
            probability.empty = belief_region_II(result,newangle);
            probability.dontknow = 1 - probability.empty;
            probability = belief_update(probability,i,j);
            super_grid[j][i] = probability;
        }
    }
}

```

```

#endif

    }

}

}

/*****
* FUNCTION NAME: return_y
* PURPOSE:      returns the value of y
*****/
datatype return_y(datatype distance, datatype angle)
{
    datatype result = distance * cosine(angle);
    return result;
}

/*****
* FUNCTION NAME: return_x
* PURPOSE:      returns the value of x
*****/
datatype return_x(datatype distance, datatype angle)
{
    datatype result = distance * sine(angle) ;
    return result;
}

#if !GRID_TYPE

/*****
* FUNCTION NAME: init_grid_bayes
* PURPOSE:      initializes the occupancy grid
*****/

void init_grid_bayes(void)
{
    for (y=0;y<ROWS;y++)
    {
        for (x=0;x<COLUMNS;x++)
        {
            super_grid[y][x].occupied = APRIORIPROB;
            super_grid[y][x].empty = APRIORIPROB;
        }
    }
}

/*****
* FUNCTION NAME: printf_out_grid_bayes
* PURPOSE:      prints the grid probabilities to the STDOUT
*****/

```

```

*****/
void printf_out_grid_bayes(void)
{
    for (y=0;y<ROWS;y++)
    {
        for (x=0;x<COLUMNS;x++)
        {
            printf("_%d_", (int) (super_grid[y][x].occupied * 100));
        }
        printf("\n");
    }
}

/*****
* FUNCTION NAME: bayes_region_I
* PURPOSE:      calculate probabitlity if cell is in Region I
*****/
datatype bayes_region_I(datatype s, datatype alpha)
{
    datatype Ps_O;
    datatype beta = BETA;
    Ps_O = (((RANGE - s)/RANGE)+((beta-alpha)/beta))/2)*MAXOCCUPIED;
    return Ps_O;
}

/*****
* FUNCTION NAME: bayes_region_II
* PURPOSE:      calculate probabitlity if cell is in Region II
*****/
datatype bayes_region_II(datatype s, datatype alpha)
{
    datatype Ps_E;
    datatype beta = BETA;
    Ps_E = (((RANGE - s)/RANGE)+((beta-alpha)/beta))/2);
    return (1-Ps_E);
}

/*****
* FUNCTION NAME: bayes_update
* PURPOSE:      Combines the current probability with the previous
*****/
datatype bayes_update(datatype Ps1_O,datatype PO_s0 )
{
    datatype PO_s1;
    PO_s1 = (Ps1_O*PO_s0)/((Ps1_O*PO_s0)+((1-Ps1_O)*(1-PO_s0)));
    return PO_s1;
}

```

```
#else
```

```

/*****
* FUNCTION NAME: init_grid_belief
* PURPOSE:      initializes the occupancy grid
*****/

```

```
void init_grid_belief(void)
```

```

{
    for (y=0;y<ROWS;y++)
    {
        for (x=0;x<COLUMNS;x++)
        {
            super_grid[y][x].occupied = 0;
            super_grid[y][x].empty = 0;
            super_grid[y][x].dontknow = 1;
        }
    }
}

```

```

/*****
* FUNCTION NAME: printf_out_grid_belief
* PURPOSE:      prints the grid probabilities to the STDOUT
*****/

```

```
void printf_out_grid_belief(void)
```

```

{
    for (y=0;y<ROWS;y++)
    {
        for (x=0;x<COLUMNS;x++)
        {
            printf("_%.2lf_", super_grid[y][x].occupied);
        }
        printf("\n");
    }
}

```

```

/*****
* FUNCTION NAME: bayes_region_I
* PURPOSE:      calculate probabitlity if cell is in Region I
*****/

```

```
datatype belief_region_I(datatype s, datatype alpha)
```

```

{
    datatype Ps_O, beta = BETA;
    Ps_O = (((RANGE - s)/RANGE) + ((beta-alpha)/beta))/2)*MAXOCCUPIED;
    return Ps_O;
}

```

```

/*****
* FUNCTION NAME: belief_region_II

```

```

* PURPOSE:          calculate probabitlity if cell is in Region II
*****/
datatype belief_region_II(datatype s, datatype alpha)
{
    datatype Ps_E, beta = BETA;
    Ps_E = (((RANGE - s)/RANGE)+((beta-alpha)/beta))/2);
    return Ps_E;
}

/*****
* FUNCTION NAME: belief_update
* PURPOSE:        Combines the current probability with the previous
*****/
GRID belief_update(GRID prob, int i, int j)
{
    GRID probfinal;
    probfinal.occupied = ((prob.occupied * super_grid[j][i].occupied)+
        (prob.occupied * super_grid[j][i].dontknow)+
        (super_grid[j][i].occupied * prob.dontknow))/
        (1 - ((prob.occupied * super_grid[j][i].empty)+
        (prob.empty * super_grid[j][i].occupied)));

    probfinal.empty = ((prob.empty * super_grid[j][i].empty)+
        (prob.empty * super_grid[j][i].dontknow)+
        (super_grid[j][i].empty * prob.dontknow))/
        (1 - ((prob.occupied * super_grid[j][i].empty)+
        (prob.empty * super_grid[j][i].occupied)));

    probfinal.dontknow = (prob.dontknow * super_grid[j][i].dontknow)/
        (1 - ((prob.occupied * super_grid[j][i].empty)+
        (prob.empty * super_grid[j][i].occupied)));
    return probfinal;
}

#endif

```

Anexo E

Resultados de Profiling

Neste apêndice encontram-se as tabelas que contêm os resultados de *profiling* testados no PC e no sistema embestado para vários níveis de otimização.

E.1 Execução em PC

Resultados para tipos de dados Double - Otimização O0						Função
Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (us/chamada)	Tempo total da função (us/chamada)	
43,14	2,64	2,64	10000	264,00	494,50	matrix_region
24,51	4,14	1,50	199980000	0,01	0,01	calc_norma
10,46	4,78	0,64	10000	64,00	64,00	init_grid_bayes
7,35	5,23	0,45	99990000	0,00	0,00	internal_product
4,90	5,53	0,30	...	...	...	pow
3,92	5,77	0,24	99990000	0,00	0,00	find_vector
2,45	5,92	0,15	...	...	...	sqrt
1,39	6,00	0,09	15640000	0,01	0,01	return_distance
1,31	6,08	0,08	...	...	...	acos
0,49	6,12	0,03	70000	0,43	0,43	bayes_region_II
0,08	6,12	0,01	...	...	...	main
0,00	6,12	0,00	140000	0,00	0,00	bayes_update
0,00	6,12	0,00	70000	0,00	0,00	bayes_region_I
0,00	6,12	0,00	10000	0,00	494,50	compute_sonar_matrix
0,00	6,12	0,00	10000	0,00	0,00	return_x
0,00	6,12	0,00	10000	0,00	0,00	return_y

Tabela E.1: Tabela de *Profiling* para dados do tipo *Double* e otimização O0.

Resultados para tipos de dados <i>Float</i> - Optimização O0						Função
Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (us/chamada)	Tempo total da função (us/chamada)	
33,38	7,29	7,29	...	...	...	acosf
26,97	13,18	5,89	...	...	...	sqrtf
16,58	16,8	3,62	10000	362,00	628,00	matrix_region
8,10	18,57	1,77	199980000	0,01	0,01	calc_norma
6,14	19,91	1,34	...	...	...	powf
3,48	20,67	0,76	10000	76,00	76,00	init_grid_bayes
2,34	21,18	0,51	99990000	0,01	0,01	internal_product
1,28	21,46	0,28	...	...	...	pow
0,92	21,66	0,20	99990000	0,00	0,00	find_vector
0,78	21,83	0,17	15640000	0,01	0,01	return_distance
0,05	21,84	0,01	140000	0,07	0,07	bayes_update
0,00	21,84	0,00	70000	0,00	0,00	bayes_region_I
0,00	21,84	0,00	70000	0,00	0,00	bayes_region_II
0,00	21,84	0,00	10000	0,00	628,00	compute_sonar_matrix
0,00	21,84	0,00	10000	0,00	0,00	return_x
0,00	21,84	0,00	10000	0,00	0,00	return_y

Tabela E.2: Tabela de *Profiling* para dados do tipo *Float* e optimização O0.

Resultados para tipos de dados <i>Double</i> - Optimização O2						Função
Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (ms/chamada)	Tempo total da função (ms/chamada)	
61,94	8,30	8,30	199980000	0,00	0,00	calc_norma
20,37	11,03	2,73	10000	0,27	1,26	matrix_region
6,12	11,85	0,82	15640000	0,00	0,00	return_distance
3,43	12,31	0,46	...	...	...	pow
2,91	12,70	0,39	99990000	0,00	0,00	internal_product
2,61	13,05	0,35	99990000	0,00	0,00	find_vector
1,94	13,31	0,26	10000	0,03	0,03	init_grid_bayes
0,67	13,40	0,09	...	...	...	acos
0,00	13,40	0,00	130000	0,00	0,00	bayes_update
0,00	13,40	0,00	70000	0,00	0,00	bayes_region_II
0,00	13,40	0,00	60000	0,00	0,00	bayes_region_I
0,00	13,40	0,00	10000	0,00	1,26	compute_sonar_matrix
0,00	13,40	0,00	10000	0,00	0,00	return_x
0,00	13,40	0,00	10000	0,00	0,00	return_y

Tabela E.3: Tabela de *Profiling* para dados do tipo *Double* e optimização O2.

Resultados para tipos de dados <i>Float</i> - Optimização O2						Função
Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (us/chamada)	Tempo total da função (us/chamada)	
38,91	9,37	9,37	...	...	...	acosf
36,96	18,27	8,90	199980000	0,00	0,00	calc_norma
15,32	21,96	3,69	10000	0,37	0,37	matrix_region
3,07	22,70	0,74	15640000	0,00	0,00	return_distance
1,87	23,15	0,45	99990000	0,00	0,00	internal_product
1,83	23,59	0,44	...	0,00	...	pow
1,25	23,89	0,30	99990000	0,00	0,00	find_vector
0,79	24,08	0,19	10000	0,02	0,02	init_grid_bayes
0,00	24,08	0,00	140000	0,00	0,00	bayes_update
0,00	24,08	0,00	70000	0,00	0,00	bayes_region_I
0,00	24,08	0,00	70000	0,00	0,00	bayes_region_II
0,00	24,08	0,00	10000	0,00	0,00	compute_sonar_matrix
0,00	24,08	0,00	10000	0,00	0,00	return_x
0,00	24,08	0,00	10000	0,00	0,00	return_y

Tabela E.4: Tabela de *Profiling* para dados do tipo *Float* e optimização O2.

Resultados para tipos de dados <i>Double</i> - Optimização O3						Função
Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (s/chamada)	Tempo total da função ms/chamada	
93,96	12,28	12,28	10000	1,23	1,23	matrix_region
2,75	12,64	0,36	...	...	...	pow
2,37	12,95	0,31	...	...	...	main
0,92	13,07	0,12	...	...	...	acos

Tabela E.5: Tabela de *Profiling* para dados do tipo *Double* e optimização O3.

Resultados para tipos de dados <i>Float</i> - Optimização O3						Função
Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (s/chamada)	Tempo total da função ms/chamada	
55,00	13,43	13,43	10000	1,34	1,34	matrix_region
42,83	23,89	10,46	...	...	...	acosf
1,52	24,26	0,37	...	...	...	pow
0,66	24,42	0,16	...	...	...	main

Tabela E.6: Tabela de *Profiling* para dados do tipo *Float* e optimização O3.

E.2 Execução no Sistema Caracterizado. $\mu\text{Blaze}@50\text{MHz}$

k	Resultados para tipos de dados <i>Float</i> - Otimização O2						Função
	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (s/chamada)	Tempo total da função (s/chamada)	
1	22,14	74,18	74,18	...	...	...	__divdf3
2	15,84	127,24	53,05	...	...	...	__unpack_f
3	12,06	167,64	40,41	...	...	...	__umodsi3
4	10,08	201,40	33,76	...	...	...	__udivsi3
5	7,02	224,92	23,52	...	...	...	__mulsf3
6	5,90	244,69	19,78	...	...	...	_fpadd_parts
7	5,00	261,43	16,74	...	...	...	__do_global_ctors_aux
8	4,42	276,24	14,81	...	...	...	__pack_f
9	4,32	290,72	14,48	...	...	...	__muldi3
10	2,69	299,73	9,01	...	...	...	__ieee754_sqrtf
11	2,65	308,59	8,86	...	...	...	__floatsisf
12	2,45	316,81	8,21	...	...	...	__divsf3
13	0,89	319,78	2,97	...	...	...	__addsf3
14	0,66	322,00	2,23	...	...	...	__subsf3
15	0,60	324,00	2,00	...	...	...	__unpack_d
16	0,56	325,89	1,89	19998	0,00	0,00	calc_norma
17	0,43	327,33	1,44	...	...	...	__ieee754_acosf
18	0,29	328,30	0,97	...	...	...	__pack_d
19	0,29	329,27	0,96	...	...	...	__fpcmp_parts_f
20	0,26	330,13	0,86	...	...	...	__ltsf2
21	0,21	330,83	0,70	9999	0,00	0,00	find_vector
22	0,17	331,39	0,56	9999	0,00	0,00	internal_product
23	0,16	331,94	0,55	...	...	...	__ledf2
24	0,15	332,44	0,50	1	0,50	3,84	matrix_region
25	0,15	332,92	0,49	...	...	...	sqrtf
26	0,11	333,28	0,36	...	...	...	acosf
27	0,10	333,61	0,33	...	...	...	__gtsf2
28	0,08	333,90	0,28	...	...	...	__extendsfdf2
29	0,08	334,17	0,28	...	...	...	__fpcmp_parts_d
30	0,06	334,36	0,19	1564	0,00	0,00	return_distance
31	0,05	334,53	0,17	...	...	...	isnanf
32	0,03	334,62	0,09	...	...	...	__lesf2
33	0,03	334,70	0,08	...	...	...	__gesf2
34	0,02	334,77	0,07	1	0,07	0,07	init_grid_bayes
35	0,02	334,82	0,06	...	...	...	__ieee754_rem_pio2f
36	0,01	334,87	0,05	...	...	...	__make_dp
37	0,01	334,92	0,05	...	...	...	__muldf3
38	0,01	334,95	0,03	...	...	...	fabsf
39	0,01	334,97	0,02	...	...	...	matherr
40	0,01	334,99	0,02	...	...	...	sinf
41	0,00	334,99	0,00	1	0,00	0,00	return_y
42	0,00	334,99	0,00	14	0,00	0,00	bayes_update
43	0,00	334,99	0,00	7	0,00	0,00	bayes_region_l
44	0,00	334,99	0,00	7	0,00	0,00	bayes_region_ll
45	0,00	334,99	0,00	...	...	...	__fixfsi
46	0,00	334,99	0,00	...	...	...	__truncdfsf2
47	0,00	334,99	0,00	...	...	...	__make_fp
48	0,00	334,99	0,00	1	0,00	0,00	return_x
49	0,00	334,99	0,00	...	...	...	__do_global_dtors_aux
50	0,00	335,00	0,00	...	...	...	rintf
51	0,00	335,00	0,00	1	0,00	3,84	compute_sonar_matrix
52	0,00	335,00	0,00	1	0,00	3,91	main

Tabela E.9: Tabela de *Profiling* para dados do tipo *Float* e otimização O2 no sistema embebido.

Resultados para tipos de dados <i>Double</i> - Optimização O3							Função
<i>k</i>	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (s/chamada)	Tempo total da função (s/chamada)	
1	14,32	77,34	77,34	...	...	...	__unpack_d
2	13,97	152,77	75,43	...	...	...	__muldf3
3	13,77	227,13	74,36	...	...	...	__divdf3
4	12,86	296,58	69,46	...	...	...	__floatsidf
5	12,64	364,83	68,25	...	...	...	__ieee754_sqrt
6	11,10	424,77	59,93	...	...	...	__muldi3
7	6,97	462,39	37,63	...	...	...	_fpadd_parts
8	6,77	498,96	36,57	...	...	...	__pack_d
9	3,13	515,88	16,92	...	...	...	__do_global_ctors_aux
10	0,91	520,79	4,91	...	...	...	__adddf3
11	0,67	524,38	3,59	...	...	...	__udivsi3
12	0,61	527,70	3,31	...	...	...	__subdf3
13	0,55	530,67	2,97	...	...	...	__ieee754_acos
14	0,35	532,55	1,88	...	...	...	isnan
15	0,30	534,15	1,60	...	...	...	__fpcmp_parts_d
16	0,23	535,37	1,22	1	1,22	1,22	matrix_region
17	0,22	536,57	1,21	...	...	...	__ltdf2
18	0,21	537,72	1,15	...	...	...	sqrt
19	0,12	538,37	0,65	...	...	...	__ledf2
20	0,12	539,00	0,63	...	...	...	acos
21	0,10	539,54	0,54	...	...	...	__gtdf2
22	0,03	539,73	0,19	1	0,19	1,41	main
23	0,02	539,81	0,09	...	...	...	fabs
24	0,01	539,89	0,08	...	...	...	__gedf2
25	0,00	539,92	0,02	...	...	...	__ieee754_rem_pio2
26	0,00	539,93	0,01	...	...	...	sin
27	0,00	539,93	0,00	...	...	...	__fixdfsi
28	0,00	539,93	0,00	...	...	...	rint

Tabela E.10: Tabela de *Profiling* para dados do tipo *Double* e optimização O3 no sistema embebedo.

k	Resultados para tipos de dados <i>Float</i> - Optimização O3						Função
	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (s/chamada)	Tempo total da função (s/chamada)	
1	28,01	52,35	52,35	...	...	...	__unpack_f
2	11,87	74,55	22,19	...	...	...	__mulsf3
3	10,16	93,54	19,00	...	...	...	_fpadd_parts
4	9,24	110,82	17,28	...	...	...	__pack_f
5	8,99	127,62	16,80	...	...	...	__do_global_ctors_
6	7,86	142,31	14,69	...	...	...	__muldi3
7	4,80	151,27	8,96	...	...	...	__ieee754_sqrtf
8	4,42	159,52	8,25	...	...	...	__floatsisf
9	4,36	167,66	8,14	...	...	...	__divsf3
10	1,90	171,21	3,54	...	...	...	__udivsi3
11	1,62	174,23	3,02	...	...	...	__addsf3
12	1,10	176,28	2,05	...	...	...	__unpack_d
13	1,08	178,30	2,02	...	...	...	__subsf3
14	0,77	179,74	1,44	...	...	...	__ieee754_acosf
15	0,71	181,06	1,32	...	...	...	__ltsf2
16	0,53	182,04	0,98	...	...	...	__pack_d
17	0,41	182,82	0,77	...	...	...	__ledf2
18	0,38	183,53	0,71	1	0,71	0,71	matrix_region
19	0,35	184,19	0,66	...	...	...	__fpcmp_parts_f
20	0,25	184,66	0,47	...	...	...	sqrtf
21	0,25	185,12	0,46	...	...	...	__gtsf2
22	0,19	185,47	0,35	...	...	...	isnanf
23	0,18	185,81	0,34	...	...	...	acosf
24	0,14	186,07	0,26	1	0,26	0,97	main
25	0,13	186,32	0,24	...	...	...	__extendsfdf2
26	0,10	186,49	0,18	...	...	...	__truncdfsf2
27	0,07	186,62	0,13	...	...	...	__fpcmp_parts_d
28	0,04	186,69	0,07	...	...	...	__make_dp
29	0,03	186,76	0,06	...	...	...	__lesf2
30	0,03	186,81	0,06	...	...	...	floorf
31	0,02	186,86	0,04	...	...	...	__gesf2
32	0,02	186,89	0,03	...	...	...	fabsf
33	0,00	186,89	0,00	...	...	...	__fixsfsi
34	0,00	186,90	0,00	...	...	...	__muldf3
35	0,00	186,90	0,00	...	...	...	rintf
36	0,00	186,90	0,00	...	...	...	cosf
37	0,00	186,90	0,00	...	...	...	__make_fp

Tabela E.11: Tabela de *Profiling* para dados do tipo *Float* e optimização O3 no sistema embebido.

Resultados para tipos de dados <i>Float</i> com FPU e Optimização O3							Função
<i>k</i>	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (ms/chamada)	Tempo total da função (ms/chamada)	
1	36,51	16,73	16,73	...	...	...	__do_global_ctors_aux
2	18,64	25,26	8,54	...	...	...	__ieee754_sqrtf
3	17,14	33,12	7,85	...	...	...	__floatsisf
4	7,67	36,63	3,51	...	...	...	__udivsi3
5	4,22	38,56	1,93	...	...	...	__unpack_d
6	3,64	40,23	1,67	...	...	...	__pack_f
7	1,86	41,08	0,85	...	...	...	__pack_d
8	1,59	41,81	0,73	...	...	...	__unpack_f
9	1,52	42,51	0,70	...	...	...	__ieee754_acosf
10	1,40	43,15	0,64	...	...	...	sqrtf
11	1,23	43,71	0,56	...	...	...	acosf
12	1,10	44,22	0,50	...	...	...	__ledf2
13	0,86	44,61	0,39	...	...	...	__fpcmp_parts_d
14	0,74	44,95	0,34	1	340,60	340,60	matrix_region
15	0,64	45,24	0,29	...	...	...	isnanf
16	0,41	45,43	0,19	...	...	...	__extendsfdf2
17	0,40	45,62	0,18	...	...	...	__make_dp
18	0,16	45,69	0,07	1	71,40	412,00	main
19	0,11	45,74	0,05	...	...	...	floorf
20	0,10	45,78	0,04	...	...	...	fabsf
21	0,05	45,80	0,02	...	...	...	matherr
22	0,02	45,81	0,01	...	...	...	__fixfsi
23	0,01	45,81	0,00	...	...	...	__muldf3
24	0,01	45,81	0,00	...	...	...	__muldi3
25	0,00	45,82	0,00	...	...	...	__truncdfsf2
26	0,00	45,82	0,00	...	...	...	rintf

Tabela E.12: Tabela de *Profiling* para dados do tipo *Float* com *FPU* e optimização O3 no sistema embebido

Resultados para tipos de dados <i>Float</i> com <i>CACHE</i> e Optimização O3							Função
<i>k</i>	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (ms/chamada)	Tempo total da função (ms/chamada)	
1	62,33	16,81	16,81	...	...	...	__do_global_ctors_aux
2	13,17	20,36	3,55	...	...	...	__udivsi3
3	5,11	21,73	1,38	...	...	...	__unpack_f
4	4,07	22,83	1,10	...	...	...	_fpadd_parts
5	3,93	23,89	1,06	...	...	...	__mulsf3
6	3,08	24,72	0,83	...	...	...	__pack_f
7	2,79	25,48	0,75	...	...	...	__muldi3
8	0,92	25,72	0,25	...	...	...	__ieee754_sqrtf
9	0,77	25,93	0,21	...	...	...	__floatsisf
10	0,72	26,13	0,19	...	...	...	__divsf3
11	0,39	26,23	0,11	...	...	...	__unpack_d
12	0,36	26,33	0,10	...	...	...	__addsf3
13	0,33	26,42	0,09	...	...	...	__pack_d
14	0,30	26,50	0,08	...	...	...	__ieee754_acosf
15	0,26	26,57	0,07	...	...	...	__fpcmp_parts_f
16	0,23	26,63	0,06	1	61,10	61,10	matrix_region
17	0,21	26,69	0,06	...	...	...	__ltsf2
18	0,17	26,73	0,05	...	...	...	__subsf3
19	0,16	26,77	0,04	...	...	...	__gtsf2
20	0,13	26,81	0,03	...	...	...	sqrtf
21	0,10	26,84	0,03	...	...	...	acosf
22	0,09	26,86	0,03	...	...	...	isnanf
23	0,09	26,89	0,03	...	...	...	__ledf2
24	0,05	26,90	0,01	...	...	...	__fpcmp_parts_d
25	0,04	26,91	0,01	1	11,70	72,80	main
26	0,04	26,92	0,01	...	...	...	__extendsfdf2
27	0,04	26,94	0,01	...	...	...	__make_dp
28	0,03	26,94	0,01	...	...	...	floorf
29	0,02	26,95	0,01	...	...	...	__truncdfsf2
30	0,02	26,95	0,00	...	...	...	__lesf2
31	0,01	26,96	0,00	...	...	...	__gesf2
32	0,01	26,96	0,00	...	...	...	fabsf
33	0,00	26,96	0,00	...	...	...	__fixsfsi
34	0,00	26,96	0,00	...	...	...	__muldf3
35	0,00	26,96	0,00	...	...	...	__kernel_sinf

Tabela E.13: Tabela de *Profiling* para dados do tipo *Float* com memória cache e optimização O3 no sistema embebido.

E.3 Optimizações específicas no sistema embebido.

Resultados para tipos de dados <i>Double</i> com Optimização O0							Função
<i>k</i>	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (s/chamada)	Tempo total da função (s/chamada)	
1	12,39	94,57	94,57	...	...	...	__unpack_d
2	10,79	176,89	82,32	...	...	...	__muldf3
3	9,98	253,02	76,13	...	...	...	__divdf3
4	9,69	326,98	73,96	...	...	...	__udivsi3
5	9,50	399,46	72,48	...	...	...	__divsi3
6	9,43	471,45	71,99	...	...	...	__ieee754_sqrt
7	7,68	530,03	58,58	...	...	...	__muldi3
8	7,42	586,68	56,65	...	...	...	__floatsidf
9	5,15	625,98	39,30	...	...	...	_fpadd_parts
10	5,14	665,18	39,21	...	...	...	__pack_d
11	3,54	692,16	26,98	...	...	...	__do_global_ctors_aux
12	3,25	716,96	24,80	...	...	...	__umodsi3
13	1,14	725,63	8,66	...	...	...	isnan
14	0,78	731,58	5,96	...	...	...	__eqdf2
15	0,73	737,15	5,57	...	...	...	__adddf3
16	0,54	741,31	4,16	...	...	...	__ieee754_pow
17	0,45	744,71	3,40	...	...	...	calc_norma
18	0,44	748,05	3,34	...	...	...	__subdf3
19	0,33	750,59	2,54	...	...	...	pow
20	0,29	752,79	2,20	...	...	...	__ieee754_acos
21	0,24	754,60	1,81	...	...	...	__fpcmp_parts_d
22	0,20	756,10	1,50	...	...	...	__ltdf2
23	0,18	757,51	1,41	...	...	...	finite
24	0,11	758,35	0,84	1	...	...	matrix_region
25	0,11	759,16	0,81	...	...	...	internal_product
26	0,10	759,96	0,80	...	...	...	find_vector
27	0,09	760,66	0,71	...	...	...	__gtdf2
28	0,09	761,32	0,65	...	...	...	sqrt
29	0,07	761,82	0,50	...	...	...	acos
30	0,06	762,29	0,47	...	...	...	__ledf2
31	0,05	762,64	0,35	...	...	...	init_grid_bayes
32	0,04	762,91	0,27	...	...	...	return_distance
33	0,02	763,02	0,11	...	...	...	fabs
34	0,01	763,11	0,09	...	...	...	__gedf2
35	0,00	763,11	0,00	14	0,00	0,00	bayes_update
36	0,00	763,11	0,00	7	0,00	0,00	bayes_region_II
37	0,00	763,12	0,00	7	0,00	0,00	bayes_region_I
38	0,00	763,12	0,00	...	...	...	XUartLite_SendByte
39	0,00	763,12	0,00	65538	0,00	0,00	return_x
40	0,00	763,12	0,00	...	...	...	__fixdfsi
41	0,00	763,12	0,00	...	...	...	outbyte
42	0,00	763,12	0,00	...	...	...	rint
43	0,00	763,12	0,00	...	...	...	xil_printf
44	0,00	763,12	0,00	1	0,00	5,33	compute_sonar_matrix
45	0,00	763,12	0,00	...	...	...	cos
46	0,00	763,12	0,00	...	...	...	outnum
47	0,00	763,12	0,00	...	...	...	padding
48	0,00	763,12	0,00	75536	0,00	0,00	__fini
49	0,00	763,12	0,00	1	0,00	5,68	main

Tabela E.7: Tabela de *Profiling* para dados do tipo *Double* e optimização O0 no sistema embebido.

k	Resultados para tipos de dados <i>Double</i> - Optimização O2						Função
	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (s/chamada)	Tempo total da função (s/chamada)	
1	11,28	78,54	78,54	...	...	...	__unpack_d
2	11,03	155,28	76,74	...	...	...	__muldf3
3	10,78	230,36	75,07	...	...	...	__floatsidf
4	10,59	304,06	73,70	...	...	...	__divdf3
5	10,00	373,67	69,61	...	...	...	__ieee754_sqrt
6	8,42	432,27	58,60	...	...	...	__muldi3
7	6,22	475,59	43,32	...	...	...	__umodsi3
8	5,51	513,97	38,38	...	...	...	__fpadd_parts
9	5,31	550,95	36,98	...	...	...	__eqdf2
10	5,05	586,09	35,14	...	...	...	__pack_d
11	4,73	618,98	32,89	...	...	...	__divsi3
12	4,63	651,21	32,23	...	...	...	__udivsi3
13	2,42	668,03	16,82	...	...	...	__do_global_ctors_aux
14	0,79	673,54	5,52	...	...	...	__adddf3
15	0,53	677,22	3,67	...	...	...	__subdf3
16	0,48	680,59	3,37	...	...	...	__errno
17	0,36	683,07	2,48	...	...	...	__ieee754_acos
18	0,30	685,13	2,06	...	...	...	isnan
19	0,28	687,08	1,95	19998	0,00	0,00	calc_norma
20	0,22	688,59	1,50	...	...	...	__fixdfsi
21	0,17	689,77	1,18	...	...	...	sqrt
22	0,16	690,85	1,08	...	...	...	__fpcmp_parts_d
23	0,15	691,87	1,02	...	...	...	__ltdf2
24	0,11	692,65	0,78	9999	0,00	0,00	find_vector
25	0,10	693,38	0,73	1	0,73	3,96	matrix_region
26	0,09	694,01	0,63	...	...	...	__gtdf2
27	0,09	694,63	0,62	...	...	...	__ledf2
28	0,09	695,24	0,60	...	...	...	acos
29	0,04	695,55	0,31	9999	0,00	0,00	internal_product
30	0,03	695,73	0,18	1564	0,00	0,00	return_distance
31	0,02	695,84	0,11	...	...	...	__gedf2
32	0,01	695,93	0,09	...	...	...	fabs
33	0,01	696,01	0,09	1	0,09	0,09	init_grid_bayes
34	0,01	696,05	0,04	...	...	...	floor
35	0,00	696,08	0,03	...	...	...	__kernel_sin
36	0,00	696,08	0,00	7	0,00	0,00	bayes_region_II
37	0,00	696,08	0,00	14	0,00	0,00	bayes_update
38	0,00	696,08	0,00	7	0,00	0,00	bayes_region_I
39	0,00	696,08	0,00	1	0,00	0,00	return_x
40	0,00	696,08	0,00	1	0,00	3,96	compute_sonar_matrix
41	0,00	696,08	0,00	1	0,00	0,00	return_y
42	0,00	696,08	0,00	...	...	...	__profile_clean
43	0,00	696,08	0,00	...	...	...	rint
44	0,00	696,08	0,00	1	0,00	4,04	main

Tabela E.8: Tabela de *Profiling* para dados do tipo *Double* e optimização O2 no sistema embebido.

Resultados para tipos de dados <i>Float</i> com Optimização O3 e algoritmo optimizado							Função
<i>k</i>	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (ms/chamada)	Tempo total da função (ms/chamada)	
1	80,76	16,96	16,96	...	...	...	__do_global_ctors_aux
2	16,88	20,51	3,54	...	...	...	__udivsi3
3	0,63	20,64	0,13	...	...	...	__unpack_f
4	0,37	20,72	0,08	1,00	76,90	79,90	main
5	0,24	20,77	0,05	...	...	...	__mulsf3
6	0,23	20,81	0,05	...	...	...	_fpadd_parts
7	0,19	20,86	0,04	...	...	...	__pack_f
8	0,17	20,89	0,04	...	...	...	__muldi3
9	0,12	20,92	0,02	...	...	...	__divsf3
10	0,10	20,94	0,02	...	...	...	__ieee754_sqrtf
11	0,09	20,96	0,02	...	...	...	__floatsisf
12	0,03	20,96	0,01	...	...	...	__subsf3
13	0,03	20,97	0,01	...	...	...	__addsf3
14	0,03	20,98	0,01	...	...	...	__unpack_d
15	0,02	20,98	0,00	...	...	...	__ltsf2
16	0,01	20,98	0,00	1,00	3,00	3,00	matrix_region
17	0,01	20,99	0,00	...	...	...	sqrtf
18	0,01	20,99	0,00	...	...	...	__pack_d
19	0,01	20,99	0,00	...	...	...	__ieee754_acosf
20	0,01	20,99	0,00	...	...	...	__fpcmp_parts_f
21	0,01	20,99	0,00	...	...	...	isnanf
22	0,01	21,00	0,00	...	...	...	__muldf3
23	0,01	21,00	0,00	...	...	...	__fixsfsi
24	0,01	21,00	0,00	...	...	...	acosf
25	0,00	21,00	0,00	...	...	...	__ledf2
26	0,00	21,00	0,00	...	...	...	rintf
27	0,00	21,00	0,00	...	...	...	__extendsfdf2
28	0,00	21,00	0,00	...	...	...	__gtsf2
29	0,00	21,00	0,00	...	...	...	__truncdfsf2
30	0,00	21,00	0,00	...	...	...	__fpcmp_parts_d
31	0,00	21,00	0,00	...	...	...	__lesf2
32	0,00	21,00	0,00	...	...	...	__gesf2
33	0,00	21,00	0,00	...	...	...	floorf
34	0,00	21,00	0,00	...	...	...	__kernel_cosf
35	0,00	21,00	0,00	...	...	...	__kernel_sinf
36	0,00	21,00	0,00	...	...	...	cosf
37	0,00	21,00	0,00	...	...	...	__make_dp

Tabela E.14: Tabela de *Profiling* para dados do tipo *Float* com optimização O3 e algoritmo optimizado no sistema embestado.

Resultados para tipos de dados <i>Float</i> com FPU, Optimização O3 e algoritmo optimizado							Função
<i>k</i>	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (ms/chamada)	Tempo total da função (ms/chamada)	
1	82,40	16,87	16,87	...	...	...	__do_global_ctors_aux
2	17,16	20,38	3,51	...	...	...	__udivsi3
3	0,36	20,46	0,07	1,00	73,40	74,20	main
4	0,03	20,46	0,01	...	...	...	__floatsisf
5	0,01	20,46	0,00	...	...	...	__unpack_d
6	0,01	20,46	0,00	...	...	...	__unpack_f
7	0,01	20,46	0,00	...	...	...	__pack_f
8	0,00	20,47	0,00	1,00	0,80	0,80	matrix_region
9	0,00	20,47	0,00	...	...	...	acosf
10	0,00	20,47	0,00	...	...	...	sqrtf
11	0,00	20,47	0,00	...	...	...	__pack_d
12	0,00	20,47	0,00	...	...	...	__ledf2
13	0,00	20,47	0,00	...	...	...	__make_dp
14	0,00	20,47	0,00	...	...	...	__kernel_cosf
15	0,00	20,47	0,00	...	...	...	__extendsfdf2
16	0,00	20,47	0,00	...	...	...	__fixsfsi
17	0,00	20,47	0,00	...	...	...	__ieee754_acosf
18	0,00	20,47	0,00	...	...	...	isnanf
19	0,00	20,47	0,00	...	...	...	rintf
20	0,00	20,47	0,00	...	...	...	__muldi3
21	0,00	20,47	0,00	...	...	...	__fpcmp_parts_d
22	0,00	20,47	0,00	...	...	...	__kernel_sinf
23	0,00	20,47	0,00	...	...	...	_crtinit
24	0,00	20,47	0,00	...	...	...	cosf
25	0,00	20,47	0,00	...	...	...	sinf

Tabela E.15: Tabela de *Profiling* para dados do tipo *Float* com *FPU*, optimização O3 e algoritmo optimizado no sistema embebido.

Resultados para tipos de dados <i>Float</i> com <i>CACHE</i> , Otimização O3 e algoritmo otimizado							Função
<i>k</i>	Tempo (%)	Tempo Acumulado (s)	Tempo da função (s/chamada)	Nº de chamadas	Tempo médio (ms/chamada)	Tempo total da função (ms/chamada)	
1	82,62	16,97	16,97	...	...	...	__do_global_ctors_aux
2	17,29	20,52	3,55	...	...	...	__udivsi3
3	0,02	20,52	0,00	...	...	...	__unpack_f
4	0,01	20,53	0,00	...	...	...	_fpadd_parts
5	0,01	20,53	0,00	...	...	...	__mulsf3
6	0,01	20,53	0,00	...	...	...	__pack_f
7	0,01	20,53	0,00	...	...	...	__muldi3
8	0,01	20,53	0,00	1	1,35	1,75	main
9	0,00	20,53	0,00	...	...	...	__divsf3
10	0,00	20,53	0,00	...	...	...	__ieee754_sqrtf
11	0,00	20,54	0,00	...	...	...	__ieee754_acosf
12	0,00	20,54	0,00	1	0,40	0,40	matrix_region
13	0,00	20,54	0,00	...	...	...	__fpcmp_parts_f
14	0,00	20,54	0,00	...	...	...	__extendsfdf2
15	0,00	20,54	0,00	...	...	...	__unpack_d
16	0,00	20,54	0,00	...	...	...	sqrtf
17	0,00	20,54	0,00	...	...	...	__fixsfsi
18	0,00	20,54	0,00	...	...	...	__floatsisf
19	0,00	20,54	0,00	...	...	...	__fpcmp_parts_d
20	0,00	20,54	0,00	...	...	...	__ledf2
21	0,00	20,54	0,00	...	...	...	__ltsf2
22	0,00	20,54	0,00	...	...	...	__make_dp
23	0,00	20,54	0,00	...	...	...	__muldf3
24	0,00	20,54	0,00	...	...	...	__pack_d
25	0,00	20,54	0,00	...	...	...	acosf
26	0,00	20,54	0,00	...	...	...	__addsf3

Tabela E.16: Tabela de *Profiling* para dados do tipo *Float* com memória cache, otimização O3 e algoritmo otimizado no sistema embebido.

Comparação de tempos de <i>Profiling</i> e de medição por contagem de ciclos de relógio (<i>Bayes</i>)				
Algoritmo	Slices ocupados (%)	Tempo obtido por <i>Profiling</i> (s)	Tempo obtido por medição de ciclo de relógio (s)	Tipo de Otimização
Referência	93,00	186,90	74,88	<i>Float</i>
Referência	96,00	26,96	11,03	<i>Float</i> e memória <i>Cache</i>
Referência	99,00	45,82	11,55	<i>Float</i> e <i>FPU</i>
Otimizado	93,00	21,00	0,22	<i>Float</i>
Otimizado	96,00	20,54	0,03	<i>Float</i> e memória <i>Cache</i>
Otimizado	99,00	20,47	0,07	<i>Float</i> e <i>FPU</i>

Tabela E.17: Tabela comparativa de tempos de *profiling* e de tempos calculados por contagem de ciclos de relógio para várias configurações e otimização O3.

Resultados de tempos contabilizados a partir do número de ciclos de relógio para diferentes dimensões da grelha de ocupação					
Algoritmo	Tempo para implementação Bayes (s)	Tempo para implementação Baye * (s)	Tempo para implementação Dempster-Shafer (s)	Tempo para implementação Dempster-Shafer * (s)	Dimensões da grelha
Referência	1,10742	1,10716	2,05971	2,05934	32x32
Referência	2,72994	2,72933	5,11080	5,10989	50x50
Referência	4,49473	4,47276	8,43365	8,43216	64x64
Referência	11,03133	11,02891	20,76054	20,75692	100x100
Referência	18,11328	18,10768	34,22491	34,21735	128x128
Referência	24,90801	24,90258	46,95622	46,94810	150x150
Referência	44,36131	44,35167	83,70196	83,68754	200x200
Optimizado	0,07049	0,07023	0,12080	0,12043	32x32
Optimizado	0,07081	0,07020	0,12131	0,12040	50x50
Optimizado	0,07123	0,07023	0,12191	0,12043	64x64
Optimizado	0,07262	0,07020	0,12402	0,12040	100x100
Optimizado	0,07590	0,07030	0,12694	0,11938	128x128
Optimizado	0,07564	0,07021	0,12853	0,12041	150x150
Optimizado	0,07984	0,07020	0,13482	0,12039	200x200

* Não entra com a função de inicialização da grelha para a contagem

Tabela E.18: Tabela de valores de tempo calculados por contagem de ciclos de relógio em função da dimensão da grelha de ocupação.

Tendência de tempo utilizado à medida que a distância aumenta (algoritmo otimizado)			
Distância ao Objecto	Tempo calculado por ciclos de relógio (Bayes) (ms)	Tempo calculado por ciclos de relógio (Dempster-Shafer) (ms)	Nº de células na submatriz
5	12,23422	17,00996	10
6	27,40822	44,2637	24
7	32,3853	52,71882	28
8	37,39728	61,22586	32
9	42,93066	70,30398	36
10	70,19978	120,40194	60

Tabela E.19: Tabela de valores de tempo calculado por contagem de ciclos de relógio em função da distância a um obstáculo (só compreende a função de actualização da grelha).

Anexo F

Bibliografia Adicional

Englobamos neste apêndice, bibliografia que embora não referenciada ao longo do nosso trabalho, foi importante na contextualização dos vários tópicos abordados. Passamos a apresentá-los em forma de lista não ordenada.

- *Motion Planning, chapter Occupancy Grid Maps for Localization and Mapping* [27];
- *Design and application of a scalable embedded systems; architecture with an FPGA based operating infrastructure* [28];
- *Bearings-only localization and mapping* [29];
- *Architectural exploration tasks for on-chip embedded systems* [30];
- *Mapping with limited sensing* [31];
- *FPGA prototyping of a RISC processor core for embedded applications* [32];
- *Self-Reconfigurable embedded systems on Low-Cost FPGAs* [33];
- *Introduction to Autonomous Mobile Robots* [34];
- *The future of reconfigurable systems* [35];
- *Reconfigurable architectures for General-Purpose computing* [36];
- *FPGA implementation of embedded fuzzy controllers for robotic applications* [37];
- *Robotic mapping using soft computing methodologies* [38];
- *Features, design tools, and application domains of FPGAs* [39];
- *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems* [40];
- *Evidential mapping for mobile robots with range sensors* [41];
- *Multi-FPGA systems* [42];

- *A comparison of microcontrollers targeted to FPGA-based embedded applications* [43];
- *Software profiling for an FPGA-based CPU core* [44]
- *Computational principles of mobile robotics* [45];
- *Mobile Robotics: A Practical Introduction* [46].