



Plataforma robótica genérica para robô de logística, serviços ou vigilância com mecanismo de troca automática de bateria

Ivo Emanuel Milheiro de Sousa

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Dr. António Paulo Moreira

Coorientador: Eng. Héber Miguel Sobreira

24 de Julho de 2016

Resumo

O desenvolvimento e a programação de plataformas robóticas genéricas, de baixo custo e com a capacidade para operarem, de forma robusta, em diversos tipos de ambientes e por longos períodos de tempo é um desafio exigente.

Assim, esta dissertação tem como principal objetivo a integração e o desenvolvimento de *software* de controlo, monitorização, localização e tomada de decisões para uma plataforma robótica móvel genérica, com vista a ser utilizada em vários tipos de aplicações, desde a logística, vigilância, publicidade, etc. Entre estes, o foco principal passa pelo estudo e otimização dos seus algoritmos de localização, pois é uma área crucial nos robôs móveis e torna-se um desafio quando aplicado a robôs com limitações de *hardware* e destinados a operar em ambientes dinâmicos.

Com isto em vista, realizaram-se análises comparativas entre os dois mais promissores algoritmos de localização (AMCL e Perfect Match) para o tipo de aplicações já referidas, na sua implementação em ROS. Em adição, investigou-se e desenvolveu-se uma solução de supervisão destes algoritmos de localização, permitindo a sua coexistência e a deteção de situações de falha.

Numa outra vertente, a autonomia de um robô e a sua capacidade de estar sempre em operação é um fator muito importante. O tempo de inatividade durante o processo de carga das baterias e o facto de geralmente esta operação estar dependente de um operador humano é um grave problema. É frequente o operador esquecer-se de colocar o robô em carga, provocando a sua saída inesperada de serviço. Por outro lado, baterias descarregadas por largos períodos de tempo acabam por degradar-se e perder capacidade de carga. Assim, para contornar estes problemas criaram-se ferramentas no sentido de integrar um sistema de troca automática de baterias, através do controlo de quatro servomotores juntamente com um mecanismo que permite um encaixe automático das mesmas.

Abstract

The development and programming of generic robotic platforms, with low cost and the ability to operate robustly in various types of environments and for long periods is a demanding challenge.

Thus, this dissertation has as main objective the integration and development of control, monitoring, localization and decision-making software, for a generic mobile robotic platform in order to be used in a variety of applications from logistics, surveillance, advertising, among others. Between them, the main focus is the study and optimization of the localization algorithms, as it is a crucial area in mobile robots and becomes a challenge when applied to robots with limited hardware and designed to operate in dynamic environments.

With this in mind, were held comparative analysis of the two most promising localization algorithms (AMCL and Perfect Match) for the type of applications mentioned above, in its ROS implementation. In addition, it was investigated and developed a supervision solution of these localization algorithms, allowing their coexistence and detection of failure situations.

In another aspect, the autonomy of a robot and its ability to be always in operation is a very important factor. The downtime during the batteries charging process and that this operation is generally dependent on a human operator is a serious problem. Often the operator forgets to put the robot in charge, causing its unexpected departure service. On the other hand, discharged batteries for extended periods of time eventually degrade and lose its charge capacity. So, to get around these problems have been created tools in order to integrate an automatic batteries exchange system, through the control of four servomotor together with a mechanism that allows an automatic docking of the batteries.

Agradecimentos

Inicialmente gostaria de agradecer à empresa CLEVERHOUSE, em particular ao Eng. Jorge Paiva pelo entusiástico apoio a este projeto, fornecendo o robô e todo o equipamento necessário para o desenvolvimento e testes efetuados.

Queria também agradecer ao Dr. Paulo Farias pela ajuda e contribuição em várias partes deste projeto, através da partilha de conhecimento e parceria na resolução de problemas que foram surgindo.

Ao Sr. Fernando Guedes e ao Jorge Barbosa um enorme obrigado pelo auxílio na integração do *hardware* do robô, dando uma ajuda indispensável para o sucesso deste projeto.

Um obrigado a toda a equipa do INESC TEC pela ajuda neste período.

Quero agradecer especificamente ao Héber Sobreira, pelo constante auxílio na execução deste projeto, contribuindo com todo o seu conhecimento e dedicação para que esta dissertação terminasse com sucesso.

Um agradecimento especial ao meu orientador António Paulo Moreira, pelo incansável acompanhamento, apoio, contribuição e enorme disponibilidade para este projeto de dissertação e para o autor deste, tendo um papel verdadeiramente fundamental para que tudo corresse da melhor maneira possível.

Aos meus amigos no geral, mas especialmente do laboratório de robótica, um grande obrigado pelo apoio e pelos momentos de diversão e descontração, tornando a execução deste projeto um processo muito mais agradável.

Por fim, um enorme agradecimento aos meus pais e familiares pelo apoio contínuo ao longo destes anos, pois sem eles isto não seria possível de alcançar.

Ivo Sousa

Conteúdo

1	Introdução	1
1.1	Contexto	1
1.2	Motivação e Objetivos	1
1.3	Contribuições inovadoras	2
1.4	Estrutura da dissertação	3
2	Revisão bibliográfica	5
2.1	Métodos de localização	5
2.1.1	Métodos de localização relativa	5
2.1.2	Localização baseada em marcos e balizas	6
2.1.3	Métodos de localização probabilística	8
2.1.4	Métodos de ajuste de mapa	14
2.2	Visão geral dos métodos de localização para aplicação em ambientes dinâmicos .	18
2.3	Sistemas para carregamento ou troca de baterias	19
2.4	Conclusões	21
3	Robô Clever	23
3.1	<i>Hardware</i>	23
3.1.1	Baterias	23
3.1.2	Motores e <i>encoders</i>	25
3.1.3	Servomotores	26
3.1.4	Conversores <i>Step-Down</i>	28
3.1.5	Laser	29
3.1.6	Arduino	30
3.1.7	Sonares e LEDS	31
3.1.8	Computador	32
3.1.9	Sistema para logística	33
3.2	<i>Software ROS</i>	35
3.2.1	Nós	35
3.2.2	Tópicos	36
3.2.3	Serviços	36
3.2.4	Ações	36
3.2.5	Arquitetura geral	37
3.2.6	Descrição dos nós de ROS	39
3.2.7	Criação de mapas	51
3.3	Aplicações da plataforma robótica	52

4	Troca automática de baterias	53
4.1	<i>Hardware</i>	53
4.2	Programa de controlo dos servomotores	56
5	Análise dos métodos de localização	59
5.1	Análise e comparação de várias configurações do Perfect Match	59
5.1.1	Análise dos parâmetros do Perfect Match	59
5.1.2	Ideias para melhoramento do desempenho do Perfect Match	64
5.1.3	Comparação prática de diferentes configurações do Perfect Match	65
5.2	Análise e comparação de várias configurações do AMCL	94
5.2.1	Análise dos parâmetros do AMCL	94
5.2.2	Comparação prática de diferentes configurações do AMCL	106
5.3	Comparação entre o AMCL e o Perfect Match	131
5.3.1	Conclusões	142
5.4	Supervisão dos algoritmos de localização	144
6	Conclusão	147
7	Trabalho futuro	149
	Referências	151

Lista de Figuras

2.1	Exemplo de triangulação relativa a marcos [1].	8
2.2	Pseudo-algoritmo do filtro de Bayes [2].	9
2.3	[3]	15
2.4	Comparação entre a posição e a orientação real do robô (branco) com as estimativas (azul) [4].	16
2.5	Localização do robô usando o algoritmo Perfect Match (os pontos vermelhos são utilizados para a localização) com e sem rejeição de <i>outliers</i> (pontos a verde) [5].	17
3.1	Robô Clever.	23
3.2	Caixa que possui internamente as baterias para a alimentação elétrica do robô. . .	24
3.3	Duas baterias KB1272F1 ligadas em série, no interior da caixa.	24
3.4	Quatro baterias internas do robô, cada uma de 6 V, ligadas em série.	25
3.5	Motor EMG49.	25
3.6	Motores EMG49 com os respectivos <i>encoders</i> , montados no robô.	26
3.7	(a) Representação dos <i>drivers</i> MD49. (b) <i>Drivers</i> dos motores montados no robô.	27
3.8	Circuito de interface para a comunicação e alimentação elétrica dos servomotores AX-12.	27
3.9	Peça de alumínio desenhada para a elevação das baterias.	28
3.10	(a) Esquema do circuito conversor abaixador utilizando o LM2575HV e (b) o mesmo circuito integrado no robô.	28
3.11	(a) Esquema do circuito conversor abaixador utilizando o MAX724 e (b) o mesmo circuito integrado no robô.	29
3.12	Laser URG-04LX-UG01 montado no robô.	29
3.13	Ligações do Arduino Uno presente no robô.	30
3.14	Sonares SRF01 com representação dos pinos de ligação.	31
3.15	(a) Sonares SRF01 integrados na placa pertencente à zona média do robô. (b) Robô com a placa dos sonares montada.	31
3.16	(a) LEDs frontais (placa branca) e (b) LEDs traseiros do robô (placa branca). . .	32
3.17	Computador interno montado no robô.	33
3.18	Estrutura integrada na superfície superior do robô para o sistema de transporte de prateleiras móveis.	33
3.19	(a) Sistema de encaixe com cilindros metálicos, atuado por solenóides. (b) Estrutura de encaixe na parte inferior da prateleira.	34
3.20	Robô acoplado com a prateleira móvel.	34
3.21	Logo do ROS.	35
3.22	Arquitetura de ROS, visualizada através do <i>rqt_graph</i> , do nível de lançamento de <i>hardware</i>	37

3.23	Arquitetura de ROS, visualizada através do <i>rqt_graph</i> , do nível de lançamento de navegação.	38
3.24	Arquitetura de ROS, visualizada através do <i>rqt_graph</i> , do nível de lançamento de decisão.	38
3.25	Parâmetros ajustáveis em <i>runtime</i> , com o <i>dynamic_reconfigure</i> do <i>driver</i> do laser.	40
3.26	Máquina de estados do funcionamento do nó <i>/control_go_to_x_y_theta</i> para controlo de posição e navegação do robô.	49
3.27	Mapas criados para a navegação e localização do robô, utilizando o laser URG-04LX-UG01.	51
4.1	(a) Caixa com as baterias externas do robô, com três contactos de cobre. (b) Dois contactos de cobre na parte inferior do robô, a partir dos quais é recebida a alimentação elétrica para o funcionamento de todos os componentes do robô. . .	53
4.2	(a) e (b) Caixa das baterias com quatro pinos de encaixe destacados a vermelho. (c) Estrutura onde é realizado o encaixe das baterias, possuindo quatro servomotores laterais (realçados a vermelho) com uma peça de alumínio integrada. (d) Vista interior de dois dos quatro servomotores integrados no andar inferior do robô. . .	54
4.3	Peça de alumínio acoplada aos servomotores, para o levantamento automático da caixa de baterias.	55
4.4	Diagrama representativo do processo de troca de baterias, com comunicação entre a estação e o robô.	55
4.5	Diagrama de classes UML do programa de controlo dos servomotores, desenvolvido em C++ e aplicado em ROS.	57
4.6	Máquina de estados do programa de controlo dos servomotores.	58
5.1	Gráfico da função de custo, com o limiares de rejeição de <i>outliers</i> (<i>O</i>).	63
5.2	Trajectória descrita pelo robô durante os testes, sobreposta ao mapa utilizado para a navegação, visualizada através do <i>rviz</i> , em ROS.	65
5.3	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config1_OF</i> e a (b) <i>Config2_OF_F1F2</i> , para um ambiente sem <i>outliers</i>	68
5.4	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config1_OF</i> e a (b) <i>Config2_OF_F1F2</i> , para um ambiente sem <i>outliers</i>	69
5.5	Sequência de posições estimadas pelo robô (seta azul) em relação ao <i>groundtruth</i> (seta verde), com os pontos do laser representados pelas cores rosa (todos os pontos) e azul (só os pontos utilizados pelo algoritmo).	70
5.6	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config3_SO</i> e a (b) <i>Config4_SO_F1F2</i> , para o mesmo ambiente sem <i>outliers</i> . . .	72
5.7	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config3_SO</i> e a (b) <i>Config4_SO_F1F2</i> , para o mesmo ambiente sem <i>outliers</i>	73
5.8	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a <i>Config5_OF_F1F2WD</i> , nas mesmas condições que as configurações anteriores. . . .	74
5.9	Evolução do erro em distância, ao longo do percurso, utilizando a <i>Config5_OF_F1F2WD</i> , nas mesmas condições que as configurações anteriores.	75
5.10	Ambiente onde se realizaram os testes, tendo-se colocado alguns obstáculos junto às paredes e no centro da trajectória do robô.	75
5.11	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a <i>Config1_OF</i> , para um ambiente com alguns <i>outliers</i>	76
5.12	Evolução do erro de distância, ao longo do percurso, utilizando a <i>Config1_OF</i> para um ambiente com alguns <i>outliers</i>	77

5.13	(a) Robô na posição inicial, bem localizado. (b) A integração de pontos do laser correspondentes à detecção de obstáculos externos ao mapa resulta num aumento momentâneo do erro de localização. (c) e (d) Numa zona com menos <i>outliers</i> o robô volta a conseguir recuperar a posição correta.	77
5.14	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a <i>Config2_OF_F1F2</i> , para um ambiente com alguns <i>outliers</i>	78
5.15	Evolução do erro de distância, ao longo do percurso, utilizando a <i>Config2_OF_F1F2</i> para um ambiente com alguns <i>outliers</i>	78
5.16	(a) Robô na posição inicial, bem localizado. (b) Os pontos a cor de rosa (<i>outliers</i>) são descartados pelo algoritmo. (c) e (d) O Perfect Match consegue manter uma boa estimativa da localização do robô (a seta azul corresponde à estimativa da localização e a seta verde ao <i>groundtruth</i>).	79
5.17	(a) Robô na posição inicial, bem localizado. (b) O algoritmo integra alguns pontos referentes aos obstáculos que estão no mapa e aumenta o erro de localização. (c) Numa zona onde há menos <i>outliers</i> e mais dados referentes ao mapa, o robô começa a convergir para a posição correta. (d) Robô bem localizado.	81
5.18	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a <i>Config6_OF_F2</i> , para um ambiente com alguns <i>outliers</i>	81
5.19	Evolução do erro de distância, ao longo do percurso, utilizando a <i>Config6_OF_F2</i> para um ambiente com alguns <i>outliers</i>	82
5.20	(a) Robô na posição inicial (seta azul), bem localizado. (b) O algoritmo descarta o conjunto de pontos a cor de rosa (<i>outliers</i> a mais de 0.5 m dos pontos esperados). (c) Os obstáculos próximos da parede (a menos de 0.5 m) são integrados no algoritmo (pontos azuis), levando a um aumento do erro de localização. (d) Na zona sem <i>outliers</i> o robô volta a recuperar a posição correta.	83
5.21	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a <i>Config7_OF_F1</i> , para um ambiente com alguns <i>outliers</i>	84
5.22	Evolução do erro de distância, ao longo do percurso, utilizando a <i>Config7_OF_F1</i> para um ambiente com alguns <i>outliers</i>	85
5.23	Ambiente onde se realizaram os testes, tendo-se aumentado a quantidade de objetos externos ao mapa, com vista a perturbar a perceção do ambiente por parte do robô.	86
5.24	Resultados para a <i>Config2_OF_F1F2</i> : (a) Robô na posição inicial (seta azul), bem localizado em relação ao <i>groundtruth</i> . (b) O algoritmo, ao utilizar alguns pontos do laser referentes a <i>outliers</i> começa a divergir da posição correta. (c) e (d) O robô acaba por se perder.	87
5.25	Resultados para a <i>Config5_OF_F1F2WD</i> : (a) Robô na posição inicial (seta azul), bem localizado em relação ao <i>groundtruth</i> . (b), (c) e (d) Com esta configuração o Perfect Match consegue manter uma boa estimativa de localização ao longo do tempo, ganhando algum erro a meio do percurso, devido a uma grande quantidade de <i>outliers</i>	88
5.26	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config2_OF_F1F2</i> e a (b) <i>Config5_OF_F1F2WD</i> , para um ambiente com muitos obstáculos.	89
5.27	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config2_OF_F1F2</i> e a (b) <i>Config5_OF_F1F2WD</i> , para um ambiente com grande quantidade de <i>outliers</i>	90

5.28	(a) Robô na posição inicial (seta azul), bem localizado em relação ao <i>groundtruth</i> . (b) e (c) Primeiro e segundo locais, respectivamente onde se forçou um deslizamento das rodas do robô, levando a um aumento do erro da estimativa de localização devido aos dados errados da odometria. (d) O robô volta a convergir para a posição correta.	91
5.29	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config2_OF_F1F2</i> e a (b) <i>Config5_OF_F1F2WD</i> , com dois momentos onde se forçou o erro da odometria (correspondentes aos dois aumentos de erro visíveis no gráfico).	92
5.30	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config2_OF_F1F2</i> e a (b) <i>Config5_OF_F1F2WD</i> , com dois momentos onde se forçou o erro da odometria (correspondentes aos dois picos no gráfico).	93
5.31	Pseudo-algoritmo de Monte Carlo Localization (MCL) [2].	95
5.32	Pseudo-algoritmo do modelo Augmented MCL [2].	97
5.33	Distribuição da densidade de probabilidade Gaussiana p_{hit} [2].	99
5.34	Distribuição da densidade de probabilidade exponencial p_{short} [2].	100
5.35	Distribuição da densidade de probabilidade uniforme p_{max} [2].	100
5.36	Distribuição da densidade de probabilidade uniforme p_{rand} [2].	101
5.37	Distribuição da densidade de probabilidade combinada dos 4 modelos: p_{hit} , p_{short} , p_{max} e p_{rand} [2].	101
5.38	Pseudo-algoritmo do modelo "beam range finder" [2].	102
5.39	Pseudo-algoritmo do modelo "likelihood field range finder" [2].	103
5.40	Pseudo-algoritmo do modelo da odometria "sample motion model" [2].	104
5.41	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> e a (b) <i>Config2_OA_02</i> , na ausência de <i>outliers</i>	109
5.42	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> e a (b) <i>Config2_OA_02</i> , na ausência de <i>outliers</i>	110
5.43	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config3_OA_05</i> e a (b) <i>Config4_OA_05L</i>	112
5.44	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config3_OA_05</i> e a (b) <i>Config4_OA_05L</i>	113
5.45	Dispersão das partículas em função dos parâmetros <i>odom_alpha</i> 's com (a) 0.01 (b) 0.2 e (c) 0.5.	114
5.46	(a) Robô bem localizado, em relação ao <i>groundtruth</i> (seta verde) com uma reduzida dispersão de partículas (setas vermelhas). (b) No primeiro local onde se forçou o erro de odometria a estimativa de posição dada pelo AMCL (seta amarela) ganhou algum erro. (c) Na segunda zona onde houve um erro de odometria o robô já estava com um elevado erro de posição e em (d) continuou a divergir, perdendo-se.	115
5.47	(a) Robô bem localizado, em relação ao <i>groundtruth</i> (seta verde) com alguma dispersão de partículas (setas vermelhas). (b) No primeiro local onde se forçou o erro de odometria a estimativa de posição dada pelo AMCL (seta amarela) ganhou algum erro. (c) Segunda zona onde houve um erro nos dados da odometria. (d) O robô conseguiu recuperar a localização correta.	116
5.48	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> , (b) <i>Config2_OA_02</i> e (c) <i>Config3_OA_05</i>	117

5.49	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> , (b) <i>Config2_OA_02</i> e (c) <i>Config3_OA_05</i> . É possível identificar os dois momentos em que se forçou um erro de odometria, correspondentes aos picos visíveis nas curvas do erro.	118
5.50	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> , (b) <i>Config2_OA_02</i> e (c) <i>Config4_OA_05L</i>	120
5.51	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> , (b) <i>Config2_OA_02</i> e (c) <i>Config4_OA_05L</i>	121
5.52	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> e a (b) <i>Config2_OA_02</i>	122
5.53	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config1_OA_001</i> e a (b) <i>Config2_OA_02</i>	123
5.54	Sequência de imagens referentes à <i>Config4_OA_05L</i> : (a) e (b) Robô bem localizado, em relação ao <i>groundtruth</i> (seta verde) com alguma dispersão de partículas (setas vermelhas). (c) Devido aos <i>outliers</i> presentes no mapa, o robô ganha algum erro de posição (seta amarela) e formam-se duas concentrações principais de partículas, paralelas. (d) As partículas correspondentes à posição desajustada do robô começam a ser eliminadas e o robô volta a convergir para a posição correta. . . .	124
5.55	Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) <i>Config3_OA_05</i> e a (b) <i>Config4_OA_05L</i>	125
5.56	Evolução do erro em distância, ao longo do percurso, utilizando a (a) <i>Config3_OA_05</i> e a (b) <i>Config4_OA_05L</i>	126
5.57	Evolução do tempo de ciclo para a execução do programa AMCL, com 300 partículas em comparação com o programa Perfect Match.	131
5.58	Evolução do tempo de ciclo para a execução do programa AMCL, com 200 partículas em comparação com o programa Perfect Match.	132
5.59	Evolução do tempo de ciclo para a execução do programa AMCL, com 100 partículas em comparação com o programa Perfect Match.	132
5.60	Evolução do tempo de ciclo para a execução do programa AMCL, com 50 partículas em comparação com o programa Perfect Match.	133
5.61	Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente sem obstáculos.	135
5.62	Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente sem obstáculos.	135
5.63	Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com alguns objetos externos ao mapa.	137
5.64	Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com alguns objetos externos ao mapa.	137
5.65	(a) Boas estimativas de posição, dadas pelo algoritmo AMCL (a seta amarela corresponde à partícula com maior peso e as restantes estão representadas com setas vermelhas) e pelo Perfect Match (seta azul), em relação ao <i>groundtruth</i> (seta verde) (b) Aumento do erro das estimativas de localização, na fase final do percurso do robô.	138

5.66	Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com muitos obstáculos.	139
5.67	Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com muitos obstáculos.	139
5.68	(a) Robô inicialmente bem localizado, tanto pela estimativa do AMCL (seta amarela), como pela estimativa do Perfect Match (seta azul), em relação ao <i>ground-truth</i> (seta verde). (b) Primeiro local onde se forçou o erro de odometria, com o consequente aumento do erro por parte dos dois algoritmos. (c) Segunda zona onde a odometria devolveu valores errados, aumentando de novo os erros das estimativas, após primeira recuperação. (d) Ambos os algoritmos de localização conseguiram voltar a recuperar a posição correta.	140
5.69	Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô com dois momentos de erros forçados de odometria.	141
5.70	Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô com dois momentos de erros forçados de odometria.	141
5.71	Diagrama de classes UML do programa supervisor dos algoritmos de localização, desenvolvido em C++ e aplicado em ROS.	145

Lista de Tabelas

3.1	Especificações das baterias KB1272F1	24
3.2	Especificações dos motores e <i>encoders</i>	25
3.3	Conectores para a alimentação elétrica dos motores	26
3.4	Conexões dos <i>encoders</i>	26
3.5	Especificações dos <i>drivers</i> MD49.	26
3.6	Especificações dos servomotores AX-12.	27
3.7	Especificações do conversor LM2575HV.	28
3.8	Especificações do conversor MAX724.	29
3.9	Especificações do laser URG-04LX-UG01.	30
3.10	Especificações dos sonares SRF01.	31
3.11	Especificações do processador Atom D400.	32
5.1	Parâmetros comuns a todas as configurações desta secção.	66
5.2	Definição da <i>Config1_OF</i> e da <i>Config2_OF_F1F2</i>	67
5.3	Resultados estatísticos da <i>Config1_OF</i> e <i>Config2_OF_F1F2</i> , para um ambiente sem <i>outliers</i>	67
5.4	Definição da <i>Config3_SO</i> e da <i>Config4_SO_F1F2</i>	71
5.5	Resultados estatísticos da <i>Config3_SO</i> e <i>Config4_SO_F1F2</i> , na ausência de <i>outliers</i>	71
5.6	Definição da <i>Config5_OF_F1F2WD</i>	74
5.7	Resultados estatísticos da <i>Config5_OF_F1F2WD</i> , na ausência de <i>outliers</i>	74
5.8	Resultados estatísticos da <i>Config1_OF</i> , com alguns obstáculos no mapa.	76
5.9	Resultados estatísticos da <i>Config2_OF_F1F2</i> , com alguns obstáculos no mapa.	79
5.10	Definição da <i>Config6_OF_F2</i>	80
5.11	Resultados estatísticos da <i>Config6_OF_F2</i> , com alguns obstáculos no mapa.	80
5.12	Definição da <i>Config7_OF_F1</i>	83
5.13	Resultados estatísticos da <i>Config7_OF_F1</i> , com alguns obstáculos no mapa.	84
5.14	Resultados estatísticos das <i>Config2_OF_F1F2</i> e <i>Config5_OF_F1F2WD</i> , com grande quantidade de obstáculos no mapa.	87
5.15	Resultados estatísticos das <i>Config2_OF_F1F2</i> e <i>Config5_OF_F1F2WD</i> , em condições de falhas da odometria.	93
5.16	Parâmetros comuns a todas as configurações do algoritmo AMCL que serão utilizadas nesta secção.	106
5.17	Definição da <i>Config1_OA_001</i> e <i>Config2_OA_02</i>	108
5.18	Resultados estatísticos das <i>Config1_OA_001</i> e <i>Config2_OA_02</i> , num ambiente sem <i>outliers</i>	108
5.19	Definição da <i>Config3_OA_05</i> e <i>Config4_OA_05L</i>	111
5.20	Resultados estatísticos das <i>Config3_OA_05</i> e <i>Config4_OA_05L</i> , com grande quantidade de obstáculos no mapa.	111

5.21 Resultados estatísticos das <i>Config1_OA_001</i> , <i>Config2_OA_02</i> e <i>Config3_OA_05</i> , com erros forçados de odometria, em dois momentos.	116
5.22 Resultados estatísticos das <i>Config1_OA_001</i> , <i>Config2_OA_02</i> e <i>Config4_OA_05L</i> , num ambiente com alguns obstáculos.	119
5.23 Resultados estatísticos das <i>Config1_OA_001</i> e <i>Config2_OA_02</i> , num ambiente com grande número de obstáculos.	123
5.24 Resultados estatísticos das <i>Config3_OA_05</i> e <i>Config4_OA_05L</i> , num ambiente com grande quantidade de <i>outliers</i>	126
5.25 Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas para o ambiente sem <i>outliers</i>	127
5.26 Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas, com erros de odometria.	127
5.27 Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas para o ambiente com poucos <i>outliers</i>	127
5.28 Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas para o ambiente com muitos <i>outliers</i>	128
5.29 Tempos de ciclo médio, máximo e mínimo para o AMCL (com 300 partículas) e para o Perfect Match.	131
5.30 Tempos de ciclo médio, máximo e mínimo para o AMCL (com 200 partículas) e para o Perfect Match.	132
5.31 Tempos de ciclo médio, máximo e mínimo para o AMCL (com 100 partículas) e para o Perfect Match.	133
5.32 Tempos de ciclo médio, máximo e mínimo para o AMCL (com 50 partículas) e para o Perfect Match.	133
5.33 Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), num ambiente livre de obstáculos.	136
5.34 Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), num ambiente com alguns <i>outliers</i>	136
5.35 Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), num ambiente com grande quantidade de <i>outliers</i>	139
5.36 Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), perante erros forçados de odometria.	141

Abreviaturas e Símbolos

AGV	Automated Guided Vehicle
AMCL	Adaptive Monte Carlo Localization
API	Application Programming Interface
DC	Direct Current
EKF	Extended Kalman Filter
FEUP	Faculdade de Engenharia da Universidade do Porto
GUI	Graphical User Interface
IMU	Inertial Measurement Units
INESCTEC	Instituto de Engenharia de Sistemas e Computadores - Tecnologia e Ciência
IP	Internet Protocol
KF	Kalman Filter
KLD	Kullback–Leibler Divergence
LED	Light Emitting Diode
PC	Personal Computer
PM	Perfect Match
MCL	Monte Carlo Localization
RAM	Random Access Memory
RPROP	Resilient Backpropagation
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UKF	Unscented Kalman Filter
USB	Universal Serial Bus
XML-RPC	Extensible Markup Language Remote Procedure Call

Capítulo 1

Introdução

1.1 Contexto

A utilização de robôs em aplicações de vigilância, publicidade, entretenimento, logística, guia de museus ou outro tipo de serviços ainda tem alguns entraves. Estes podem começar no custo do robô (quer de compra, quer de manutenção, configuração e colocação em serviço), estar relacionados com a sua baixa flexibilidade (capacidade de ser utilizado em diferentes aplicações), mas também pela baixa autonomia e necessidade de períodos longos de carga da bateria durante os quais está inoperacional [6, 7, 8].

1.2 Motivação e Objetivos

Nesta dissertação pretende-se contribuir para o desenvolvimento de uma plataforma móvel robotizada que responda aos pontos focados anteriormente. Nesse sentido será desenvolvido um sistema que possibilite ao robô a mudança de bateria de uma forma automática, ou seja, sem a intervenção de nenhum operador, e sempre que o robô detete que a carga da bateria está a chegar ao fim. Desta forma poderá operar continuamente e sem tempos de paragem.

Por outro lado, pretende-se desenvolver um *software* de configuração do sistema (*e.g.*, auto-localização, navegação, ações a tomar, etc.) que possibilite a utilização da base do robô em diferentes aplicações que podem passar pela logística, publicidade, guia, vigilância ou outras. Para este tipo de aplicações é necessário que o robô possua métodos de localização destinados principalmente a ambientes dinâmicos, nos quais há um acréscimo na dificuldade de auto-localização e uma necessidade adicional da robustez dos algoritmos. Isto é devido essencialmente ao elevado número de pessoas que partilham a mesma área do robô e às constantes mudanças na localização da mobília (*e.g.*, cadeiras, mesas) no ambiente de ação; perturbando os dados adquiridos pelos sensores do robô e, consequentemente, havendo uma maior incerteza da sua auto-localização [2]. Adicionalmente, pretende-se utilizar métodos de localização que aproveitem preferencialmente as características naturais do ambiente em que o robô se encontra, sem a implementação de estruturas artificiais que, apesar de poderem facilitar a sua auto-localização, podem tornar-se pouco

flexíveis para os tipos de aplicações pretendidas [9, 10]. Para além do que já foi referido, o foco desta dissertação passará pela utilização de robôs relativamente baixos, possuindo a capacidade de se movimentar por baixo de algumas peças de mobília (*e.g.*, mesas, prateleiras, cadeiras). Consequentemente, todos os seus sensores também estarão num nível baixo, podendo trazer algumas limitações acrescidas na perceção do meio envolvente.

1.3 Contribuições inovadoras

Este projeto de dissertação focou-se especialmente no problema da auto-localização de pequenos robôs móveis, apenas com um laser frontal perto do chão e com um raio de deteção de cerca de 180°. Em adição, devido às suas pequenas dimensões e baixa altura não possuiu espaço suficiente para comportar uma maior quantidade de baterias ou baterias de maiores dimensões, tendo portanto de se utilizar um computador com baixo poder de processamento e baixo consumo energético, com vista a não comprometer a autonomia do robô. Assim, é de grande importância que os algoritmos de localização possuam o maior nível de otimização e fiabilidade possível, tendo-se por isso realizado estudos comparativos entre o algoritmo "state of the art" do ROS, o Adaptive Monte Carlo Localization (AMCL) [2, 11, 12, 13], muito popular e largamente utilizado pela comunidade científica, mas que se torna computacionalmente pesado e com tempos de ciclo elevados quando utiliza um elevado número de partículas e o Perfect Match, que em contrapartida é um algoritmo leve e rápido, inicialmente criado para ser utilizado no futebol robótico [3], mas que depois foi generalizado para outros tipos de aplicações [4, 5].

Neste campo, os problemas que advêm destas características físicas e de equipamento não estão resolvidos de forma comprovadamente robusta. Os casos documentados até à data utilizam um maior número de sensores ou com ângulos de visão superiores (geralmente 360°) [14], por norma a alturas superiores, longe do chão, ou utilizam computadores mais poderosos, permitindo recorrer a um elevado número de partículas, no caso da utilização do filtro de partículas, de modo a conseguir-se uma solução minimamente robusta para o problema da auto-localização.

Com isto em vista, estudou-se e desenvolveu-se um algoritmo de fusão e supervisão de diferentes métodos de localização, executados em paralelo ao longo da movimentação do robô, cujo objetivo é detetar situações anómalas: excesso de *outliers* e início de perda de localização, resultando num sistema de localização robusto, fiável e preciso para as circunstâncias anteriormente referidas (limitações da estrutura e do *hardware* do robô). Ao comparar o *output* dado por cada um destes algoritmos e validá-los entre si, juntamente com outros mecanismos de localização relativa (*e.g.*, odometria, IMUs), é possível detetar situações anómalas e corrigi-las. A integração de outro método baseado num paradigma diferente que possui outro tipo de características e pontos fortes, em adição com um processo de supervisão que garanta a sua coexistência e detecte possíveis erros é uma ideia promissora e que merece ser explorada.

Noutro campo, criou-se ferramentas com vista à implementação de um mecanismo de troca automática de baterias para este tipo de robôs móveis: de pequenas dimensões e para aplicação em serviços, vigilância, publicidade, etc. Geralmente, os mecanismos de troca automática de bateria

só são atualmente aplicados a AGVs industriais, sendo esta uma solução pouco difundida e pouco utilizada por outros tipos de robôs móveis. Para este caso em concreto e para grande parte de outras aplicações, é de grande relevância que os robôs consigam operar autónoma e continuamente, sem a necessidade da intervenção de operadores, ou tempos de paragem para o carregamento das baterias.

1.4 Estrutura da dissertação

Esta dissertação está organizada em 7 capítulos. No Capítulo 2 fez-se uma análise geral dos métodos de localização existentes, apresentando-se também algumas soluções relacionadas com o carregamento e troca automática de baterias aplicadas em robôs móveis. No Capítulo 3 fez-se uma descrição detalhada de todo o *hardware* e *software* utilizado e desenvolvido para o robô Clever, assim como algumas das possíveis aplicações desta plataforma robótica. No Capítulo 4 apresentou-se as ferramentas desenvolvidas para o sistema de troca automática de baterias. O Capítulo 5 focou-se na comparação entre dois algoritmos de localização: AMCL e Perfect Match. Por fim, o Capítulo 6 expõe as conclusões desta dissertação e no Capítulo 7 estão apresentadas algumas sugestões para trabalho futuro.

Capítulo 2

Revisão bibliográfica

2.1 Métodos de localização

A auto-localização em robôs móveis exprime um papel fundamental na sua autonomia, interação e desempenho das tarefas a eles destinadas. Especialmente em robôs de serviços é importante que se localizem de forma robusta e com flexibilidade para lidar com situações inesperadas. Por outro lado, um fator chave para o sucesso passa por uma interação apropriada com os objetos e pessoas presentes no ambiente em que se encontram. Os métodos de localização podem basear-se na localização relativa, a partir de uma determinada posição inicial conhecida ou, por outro lado, podem destinar-se a localizar o robô no referencial global do mapa. Numa outra vertente, os métodos de localização probabilísticos podem estimar tanto a posição local do robô num determinado subespaço do mapa, como a sua posição absoluta. Para a aplicação de robôs destinados a interagir em ambientes dinâmicos, focar-se-á a análise essencialmente nos métodos de localização probabilística, de ajuste de mapa e nas técnicas mais comuns da localização relativa, pois estas últimas têm sempre um papel importante e complementar em conjugação com os outros tipos de métodos.

2.1.1 Métodos de localização relativa

As técnicas utilizadas para a localização relativa baseiam-se na determinação da posição e orientação do robô, de forma incremental, a partir de um determinado ponto inicial. Este método utiliza sensores a bordo do robô, para receber informações internas, sendo principalmente utilizados *encoders*, acelerómetros, giroscópios, etc. Como estes tipos de técnicas não utilizam observações do ambiente, mas apenas sensores internos ao robô (interocetivos), as estimativas realizadas terão sempre erros acumulados. Por norma, devido às suas limitações, estes métodos são integrados com técnicas de localização exterocetivas.

2.1.1.1 Odometria e *Dead Reckoning*

Quando se pretende obter uma estimativa da posição relativa do robô através de sensores interocetivos, são utilizadas essencialmente a odometria e o *dead reckoning*. A odometria é a solução

mais simples, utilizando equações da cinemática de robôs móveis para estimar a sua posição. Estas equações utilizam a integração da velocidade do robô ao longo do tempo por forma a calcular a posição e orientação atual [15]. Para se obter a velocidade, utilizam-se *encoders* cuja função é contabilizar o número de rotações das rodas, em cada ciclo. As integrações sucessivas têm o efeito negativo de aumentar o erro de posição e orientação do robô ao longo do tempo, de forma proporcional à distância percorrida. Por outro lado, a utilização da odometria tem custos reduzidos, proporciona amostragens a muito altas frequências e tem boa precisão se for utilizada para distâncias curtas. Estas características são suficientes para que esta técnica seja a mais comum entre os métodos de navegação e localização utilizada em robôs móveis. Como extensão da odometria, o *dead reckoning* utiliza a velocidade angular e a aceleração para obter uma melhor estimativa da posição do robô, através de IMUs (*Inertial Measurement Units*) [9].

2.1.2 Localização baseada em marcos e balizas

A utilização de marcos artificiais e balizas para a localização de robôs móveis pode ser uma boa solução para o problema da localização global. Inicialmente é importante fazer uma distinção entre marcos (*landmarks*) e balizas (*beacons*). A diferente classificação baseia-se essencialmente nos métodos utilizados para a determinação da posição [9, 10].

- Quando se recorre a técnicas como a triangulação e a trilateração, os marcadores simultaneamente envolvidos são classificados como balizas. Este tipo de localização é contínuo ao longo dos vários percursos do robô, mantendo sempre, pelo menos, uma das balizas no campo de deteção dos sensores do robô.
- No caso da utilização de marcos, o robô reconhece-os ao longo dos seus percursos, localizando o robô em relação às suas posições, já previamente conhecidas. Contrariamente à localização baseada em balizas, nem sempre existem dados dos sensores do robô referentes à deteção de algum dos marcos num dado instante de tempo, tornando a sua localização intermitente e limitando as trajetórias do robô apenas a áreas onde estes existam.

2.1.2.1 Localização baseada em marcos

A partir do que já foi exposto, também é importante distinguir entre marcos naturais e marcos artificiais. Como é intuitivamente perceptível, os marcos naturais englobam todas as características do ambiente já existentes e cuja função não está relacionada com a navegação do robô (*e.g.*, formas geométricas correspondentes às paredes, portas, esquinas). Por outro lado, os marcos artificiais são colocados no ambiente que o robô vai frequentar, com o objetivo de possibilitar e facilitar a sua navegação (*e.g.*, etiquetas com código de barras, estruturas com cores ou formas geométricas facilmente distinguíveis) [9].

As técnicas baseadas em marcos possuem um elevado grau de precisão sempre que estes se encontram dentro do raio de alcance dos sensores do robô, sendo, no entanto, necessário outro tipo de estratégia de localização quando não estão. Geralmente é utilizada a odometria ou o *dead*

reckoning para que a posição do robô seja continuamente estimada entre cada zona de marcos artificiais. Como já foi descrito, a utilização de métodos de localização incrementais aumenta a incerteza da posição ao longo do tempo, sendo necessário corrigi-la quando o robô volta a dispor de marcos no seu campo de visão [16]. Para a fusão dos dados recebidos pelos vários tipos de sensores, recorre-se normalmente ao Filtro de Kalman (ou variantes do mesmo), sendo possível fazer uma integração ponderada e modelar os ruídos provenientes de cada processo de medição [1, 17].

Para este tipo de aplicações, é comum utilizarem-se sistemas baseados em visão por forma a ser possível distinguir e identificar facilmente os diversos marcos ao longo do mapa, proporcionando uma localização mais precisa [18]. Deste modo, conhecendo-se as diferentes características dos marcos (*e.g.*, cor, forma, dimensões) e as suas posições no referencial global (registadas previamente no mapa), torna-se fácil a estimação da localização do robô [19].

As maiores desvantagens dos sistemas de visão resultam da necessidade de mais recursos computacionais, sendo necessário o processamento de maiores quantidades de informação e das câmaras serem mais caras que outros tipos de sensores [9].

2.1.2.2 Localização baseada em balizas

As balizas também podem ser classificadas em dois grupos: balizas ativas e balizas passivas. As balizas ativas possuem uma fonte de energia própria, podendo ser emissoras, recetoras ou simultaneamente emissoras e recetoras. Como já foi referido, diretamente relacionado com a localização utilizando balizas, esta pode ser obtida através de técnicas de triangulação, trilateração ou de ambas em simultâneo [9, 10]. A triangulação é baseada na medição de ângulos simultaneamente em relação a, normalmente, três ou mais pontos de referência (neste caso, balizas) – Figura 2.1. Por outro lado, a trilateração baseia-se na medição das distâncias (ou diferenças de distâncias).

Como aplicação deste tipo de técnicas, algumas abordagens, com recurso a sensores laser (angulares), utilizam balizas refletoras, que retornam feixes de luz fortes, utilizados para a deteção do ângulo do robô em relação a cada baliza, no plano horizontal [1]. As localizações dos refletores são guardadas na memória do robô, a priori, representando-as através das coordenadas em relação ao referencial global. Para localizar o robô, é utilizada a técnica de triangulação usando pelo menos três dos refletores identificados [17, 20].

Os métodos de localização baseados em marcadores artificiais (marcos e balizas) têm a grande desvantagem de necessitar de modificações físicas do ambiente onde o robô vai atuar, tornando-a numa solução pouco apelativa para diversas aplicações com requisitos de flexibilidade, aspeto visual, limitações orçamentais e limitações nas características geométricas do local. Para além disso, por forma a ser possível obter uma localização precisa, é necessário um número de marcadores proporcional às dimensões do espaço, podendo tornar-se inviável [16, 21, 1].

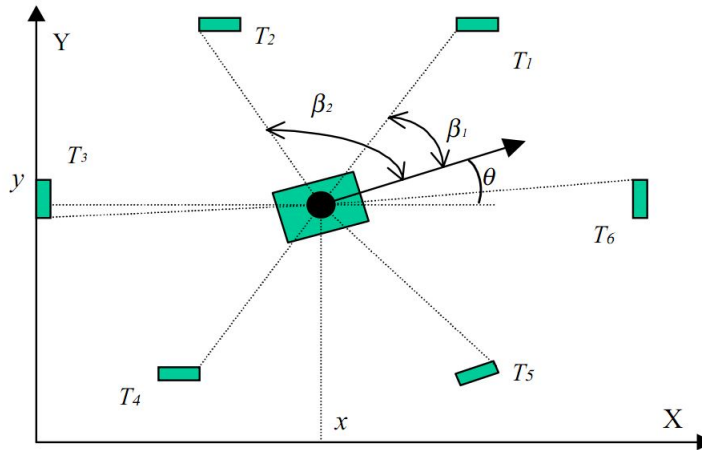


Figura 2.1: Exemplo de triangulação relativa a marcos [1].

2.1.3 Métodos de localização probabilística

As técnicas de localização probabilística estimam a posição do robô através da integração entre os dados medidos no momento atual e o anterior conhecimento do estado do sistema e dos sensores de medição, para que o erro seja minimizado estatisticamente. Ao aplicar este tipo de métodos, é necessário o conhecimento anterior do mapa do ambiente, das distribuições iniciais de probabilidade e dos modelos estatísticos, tanto dos sensores como do movimento do robô. As secções seguintes incidirão essencialmente na apresentação do Filtro de Bayes, na análise dos filtros de Kalman (KF) e da localização de Monte Carlo (MCL), assim como as suas variantes.

2.1.3.1 Filtro de Bayes

Genericamente, o problema da localização de robôs pode ser formulado como um problema de estimação de estado. Neste contexto x_t , o estado do sistema, é tipicamente especificado pela posição do robô num espaço cartesiano bidimensional e pela orientação do robô. As medições z_t incluem medidas de sensores exteroceptivos (*e.g.*, *laser range finders*, sonares, imagens de câmara), e as informações de controlo u_t geralmente consistem nas leituras de odometria do robô [22]. O próximo estado de probabilidade descreve como a posição do robô muda com base apenas nas informações recolhidas pelos *encoders* nas rodas do robô (odometria). Esta probabilidade condicional é tipicamente um modelo de cinemática do robô, com incerteza adicional. O modelo perceptual descreve a probabilidade de fazer uma dada observação z_t dado que o robô está na posição x_t [23].

O algoritmo de Bayes calcula a distribuição de probabilidade a partir das medidas dos sensores e do estado do robô [2]. Para ser calculada a confiança de uma determinada estimativa no instante t , utiliza-se a confiança no instante $t - 1$ juntamente com as medidas e o estado de controlo mais recentes, de forma recursiva. O filtro de Bayes baseia-se no princípio de Markov, *i.e.*, que a

localização do robô é o único fator no ambiente que sistematicamente influencia os dados lidos dos sensores [23].

Está exposto de forma representativa na Figura 2.2 o pseudo-algoritmo do filtro de Bayes:

```

1:  Algorithm Bayes_filter( $bel(x_{t-1}), u_t, z_t$ ):
2:      for all  $x_t$  do
3:           $\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx$ 
4:           $bel(x_t) = \eta p(z_t | x_t) \overline{bel}(x_t)$ 
5:      endfor
6:      return  $bel(x_t)$ 

```

Figura 2.2: Pseudo-algoritmo do filtro de Bayes [2].

Na linha 3 calcula-se a estimativa da confiança do estado, no momento t ($\overline{Bel}(x_t)$), processando o controlo no instante t (u_t) e a confiança anterior, em $t - 1$ ($\overline{Bel}(x_{t-1})$). Mais especificamente, a estimativa da confiança é calculada através do integral entre o produto da distribuição atribuída a x_{t-1} (do ciclo anterior) e a distribuição de probabilidade de, dado u_t , obter-se o estado x_t a partir de x_{t-1} . Este passo é classificado como “fase de predição”.

Na linha 4 é calculada a confiança final, a partir da estimativa da confiança do estado (da linha anterior) multiplicada pela probabilidade da medição z_t ter sido observada para cada estado x_t - “fase de atualização”. O resultado é depois normalizado através da multiplicação pela constante para se obter um valor em probabilidade (entre 0 e 1).

A confiança final é então retornada na última linha do algoritmo. Para que seja possível prever a confiança posterior de forma recursiva, é necessário fornecer uma estimativa inicial do estado x_0 , sendo comum optar-se por uma de duas práticas: centrar toda a probabilidade num ponto específico, caso se tenha certeza total da localização inicial; ou distribuir uniformemente a probabilidade por todo o domínio de , caso se desconheça na totalidade este valor inicial.

Demonstração matemática¹ A ideia-chave da filtragem de Bayes é estimar a densidade de probabilidade sobre o espaço de estados tendo em conta os dados recebidos [11]. Esta probabilidade posterior é normalmente denominada de confiança e é denotada:

$$Bel(x_t) = p(x_t | d_{0:t}) \quad (2.1)$$

Utilizando como as informações disponível desde o instante 0 até ao instante t . Para aplicações em robôs móveis distinguem-se as informações de controlo (u_t) e as informações externas, de medição (z_t), ficando:

$$Bel(x_t) = p(x_t | z_{t-1}, u_t, z_{t-2}, u_{t-1}, \dots, u_0) = p(x_t | z_{1:t}, u_{1:t}) \quad (2.2)$$

¹Baseado em [2].

Aplicando a regra de Bayes:

$$p(x_t | u_{1:t}, z_{1:t}) = \frac{p(z_t | x_t, z_{1:t-1}, u_{1:t})p(x_t | z_{1:t-1}, u_{1:t})}{p(z_t | z_{1:t-1}, u_{1:t})} \quad (2.3)$$

$$p(x_t | u_{1:t}, z_{1:t}) = \eta p(z_t | x_t, z_{1:t-1}, u_{1:t})p(x_t | z_{1:t-1}, u_{1:t}) \quad (2.4)$$

Considerando que o estado x_t está completo, sem necessidade de informações anteriores ao instante t , expressa-se a seguinte independência condicional:

$$p(z_t | x_t, z_{1:t-1}, u_{1:t}) = p(z_t | x_t) \quad (2.5)$$

Simplificando 2.4 a partir da relação 2.5 fica:

$$p(x_t | z_{1:t}, u_{1:t}) = \eta p(z_t | x_t)p(x_t | z_{1:t-1}, u_{1:t}) \quad (2.6)$$

Ao relacionar a igualdade de 2.2 com 2.6 obtém-se:

$$Bel(x_t) = \eta p(z_t | x_t) \overline{Bel}(x_t) \quad (2.7)$$

Com

$$\overline{Bel}(x_t) = p(x_t | z_{1:t-1}, u_{1:t}) \quad (2.8)$$

Isto pode ser expandido como o integral:

$$\overline{Bel}(x_t) = \int p(x_t | x_{t-1}, u_t) p(x_{t-1} | z_{1:t-1}, u_{1:t-1}) dx_{t-1} \quad (2.9)$$

Obtendo-se as expressões já demonstradas no pseudo-algoritmo de Bayes da Figura 2.2, ficando a demonstração completa [12].

2.1.3.2 Localização Monte Carlo (MCL)

A Localização de Monte Carlo (também conhecida como filtro de partículas) é um algoritmo de localização baseado na formulação de Bayes, que estima a posição do robô a partir de um conjunto de hipóteses (partículas) distribuídas e ponderadas num determinado ambiente [11]. As partículas de estado no mapa começam a ser distribuídas aleatoriamente e terão as suas posições e pesos alterados de acordo com as sucessivas leituras dos sensores [24]. O robô estima a sua posição a partir da área do mapa com maior concentração de partículas.

Os modelos de representação de partículas têm uma série de características importantes [11]:

- Concentram os recursos computacionais em áreas que são mais relevantes, por amostragem proporcional à probabilidade posterior.
- Pode acomodar características de sensores, dinâmicas de movimento, e distribuições de ruído praticamente arbitrárias.

- Ao controlar o número de amostras online, pode adaptar-se aos recursos computacionais disponíveis.
- Os filtros de partículas possuem uma implementação fácil, tornando-os uma boa solução para a localização de robôs móveis.

Por outro lado, há algumas desvantagens e limitações deste tipo de abordagens:

- Se o tamanho do conjunto da amostra for pequeno, um robô bem localizado pode perder a estimativa da sua posição pelo facto do algoritmo não conseguir gerar uma amostra no local certo.
- Para o caso de o robô ser raptado, o algoritmo MCL regular também não é apropriado, visto que podem não haver amostras distribuídas nas proximidades da nova posição do robô, sendo impossível a sua recuperação.
- Para além disso, é verificada uma grande degradação na eficácia deste método no caso de os sensores serem muito precisos. No limite, o algoritmo MCL básico irá falhar se os sensores não possuírem qualquer ruído.
- A qualidade da solução aumenta com o número de partículas.

Algoritmo MCL² Como já foi visto, os algoritmos de filtros de partículas representam a confiança de um determinado estado do robô através de um conjunto de N amostras distribuídas de acordo com essa mesma confiança:

$$Bel(x_t) \approx S_t = \{x_t^i, w_t^i\}_{i=1, \dots, N} \quad (2.10)$$

Nesta expressão cada x_t^i representa um estado possível do robô, da partícula i , no instante t e w_t^i é um número não negativo que caracteriza o peso (importância) da partícula i , no instante t .

De forma recursiva, é realizado:

- Fase de predição – parte-se inicialmente do conjunto de partículas da iteração anterior S_{t-1} , aplicando-se o modelo da dinâmica do robô para cada partícula s_{t-1} , amostrando da função de densidade $p(x_t | s_{t-1}^i, u_{t-1})$, *i.e.*,

– Para cada partícula s_{t-1}^i , retira-se uma amostra $s_t^{'i}$ de $p(x_t | s_{t-1}^i, u_{t-1})$.

Assim, obtém-se um novo conjunto de partículas S_t' que aproxima as amostras de $p(x_t | z_{t-1})$.

- Fase de atualização – aqui tem-se em consideração a medição dos sensores do robô, z_t , e atribui-se um peso w_t^i a cada uma das partículas de acordo com a probabilidade da partícula ter aquele estado $s_t^{'i}$ sabendo a medida z_t .

²Baseado em [11, 12, 2, 13].

- Para cada partícula, retira-se uma amostra s_t^i de S_t , de $\{s_t^i, w_t^i\}$.

Obtém-se então S_t após a amostragem do conjunto pesado $\{s_t^i, w_t^i\}$.

- Por fim, entra-se na fase de re-amostragem, que elimina as partículas com um peso abaixo de um determinado limiar e cria novas amostras perto das que têm maior probabilidade.

Para a inicialização do filtro, é necessário atribuir um conjunto aleatório S_0 inicial, a partir da probabilidade $p(x_0)$.

2.1.3.3 Melhoramentos do algoritmo MCL

Random Particle MCL Como foi exposto anteriormente, o algoritmo MCL regular possuiu algumas limitações que, para algumas aplicações, não são desprezáveis. Uma dessas limitações está relacionada com o evento de rapto, *i.e.*, alterar manualmente a posição do robô enquanto este está em funcionamento, perturbando gravemente o processo de auto localização devido a uma “teleportação” inesperada. Tendo em conta que o algoritmo MCL regular concentra gradualmente as suas partículas na posição com maior probabilidade para a localização do robô, verifica-se que as outras zonas deixarão completamente de ter partículas “sobreviventes”, tornando então impossível a sua recuperação, no caso de rapto. Este comportamento é agravado nas situações em que há uma grande área para a distribuição das partículas e no caso de se utilizar um número reduzido de partículas para o algoritmo [2].

Com o objetivo de colmatar estas limitações, utilizam-se algumas técnicas heurísticas simples. Na base do algoritmo Random Particle MCL está a ideia da injeção de partículas de forma aleatória com uma distribuição relacionada com a precisão da estimação da localização. Essa relação é atingida a partir da monitorização da probabilidade das medidas dos sensores, relacionando-a com a probabilidade média das medidas. Deste modo, focalizam-se os esforços computacionais nas regiões que realmente importam, evitando o sobrecarregamento desses recursos disponíveis [2].

KLD-Sampling: Adaptive Particle Filter Esta abordagem foi desenvolvida como forma de aumentar a eficiência e diminuir os esforços computacionais da utilização dos filtros de partículas. A utilização de um número constante de amostras desde o início do algoritmo pode tornar-se desajustado e ineficiente, visto que a complexidade das densidades de probabilidade pode alterar-se consideravelmente ao longo do tempo [12]. Durante a localização global, o robô desconhece em completo a sua posição atual, por isso, as possibilidades da sua localização cobrem uniformemente todo o espaço. Por outro lado, na fase de seguimento de posição, a incerteza é geralmente pequena e frequentemente focada em espaços de dimensões inferiores. Assim, são necessárias mais amostras durante a localização global, de forma a ser possível aproximar a densidade real com maior precisão, das que são necessárias apenas para o seguimento de posição [25]. Baseando-se neste princípio, aplica-se um método cuja ideia fundamental passa pela limitação do erro introduzido pela representação à base de amostras do filtro, utilizando-se um número reduzido de partículas se a densidade estiver concentrada numa pequena área do espaço, e aumentando-se o número de

amostras no caso da incerteza em relação ao estado ser elevada. Este algoritmo apresenta uma baixa complexidade de implementação e um mais reduzido peso computacional [12].

Mixture-MCL Esta técnica surgiu para compensar as falhas suscetíveis de acontecer no caso de se utilizarem sensores com pouco ruído, ou no limite, sem ruído. Uma das possíveis soluções para este problema seria aumentar artificialmente a dimensão de ruído dos sensores, podendo ser interpretada como a incerteza intrínseca à natureza aproximada do filtro de partículas [25]. Por outro lado, pode optar-se por uma solução mais consistente, em que se altera o processo de amostragem de uma fração das partículas [2, 11]. Basicamente, esta técnica é a combinação entre uma amostragem regular e o oposto do MCL, invertendo o processo de amostragem. Enquanto os algoritmos MCL básicos primeiro preveem uma nova posição usando odometria e em seguida utilizam as medições do sensor para avaliar a relevância dessa amostra, o Mixture-MCL estima a posição usando as medições mais recentes dos sensores e, em seguida, utiliza a odometria para avaliar a conformidade deste palpite com a anterior confiança do robô, tendo também em conta os dados anteriores da odometria. O objetivo deste algoritmo é incorporar duas metodologias que em combinação se complementam e apresentam bons resultados. No entanto, este método só por si poderá ser ineficiente uma vez que a geração de partículas é realizada ignorando toda a história, *i.e.*, utilizando apenas as mais recentes informações dos sensores. Adicionalmente, o cálculo das duas principais fases do algoritmo (geração de partículas através do modelo de medição e o cálculo do peso das partículas) pode ser difícil de realizar, sendo necessário recorrer a métodos de aproximação (*e.g.*, *density trees*) e, conseqüentemente, aumentar a complexidade da sua implementação. Esta técnica consegue colmatar algumas das desvantagens da utilização de algoritmos MCL básicos, pois consegue estimar de forma bem-sucedida a posição do robô com um conjunto de amostras pequena, recupera rapidamente de raptos e tem um bom funcionamento no caso de os sensores serem demasiado precisos [2].

2.1.3.4 Filtros de Kalman

O filtro de Kalman (KF) [26] é um algoritmo recursivo baseado nos filtros gaussianos, cuja ideia principal é a de que a confiança num determinado conjunto de valores é representada por uma distribuição normal constituída por múltiplas variáveis [2]. Em cada instante t , a confiança é caracterizada por dois fatores: a média (μ_t) e a covariância (σ_t). Esta é uma das técnicas mais conhecidas e utilizadas para a fusão de informação, combinando uma fase de predição de estado com os dados recebidos dos sensores, calculando uma média ponderada cujo grau de certeza de cada valor está diretamente relacionada com a sua covariância [27]. O facto deste ser um algoritmo recursivo, *i.e.*, só necessitar da estimativa obtida no ciclo anterior e não de todo o seu histórico temporal, torna-o numa técnica muito eficiente computacionalmente e facilmente aplicado em processos de tempo real [2, 28, 29]. O filtro de Kalman é um estimador ótimo, garantindo uma minimização do erro de estimação a partir da medição de grandezas ao longo do tempo, perturbadas por ruídos. De forma a operar no sentido da convergência para uma solução ótima é necessário

que o sistema seja linear e que os ruídos associados sejam sequências de ruído branco gaussiano, *i.e.*, independentes e com média nula [30].

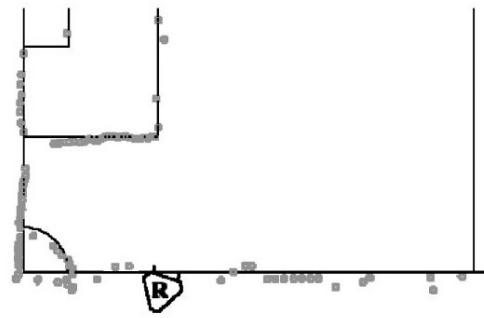
De forma a ultrapassar os problemas e as limitações do filtro de Kalman, visto que nos problemas reais raramente se verifica a premissa de que as medições e as transições de estados são funções lineares, surgiu o Extended Kalman Filter (EKF). Esta variante permite a expansão do filtro de Kalman para aplicações com sistemas não lineares, usando uma aproximação linear da estimativa atual, através da aplicação de derivadas parciais [2, 31]. O EKF, devido à sua simplicidade de implementação e eficiência computacional, é das ferramentas mais populares para a resolução dos problemas de estimação de estados aplicados à robótica [2]. Contudo, esta técnica, ao contrário do KF não é, no geral, um estimador ótimo e possui limitações. A eficácia com que aproxima as soluções está relacionada tanto com o grau das não linearidades como com o grau de incerteza no estado. Assim, o EKF pode não garantir a convergência das estimativas ou até pode convergir para resultados errados [32].

Outra das principais variantes do KF é o Unscented Kalman Filter (UKF) [33] que foi desenvolvido como alternativa ao EKF, também para aplicações de sistemas não lineares. Este método não utiliza a mesma estratégia de linearização implementada no EKF. Em contrapartida, o algoritmo do UKF aplica uma função matemática denominada de *unscented transform* que utiliza um conjunto de pontos específicos para parametrizar as médias e as covariâncias das distribuições de probabilidade. Esta abordagem, teoricamente, apresenta melhores aproximações do que o EKF, sem a necessidade de calcular os Jacobianos das matrizes [2, 33, 34]. Ainda assim, nem sempre é a solução mais adequada para algumas aplicações [35, 36, 37].

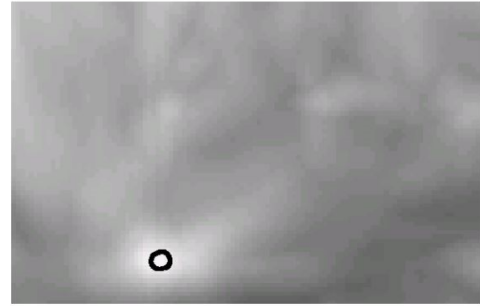
2.1.4 Métodos de ajuste de mapa

2.1.4.1 Perfect Match

O Perfect Match é um algoritmo de auto-localização que surgiu como resposta a algumas dificuldades no processo da auto-localização, de uma forma rápida e pouco pesada computacionalmente, de robôs móveis, criado especificamente para ser aplicado no futebol robótico (“Robocup Midsized league”) [3], mas tendo o potencial de ser generalizado para outro tipo de ambientes (e aplicações) [4]. Como objetivos principais deste método, é necessário que o algoritmo resulte num processo de localização robusto, com uma estimação precisa da posição e que seja computacionalmente eficiente. Por outro lado, não são utilizados marcos artificiais para a localização global do robô, estando apenas limitado às características próprias do ambiente (*e.g.*, linhas, paredes, etc). O Perfect Match faz a modelação do processo de localização como um problema de minimização de erros, utilizando um minimizador numérico eficiente – Resilient Propagation (RPROP). Outro fator importante é o de o mapa ser representado por uma grelha de ocupação, sendo possível calcular-se antecipadamente e uma única vez a matriz com as distâncias de cada célula aos obstáculos, assim como as respetivas matrizes de gradiente (na direção do x e do y) [5]. Desta forma, o tempo de processamento do algoritmo é reduzido consideravelmente - fator crucial para aplicações online.



(a) Estimação da posição e orientação do robô num campo de futebol robótico.



(b) Representação da função do erro, com um mínimo global (zona clara) bem definido, indicando a posição estimada do robô.

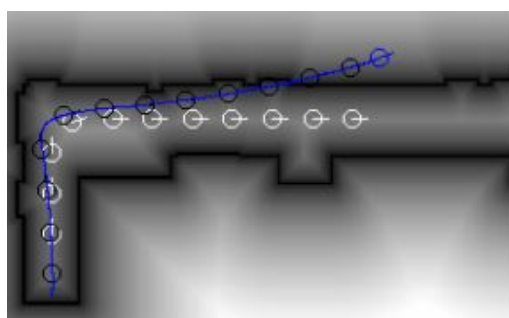
Figura 2.3: [3]

Devido às características deste método, não é propriamente adequado para ser aplicado em ambientes amplos, com grandes áreas e em que não existam características locais que ajudem a reduzir a ambiguidade [24].

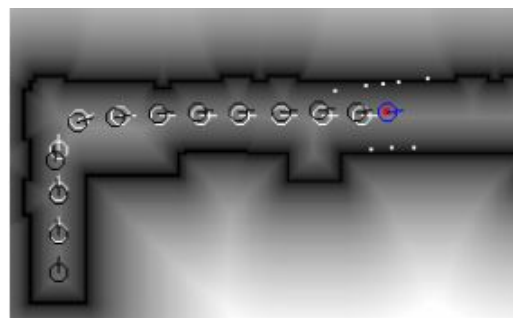
Como adaptação ao algoritmo original, para ser mais direcionado para aplicações em ambientes *indoor* e com outro tipo de sensores (*laser range finder*, sensores infravermelhos), foi proposto um método com alguns ajustes para melhor se adaptar à nova realidade, calculando-se, em cada ciclo do programa, um número fixo de alternativas adicionais (aleatórias) à melhor estimativa da posição atual do robô [4]. Após este processo, é feita a comparação entre as funções de custo das estimativas para poder selecionar-se a melhor. Por forma a evitar que a melhor estimativa esteja sempre a alterar-se a cada iteração, é colocado um requisito extra, garantindo que a melhor estimativa se mantém destacada durante um número fixo de iterações, e só nessas circunstâncias é que é adotada.

Utilizando uma boa configuração para o método Perfect Match, obtêm-se resultados similares aos dos filtros de partículas com uma diminuição muito considerável dos recursos computacionais (e consequentemente do tempo de processamento). Adicionalmente, apesar de apresentar bons resultados atuando de forma isolada para a localização de robôs móveis, a integração (*e.g.*, através do Filtro de Kalman) de outro tipo de medições (*e.g.*, odometria) pode tornar o algoritmo mais robusto [5], especialmente em situações em que ocorram medidas com demasiado ruído ou em casos de maior ambiguidade.

Adicionalmente, ao ser implementado um limiar de rejeição na função de custo deste algoritmo, é possível obter resultados mais robustos na rejeição de *outliers* [5].



(a) Utilização apenas da odometria.



(b) Utilização de 11 sensores infravermelhos.

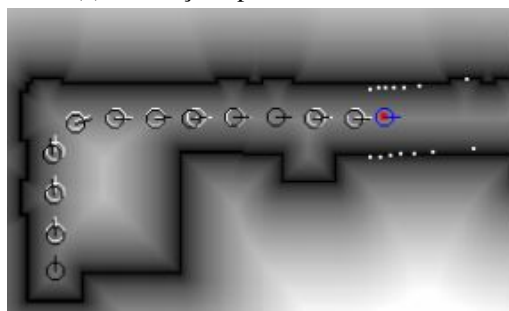
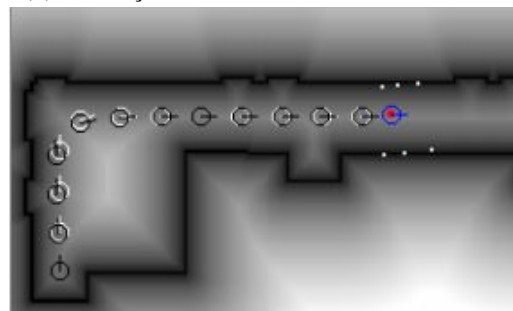
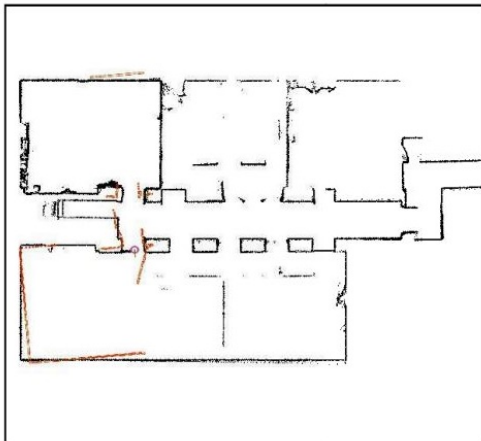
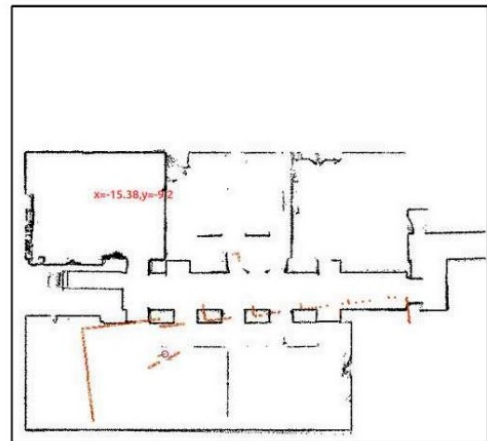
(c) Utilização de *laser range finder* considerando 13 pontos.(d) Utilização de *laser range finder* considerando apenas 7 pontos.

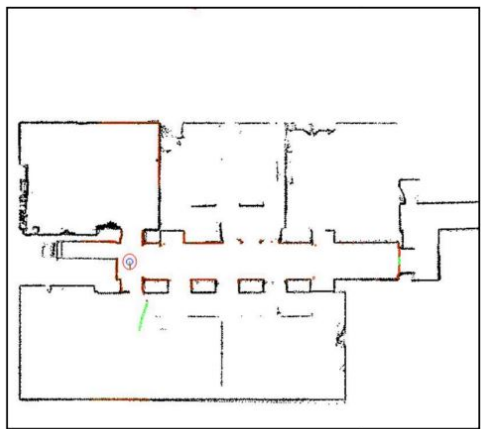
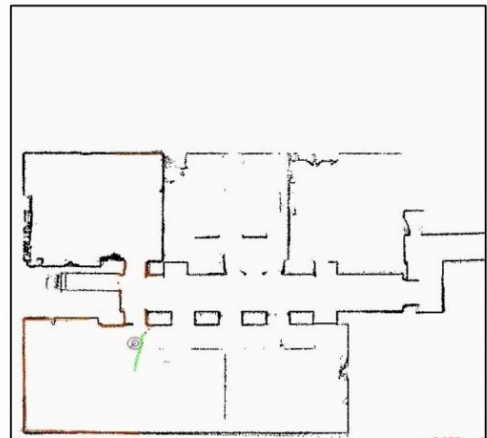
Figura 2.4: Comparação entre a posição e a orientação real do robô (branco) com as estimativas (azul) [4].



(a) Algoritmo Perfect Match original.



(b) Falha de localização devido a um obstáculo externo ao mapa.

(c) Algoritmo Perfect Match com rejeição de *outliers*.

(d) Localização bem-sucedida.

Figura 2.5: Localização do robô usando o algoritmo Perfect Match (os pontos vermelhos são utilizados para a localização) com e sem rejeição de *outliers* (pontos a verde) [5].

2.2 Visão geral dos métodos de localização para aplicação em ambientes dinâmicos

Após a análise dos diferentes métodos e estratégias de localização de robôs móveis, é possível tirar algumas conclusões quanto à aplicabilidade em vários tipos de ambientes. Especificamente é importante inferir quais os algoritmos que mais se adequam a robôs destinados a participar em locais que tenham características que possam perturbar a auto-localização de robôs móveis (*e.g.*, pessoas a passar e a posicionar-se à volta do robô, aparecimento de obstáculos inicialmente não previstos, etc.). Este tipo de algoritmos tem de apresentar uma elevada robustez, facilidade de adaptação e uma boa capacidade de distinção entre medições válidas e interferências externas ao ambiente envolvente. Adicionalmente, os métodos de localização não poderão ser baseados em características que impliquem uma alteração do ambiente (*e.g.*, implementação de marcos artificiais, de balizas, etc.), mas apenas em características próprias do local.

Os algoritmos baseados em filtro de partículas são bons candidatos a serem aplicados em ambientes dinâmicos, devido principalmente à sua grande flexibilidade e robustez na acomodação de vários tipos de ruído e boa adaptação a dinâmicas não modeladas, como é o caso. Contudo, é necessário realizar-se algumas extensões, visto que os métodos probabilísticos são implementados baseando-se no princípio de Markov, *i.e.*, que o ambiente que rodeia o robô é estático. Para contornar o problema, integra-se uma fase de pré-processamento dos dados recebidos, por forma a ser possível filtrar os resultados que não se ajustam com a realidade esperada (*outliers*) [2]. Deste modo, obtém-se um algoritmo mais robusto com a capacidade de distinção entre as medidas esperadas (de acordo com o mapa do ambiente e da sua localização passada) e as informações causadas por perturbações externas, que de outro modo poderiam influenciar drasticamente a estimação da posição do robô.

No campo dos métodos de ajuste de mapa, o algoritmo Perfect Match revela um grande potencial para ser aplicado em locais dinâmicos. Como já foi discutido anteriormente, com algumas adaptações ao algoritmo original [3] é possível utilizá-lo com diferentes tipos de sensores e em ambientes *indoor* gerais [4, 5].

A aplicação do método SLAM para a geração de um mapa do ambiente e de um mapa baseado na movimentação dos humanos provou ser bem-sucedida, através da aquisição e processamento dos dados recebidos dos sensores, extraindo as informações referentes ao ambiente estático e dinâmico. Com esse objetivo, houve a integração de vários algoritmos cuja função é identificar e diferenciar algumas características geométricas humanas em oposição a estruturas que fazem parte do ambiente estático, permitindo acompanhar o movimento das pessoas [38]. Este tipo de técnicas aumenta muito a complexidade da estimação, pois não tem apenas de acompanhar a posição do robô, mas também o movimento de uma quantidade desconhecida de humanos que partilham o mesmo ambiente, sendo um processo desnecessário para grande parte das aplicações.

2.3 Sistemas para carregamento ou troca de baterias

Outro foco de interesse para ser analisado neste capítulo está relacionado com as limitações de autonomia dos sistemas de bateria utilizados pelos robôs móveis atuais e as possíveis soluções para os contornar.

Atualmente, os robôs móveis são utilizados em diversos campos e aplicações, tais como: automação industrial, detecção de ambientes perigosos, automação de escritórios, serviços hospitalares, entretenimento, exploração do espaço, automação agrícola, automação na pesca, forças armadas, sistemas de segurança, educação, vigilância, automação de tarefas domésticas, guias de museus [39, 7, 8, 40, 6]. Um dos entraves na utilização de robôs móveis para este tipo de aplicações é o tempo de operação a que estes estão limitados, sendo obrigados a atuar em ciclos de funcionamento descontínuos. Isto está diretamente relacionado com a autonomia das suas baterias e com a dependência humana para o seu carregamento, ou troca [6]. A inclusão de baterias de alta capacidade traz associado um aumento do peso e das suas dimensões, sendo estas de grande porte. Consequentemente, leva a uma diminuição da mobilidade dos robôs e também de uma necessidade de aumento significativo das dimensões da estrutura do robô. Por estas razões, estes tipos de soluções não são utilizadas como resposta aos problemas de duração insuficiente da alimentação dos robôs [8].

Por outro lado, existem outros tipos de estratégia que resolvem ou minimizam alguns dos problemas anteriormente referidos: a utilização de estações de carregamento automático para os robôs, estações que permitam uma troca automática de baterias, ou carregamento contínuo sem fios (*e.g.*, por indução [41]).

A utilização de estações para o carregamento automático dos robôs fornece os meios para que estes não necessitem de intervenção humana nos momentos em que seja necessário trocar ou recarregar as suas baterias, aumentando a sua independência e autonomia [42].

Um sistema de carregamento foi desenvolvido em sincronismo com um sistema de teleoperação baseado na web que define e atribui remotamente tarefas para um robô [43]. O objetivo destes sistemas é garantir um ciclo de funcionamento que pode ser executado ao longo de todo o ano. Por forma a ser possível atingir esse objetivo, é necessário implementar um sistema de carregamento para o robô se tornar completamente autónomo. Este sistema possuiu uma baliza de infravermelhos, utilizada para a localização do carregador à distância; uma grelha de referência no meio da estação de carregamento, para ser detetada por laser quando o robô já estiver relativamente perto do carregador; e uma ficha elétrica abaixo da grelha de referência, onde o robô se encaixa.

Quando o nível de bateria de um robô atinge um valor baixo durante sua operação, este perde a possibilidade de ser controlado por comandos e passa a existir o risco de algum evento perigoso acontecer. Com base nesta premissa, monitorizando o nível da bateria, é possível executar um processo de auto recarregamento em tempo real quando a carga desce abaixo de um determinado limiar. Assim, em [39] desenvolveu-se um mecanismo de deteção desse momento de funcionamento, que quando é ativado, faz o robô procurar pela estação de carregamento através da localização de um marco (integrado na estrutura da estação), usando um *laser range finder*.

Uma questão importante que se coloca está relacionada com a solução utilizada para guiar o robô apropriadamente até à estação, sendo que a sua estrutura e o método de encaixe também têm um papel fundamental. A estação fornece dois pontos de contacto para se conectar com a estrutura de conexão das baterias do robô móvel, através de um dispositivo *auto-switch*. Para além disso, de forma a direccionar a inserção dos pinos de carregamento, montou-se uma vara de guia na estação de acoplamento. Durante o carregamento da bateria, o robô móvel comunica com a estação, sabendo o estado da carga em tempo real.

Numa outra vertente, para ser aplicada em robôs móveis destinados a interagir em ambientes domésticos, foi proposto um sistema que utiliza sensores infravermelhos de custos reduzidos, mas que pode operar numa ampla região, e uma estação com mecanismo de encaixe passivo com a capacidade para compensar possíveis erros de alinhamento e encaixe [40]. Quando o robô está localizado dentro da região limite da estação, o sistema direcciona o robô para se posicionar imediatamente a frente da estação, para carregamento. Como é utilizada a força do robô para se acoplar, não existe a necessidade de a estação possuir elementos ativos, ou sensores. O sistema possui um transmissor e um recetor de infravermelhos, estando o transmissor localizado na parte da frente do robô, enquanto que o recetor está integrado no centro da estação.

Especificamente para a aplicação em patrulha e segurança, foi concebido um sistema constituído por um robô móvel e uma estação de carregamento estacionária [7]. O robô tem uma câmara montada na sua parte superior, captando um sinal de vídeo que posteriormente é processado com vista a detetar um marco artificial (que indica a localização da estação de carregamento). Posteriormente, o sistema de visão orienta o movimento com a finalidade de alcançar o acoplamento. Quando o robô se aproxima da estação com uma orientação e um *offset* desajustados, a vara montada na estrutura serve como guia para garantir que os pinos são corretamente acoplados nos adaptadores. A estação possui dois grupos de pinos para o carregamento: um grupo localizado a um nível superior, responsável pelo carregamento das baterias do computador e dos circuitos de sensores; e outro grupo a um nível inferior, responsável pelo carregamento das baterias dos motores e dos drivers. Durante o carregamento, o robô está ligado por forma a acompanhar o processo e faz o desacoplamento quando este estiver concluído.

Noutro estudo [8], utiliza-se o método SLAM para construir o mapa do ambiente de ação do robô, a priori, assim como a atribuição da localização específica da estação de acoplamento para o carregamento do robô, nesse mesmo mapa. Assim, quando o robô é confrontado com a situação de bateria fraca durante a execução de uma tarefa, o robô irá mover-se em direção à estação de forma autónoma, posteriormente seguindo o percurso guia construído com fita magnética, que está localizado na entrada da estação. No entanto, existe a possibilidade de o robô entrar mal alinhado na estação, devido aos erros acumulados ou aos erros de localização. Para contornar esse problema, no momento em que os braços da estação entram em contacto com o robô, o suporte móvel roda de forma a ficar alinhado com o robô, que é detetado por um sensor. Quando termina o acoplamento, os elétrodos da estação, em contacto com o robô, fornecem a energia necessária para que sejam evitados possíveis danos no robô, durante a troca de baterias. Assim, a bateria gasta é retirada, sendo colocada numa unidade de carga vazia, e uma bateria totalmente carregada

é fornecida ao robô. Por fim, os braços da estação soltam o robô, que volta a ficar operacional para continuar a executar as tarefas que lhe foram atribuídas. O processo de troca de bateria dura cerca de 45 segundos, não necessita que o robô se desligue e o mecanismo de acoplamento permite +-9° de desalinhamento entre a estação e o robô móvel.

Num outro artigo [44], foi apresentado um conceito de um sistema de troca de baterias “hot-swap” contínuo, a partir da ideia da não necessidade de espera enquanto as baterias do robô são carregadas, sendo apenas necessário trocar a bateria descarregada por uma bateria totalmente carregada. Para ser possível a adaptação do conceito “hot-swap” com um robô móvel, é necessário configurá-los para ter a bateria do tipo de cartucho, com trinco mecânico e estrutura de junção. Adicionalmente a estação deverá ter a estrutura de inserção/remoção de baterias, necessitando de estruturas mecânicas adicionais tanto nas baterias como no robô, sendo possível fazê-lo com uma construção simples, baixo custo e com um tempo pequeno na troca.

2.4 Conclusões

Neste capítulo analisou-se um conjunto de métodos de localização para robôs móveis, com especial foco nos algoritmos AMCL e Perfect Match, sendo estes os mais promissores para a aplicação concreta desta dissertação. Isto é devido essencialmente aos requisitos de utilização das características próprias do ambiente, sem a necessidade de colocação de objetos externos para facilitar o processo de localização dos robôs móveis, assim como algumas limitações de *hardware* existentes no robô utilizado para o presente projeto. O AMCL é um dos algoritmos mais utilizados para a auto-localização de robôs móveis aplicados nas circunstâncias descritas anteriormente, tendo revelado bons resultados práticos ao longo dos anos. Por outro lado, o Perfect Match é um algoritmo muito otimizado e com reduzido peso computacional, também provando ser robusto e eficaz em diversas aplicações.

Numa outra vertente, depois de explorar alguns trabalhos relacionados com a autonomia das baterias nos robôs móveis concluiu-se que as soluções de troca de baterias ou são para robôs ainda aplicados em ambientes laboratoriais e/ou experimentais [8, 44] ou são soluções industriais^{3,4} muito caras para a maioria das aplicações. Assim, existe a necessidade de uma nova solução de aplicação mais ampla.

³www.egeminusa.com/pages/battery_charging/agvs_battery_charging_autoswap.html

⁴www.ocme.it/products-and-technologies/machine-range/internal-logistics/agvlgv—automatedlaser-guided-vehicle/agv-battery-station/automatic-battery-exchange.aspx

Capítulo 3

Robô Clever

O robô Clever, patrocinado pela empresa CLEVERHOUSE, foi utilizado no desenvolvimento do presente projeto de dissertação. Assim, neste capítulo, serão documentadas as características relevantes do robô, tanto da estrutura como do *hardware* e de todo o *software* nele presente.

3.1 Hardware

Este robô tem um formato circular, com um raio aproximado de 0.23 m e tem cerca de 0.275 m de altura, medidos em relação ao chão (Figura 3.1). Desloca-se num modelo de tração diferencial, com 2 rodas paralelas e centradas em relação à geometria do robô.



Figura 3.1: Robô Clever.

3.1.1 Baterias

Para a alimentação de todo o equipamento elétrico presente no robô, utiliza-se uma caixa de baterias externa, representada na Figura 3.2, que fornece uma tensão DC de cerca de 24 V (24 a 28 V), composta por duas baterias KB1272F1¹ ligadas em série, como é possível verificar na Figura 3.3. Na parte superior da caixa estão presentes 3 contactos de cobre, que, ao serem encostados nos contactos metálicos presentes na parte inferior do robô, conduzirão a corrente necessária para

¹http://www.vision-batt.com/site/product_files/CP1272.pdf

a sua alimentação elétrica. No Capítulo 4 será feita uma apresentação e explicação mais pormenorizada das ferramentas desenvolvidas com vista ao sistema de troca automática de baterias.



Figura 3.2: Caixa que possui internamente as baterias para a alimentação elétrica do robô.

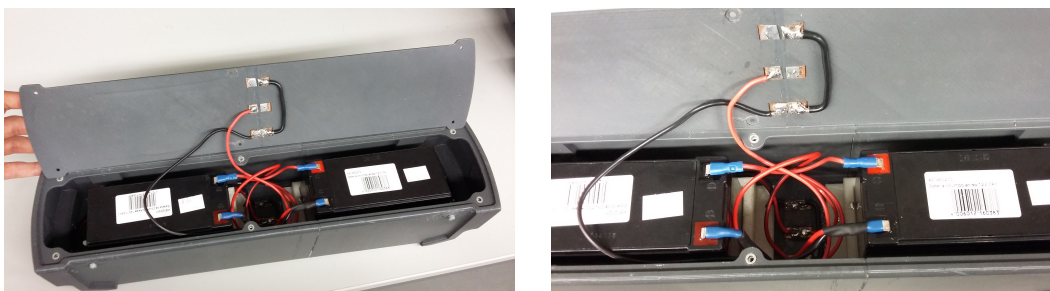


Figura 3.3: Duas baterias KB1272F1 ligadas em série, no interior da caixa.

Tabela 3.1: Especificações das baterias KB1272F1

Tensão nominal	12 V
Número de células	6
Ciclo de vida	5 anos
Capacidade Nominal a 25 °C:	
Taxa de 20h (0.36A, 10.5V)	7.2 Ah
Taxa de 10h (0.69A, 10.5V)	6.9 Ah
Taxa de 5h (1.16A, 10.5V)	5.8 Ah
Taxa de 1h (4.8A, 9.6V)	4.8 Ah
Resistência interna	22 mΩ
Corrente de curto circuito	350 A
Corrente máxima de carregamento	2.88 A

Para além destas baterias externas (Tabela 3.1), o robô possuiu internamente 4 baterias mais pequenas, de 6 V (ligadas em série - Figura 3.4), cuja função é fornecer temporariamente a energia necessária ao robô quando este está na fase da mudança automática de bateria. Mais especificamente, quando o robô deposita a caixa de baterias descarregada, necessitará de se deslocar até ao

local onde será realizado o encaixe da bateria carregada, tendo também de atuar os servomotores.

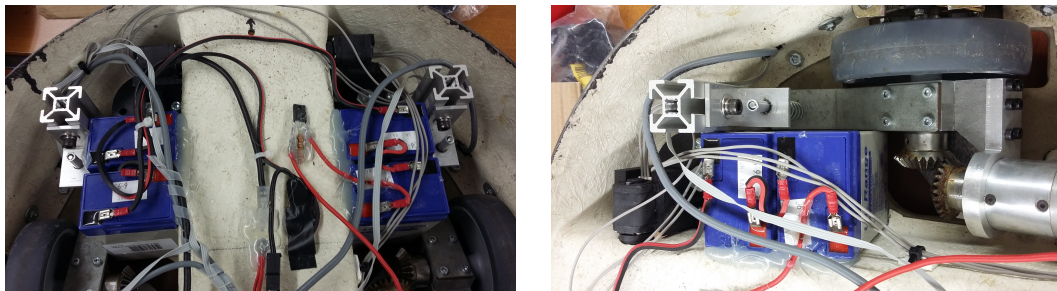


Figura 3.4: Quatro baterias internas do robô, cada uma de 6 V, ligadas em série.

3.1.2 Motores e *encoders*

Os motores utilizados são os EMG49, equipados com *encoders* e uma caixa de velocidades redutora de 49:1²:

Tabela 3.2: Especificações dos motores e *encoders*

Tensão nominal	24 V
Binário nominal	16 kg/cm
Velocidade nominal	122 rpm
Corrente nominal	2100 mA
Velocidade sem carga	143 rpm
Corrente sem carga	500 mA
Corrente de “Stall”	13 A
Potência de saída nominal	34.7 W



Figura 3.5: Motor EMG49.

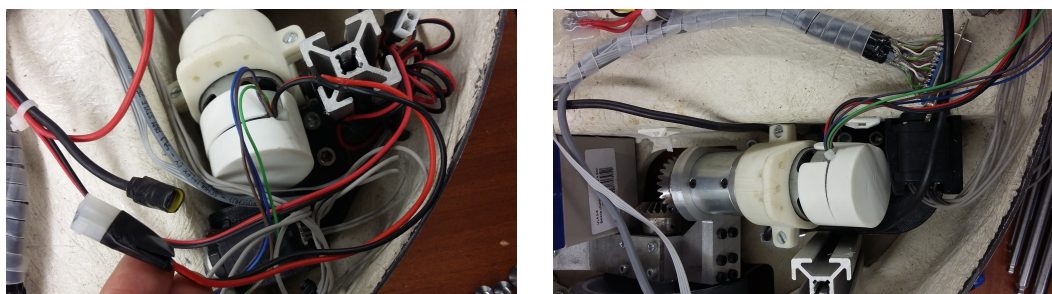
²<http://www.robot-electronics.co.uk/htm/emg49.htm>

Tabela 3.3: Conectores para a alimentação elétrica dos motores

Cabo Preto	Motor-
Cabo Vermelho	Motor+

Tabela 3.4: Conexões dos *encoders*

Roxo	Hall Sensor B Vout
Azul	Hall sensor A Vout
Verde	Hall sensor Ground
Castanho	Hall sensor Vcc

Figura 3.6: Motores EMG49 com os respectivos *encoders*, montados no robô.**Drivers dos motores:**

Para o controlo dos motores, foi utilizado o *driver* MD49, especialmente compatível com os motores EMG49 utilizados³, representado na Figura 3.7 .

Tabela 3.5: Especificações dos *drivers* MD49.

Alimentação	24 V
Comunicação série	TTL (nível de 5 V)
<i>Baudrate</i>	9600 ou 38400 bps

3.1.3 Servomotores

Contrariamente às secções descritas anteriormente, em que foi documentado o equipamento que já existia antes de se iniciar este projeto de dissertação, a inclusão e o desenvolvimento do sistema dos 4 servomotores foi trabalho realizado exclusivamente neste projeto. A utilização dos 4 servomotores AX-12 da Dynamixel⁴ está relacionada com o encaixe automático da caixa das baterias, servindo como atuadores para este processo.

³<http://www.robot-electronics.co.uk/htm/md49tech.htm>

⁴http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm

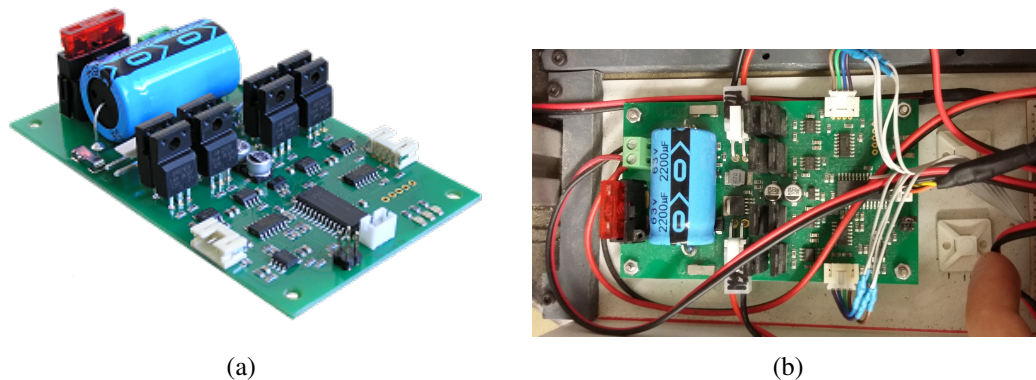


Figura 3.7: (a) Representação dos *drivers* MD49. (b) *Drivers* dos motores montados no robô.

Os servomotores são diretamente controlados pelo computador, comunicando por uma ligação USB, através de um circuito que funciona como interface tanto para a comunicação como para a alimentação elétrica dos motores (Figura 3.8).

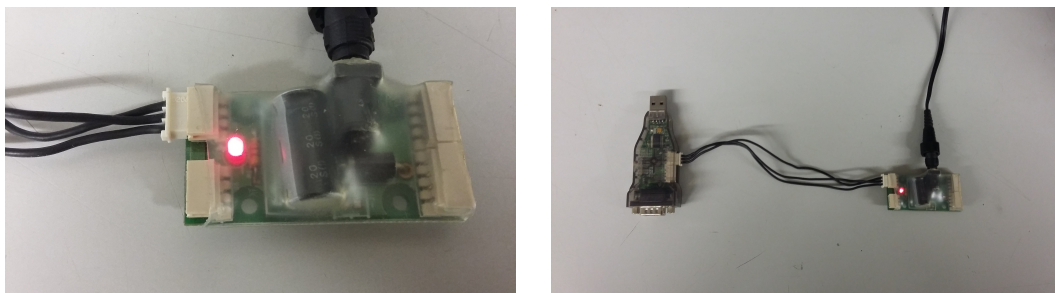


Figura 3.8: Circuito de interface para a comunicação e alimentação elétrica dos servomotores AX-12.

Tabela 3.6: Especificações dos servomotores AX-12.

Peso	53.5 g
Dimensões	32 mm x 50 mm x 40 mm
Resolução	0.29°
<i>Stall</i> Torque	1.5 N.m (a 12.0 V, 1.5 A)
Velocidade sem carga	59 rpm (a 12V)
Rotação	0° a 300°, ou em modo de roda, sem limitação de ângulos.
Temperatura	-5 °C a +70 °C
Tensão	9 a 12 V (recomendado 11.1 V)
Protocolo de comunicação	Comunicação série assíncrona <i>half duplex</i> (8bit, 1stop, No Parity)
Ligação Física	TTL Level Multi Drop (conexão em modo <i>daisy chain</i>)
IDs	254 IDs (0 a 253)
<i>Baudrate</i>	7343bps a 1 Mbps
<i>Feedback</i>	Posição, Temperatura, Carga, Tensão de entrada, etc.

Em cada um destes servos acoplou-se uma peça de alumínio (Figura 3.9), desenhada especificamente para encaixar e elevar a bateria, no momento da sua troca automática. Para o controlo destes atuadores desenvolveu-se um programa em C++ aplicado no ambiente ROS, o qual será exposto no Capítulo 3.2.6.



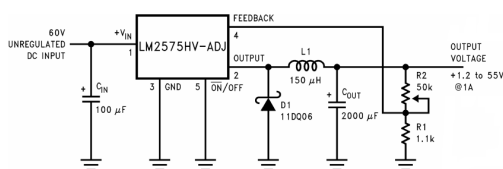
Figura 3.9: Peça de alumínio desenhada para a elevação das baterias.

3.1.4 Conversores *Step-Down*

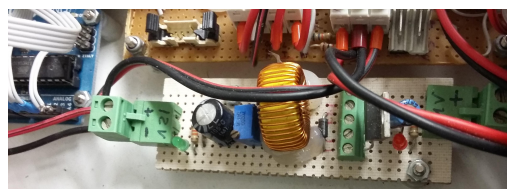
Na transição entre o andar de base e o andar superior do robô, há um barramento de 24 V, alimentado diretamente das baterias do robô. Assim, para se obter um nível de tensão apropriado para os diferentes componentes eletrónicos presentes, utiliza-se 2 conversores abaixadores DC-DC. Um desses conversores (LM2575HV)⁵ alimenta o Arduino e um circuito eletrónico de controlo auxiliar.

Tabela 3.7: Especificações do conversor LM2575HV.

Tensão de saída ajustável	continuamente, através do potenciómetro, de 3.3 a 15 V
Corrente de saída	até 1 A
Tensão de entrada	até 7 a 40 V



(a)



(b)

Figura 3.10: (a) Esquema do circuito conversor abaixador utilizando o LM2575HV e (b) o mesmo circuito integrado no robô.

⁵<http://www.ti.com.cn/cn/lit/ds/symlink/lm2575-n.pdf>

Para a alimentação dos 4 servomotores, utiliza-se outro circuito conversor abaixador DC-DC, com o MAX724⁶. Cada servomotor consome, no máximo 900 mA, totalizando 3.6 A de corrente máxima necessária para alimentar os 4 do sistema.

Tabela 3.8: Especificações do conversor MAX724.

Tensão de saída ajustável	2.5 V a 35 V
Corrente de saída	até 5 A
Tensão de entrada	8 a 40 V

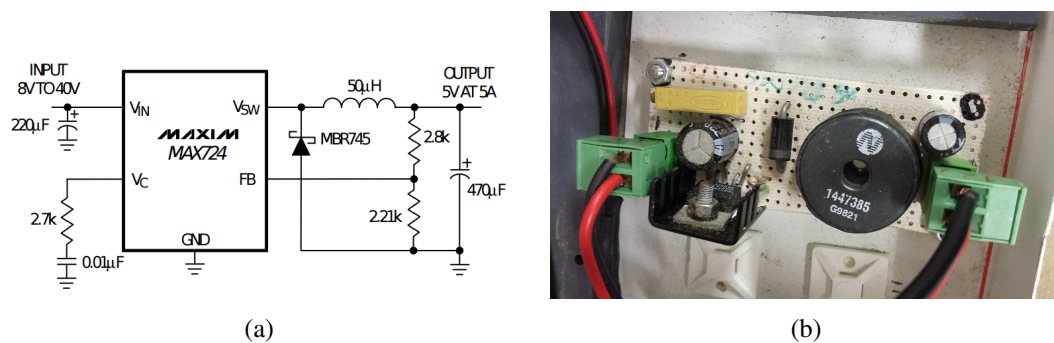


Figura 3.11: (a) Esquema do circuito conversor abaixador utilizando o MAX724 e (b) o mesmo circuito integrado no robô.

3.1.5 Laser

O laser utilizado por este robô é o URG-04LX-UG01⁷, montado na zona frontal do andar superior do robô, invertido, como é possível verificar na Figura 3.12. Este comunica diretamente com o computador através de uma ligação USB, por onde é eletricamente alimentado.



Figura 3.12: Laser URG-04LX-UG01 montado no robô.

⁶<https://datasheets.maximintegrated.com/en/ds/MAX724-MAX726.pdf>

⁷https://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_UG01_spec_en.pdf

Tabela 3.9: Especificações do laser URG-04LX-UG01.

Laser infravermelho	comprimento de onda 785 nm
Área de detecção	240°
Distância de detecção	de 20 mm a 4000 mm
Tensão de alimentação	5 V DC, através da ligação USB
Consumo de corrente	500 mA
Resolução	1 mm
Resolução angular	0.36°
Número de pontos	683
Tempo de varrimento	100 ms/varrimento
Peso	160 g
Dimensões externas	50 x 50 x 70 mm

3.1.6 Arduino

No robô está presente um microprocessador Arduino Uno, o qual é responsável por interagir com um conjunto de sensores e atuadores presentes no robô: sonares, LEDs, solenoides. Este comunica com o computador através de uma ligação USB, operando como interface entre todo o *software* de controlo que é executado no computador, em ROS, e alguns dos sensores e atuadores integrados no robô. Na Figura 3.13 estão representadas as ligações do Arduino.

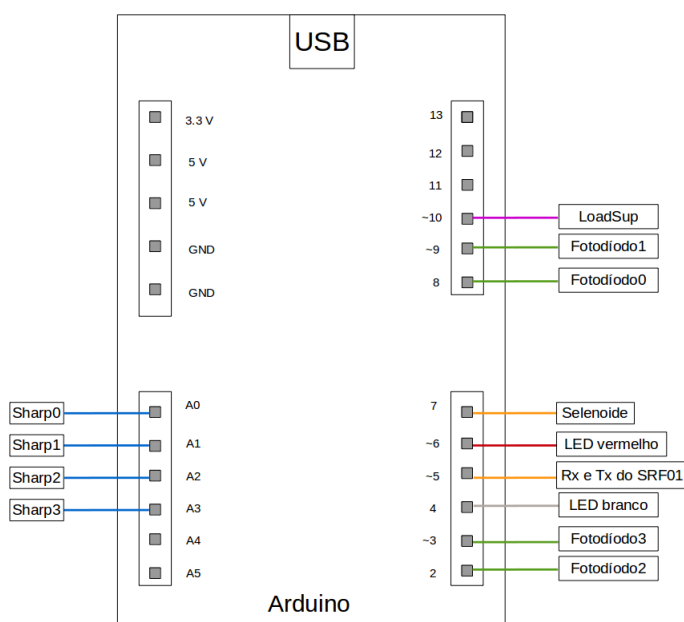


Figura 3.13: Ligações do Arduino Uno presente no robô.

3.1.7 Sonares e LEDS

Na zona frontal do andar superior do robô, estão distribuídos 4 sonares SRF01⁸ para auxílio extra (além do laser frontal) na detecção e desvio de obstáculos. Estes estão ligados a um circuito elétrico que por sua vez está conectado aos pinos de entrada e saída do Arduino.

Tabela 3.10: Especificações dos sonares SRF01.

Distância de detecção	de 0-18 cm a 600 cm
Resolução	3 a 4 cm
Frequência	40 KHz
<i>Baudrate</i>	9600, 19200 ou 38400 bps
Tensão de alimentação	entre 3.3 e 12V (5.5V recomendado)
Corrente de consumo	25 mA em operação, 11 mA em <i>Standby</i> e 55 uA suspenso
Ligação física	TTL
Dimensões	2 cm (diâmetro) x 4 cm

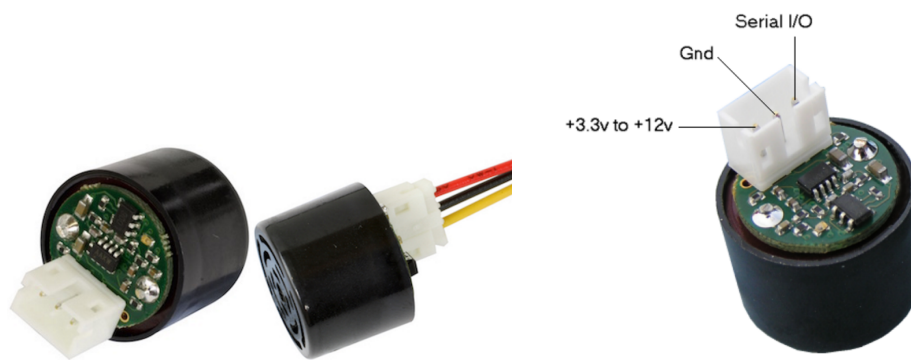


Figura 3.14: Sonares SRF01 com representação dos pinos de ligação.



Figura 3.15: (a) Sonares SRF01 integrados na placa pertencente à zona média do robô. (b) Robô com a placa dos sonares montada.

Para além dos sonares, o robô possuiu dois conjuntos de LEDs, um na zona frontal e outro na parte traseira (Figura 3.16), cobrindo o perímetro externo do robô. Estes também são controlados

⁸<https://www.robot-electronics.co.uk/html/srf01tech.htm>

pelo Arduino, que por sua vez está diretamente ligado ao computador, onde está integrado o *software* principal, em ROS. Este conjunto de LEDs é principalmente para aplicações de publicidade, colocados com o intuito de despertar a atenção das pessoas que partilhem o mesmo espaço do robô.

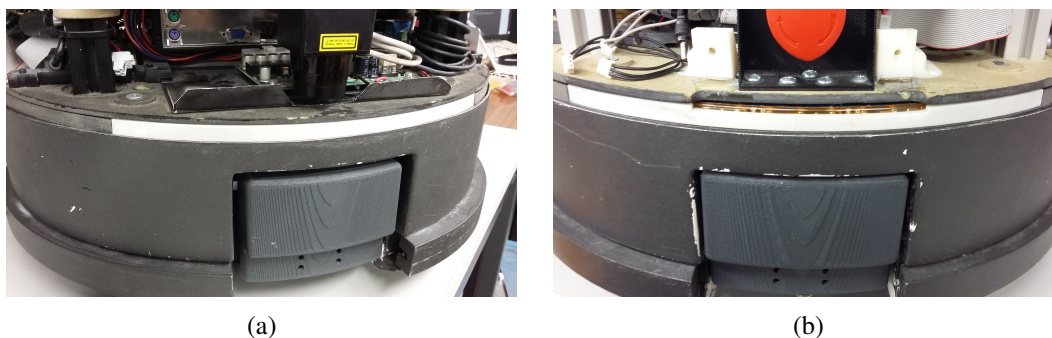


Figura 3.16: (a) LEDs frontais (placa branca) e (b) LEDs traseiros do robô (placa branca).

3.1.8 Computador

O robô está equipado com um computador que utiliza o sistema operativo Linux, com a distribuição Ubuntu 14.04⁹. Esta é a unidade central de todos os processos comportamentais do robô, onde está a ser executada toda a hierarquia ROS que será descrita em pormenor na Secção 3.2. O computador possuiu 1 Gb de memória RAM e utiliza o processador Atom D400¹⁰, da Intel®. Aqui reside uma das questões mais importantes referentes às limitações do software do robô.

Tabela 3.11: Especificações do processador Atom D400.

Núcleos de processamento	1
<i>Threads</i>	2
Arquitetura	64 bits
<i>Cache</i>	512 KB L2
Frequência de base	1.66 GHz
Intervalo de tensão de operação	0.800 V a 1.175 V
Potência de consumo	10W

Como é possível verificar por algumas destas especificações técnicas mais relevantes, este é um processador com pouco poder de cálculo (baixa frequência de base e apenas 1 núcleo de processamento), mas em contrapartida consome pouca potência (máximo de 10 W).

É importante referir que este é um consumo constante durante todo o período de atividade do robô, mesmo quando este está parado, em oposição, por exemplo, aos motores que, apesar de consumirem mais energia é apenas durante a movimentação do robô. Este é um fator muito relevante no contexto de robôs de reduzidas dimensões, pois estes não têm capacidade para integrar baterias de elevado peso e grandes dimensões.

⁹<http://www.ubuntu.com/>

¹⁰<http://www.intel.com/content/www/us/en/processors/atom/atom-d400-d500-vol-1-1-datasheet.html>

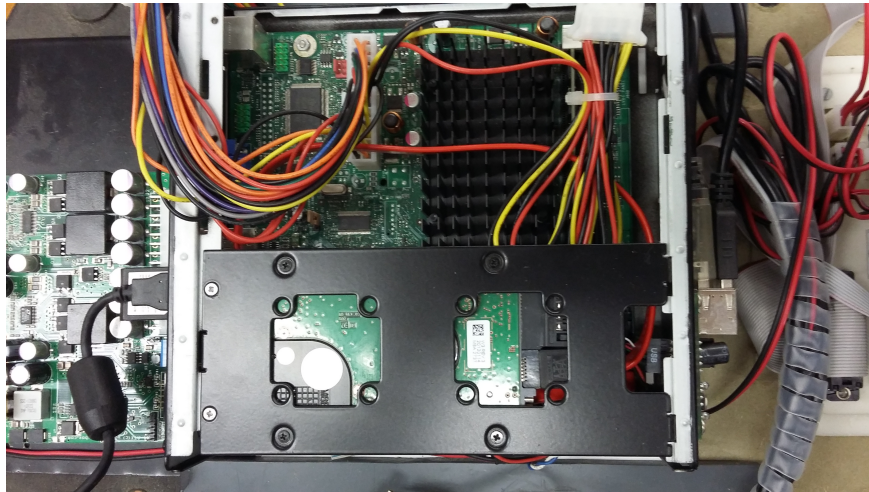


Figura 3.17: Computador interno montado no robô.

3.1.9 Sistema para logística

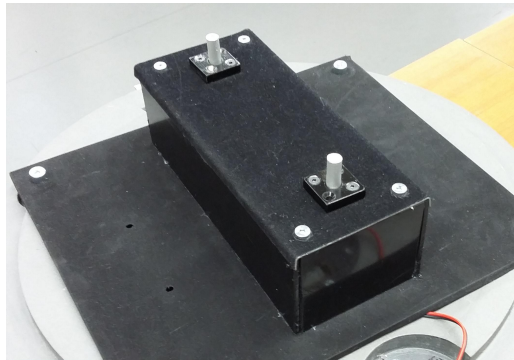
Um das aplicações pensadas para a plataforma robótica genérica está relacionada com operações de logística, mais concretamente no transporte de prateleiras móveis. Assim, ao longo deste projeto desenvolveram-se os módulos necessários para este sistema, nomeadamente uma estrutura com um mecanismo atuado por solenóides e um encaixe colocado por baixo das prateleiras móveis, para permitir o acoplamento entre estas e o robô.



Figura 3.18: Estrutura integrada na superfície superior do robô para o sistema de transporte de prateleiras móveis.

Na Figura 3.18 é possível visualizar a placa preta com o mecanismo dos solenóides aparafusada na parte superior do robô, cuja função é elevar dois cilindros metálicos por forma a encaixar com a superfície inferior das prateleiras. Os dois cilindros metálicos encontram-se no relevo central da estrutura, alinhados com a frente do robô, representados com maior detalhe na Figura 3.19a. Os solenóides estão ligados a um barramento de 12 V (junto à estrutura, na parte superior do robô), o qual fornece a potência necessária para a atuação dos mesmos. Na superfície inferior das prateleiras integrou-se um encaixe para permitir o acoplamento entre os cilindros e a prateleira móvel, visível na Figura 3.19b.

A ideia geral é o robô deslocar-se para baixo das prateleiras, encaixar os cilindros metálicos na parte inferior destas e arrastá-las até ao local de destino, baixando de novo os cilindros e soltando a prateleira. Na Figura 3.20 está visualmente representado o sistema descrito até aqui.



(a)



(b)

Figura 3.19: (a) Sistema de encaixe com cilindros metálicos, atuado por solenóides. (b) Estrutura de encaixe na parte inferior da prateleira.

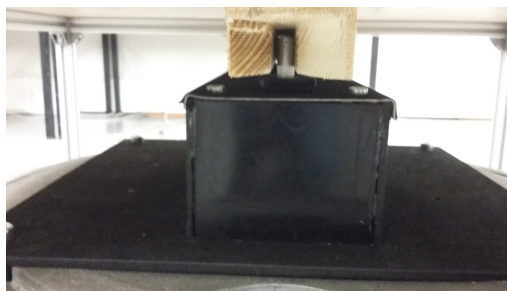
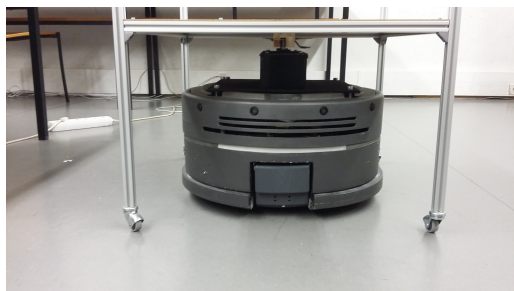


Figura 3.20: Robô acoplado com a prateleira móvel.

3.2 Software ROS

Inicialmente, nesta secção, irá fazer-se uma introdução à plataforma de desenvolvimento ROS. Depois será feita uma descrição, com detalhe, de todo o *software* que é executado no computador do robô, desde a arquitetura geral até aos programas individuais e as suas funções, desenvolvidos em ROS.



Figura 3.21: Logo do ROS.

Como já foi referido anteriormente, utilizou-se, para o desenvolvimento deste sistema, o *framework* de desenvolvimento ROS (com a versão *indigo*¹¹), muito popular nos últimos anos para a integração e implementação de *software* a ser utilizado em aplicações de robótica.

O Robot Operating System (ROS)¹² é um *framework* de desenvolvimento de *software* que integra uma vasta coleção de bibliotecas e ferramentas que facilitam a criação e implementação de sistemas robóticos: abstração de *hardware*, abstração de comunicação entre programas, *drivers* para diversos tipos de dispositivos, programas para visualização de dados, gestão de pacotes, etc. A reutilização de código é outro aspeto importante no contexto da criação de *software* de elevada complexidade (como é no caso da robótica) e que é facilitada utilizando as bibliotecas criadas por comunidades de todo o mundo, aumentando a velocidade no desenvolvimento das aplicações. Isto porque a execução de tarefas aparentemente simples na perspetiva humana envolve a integração de um grande número de programas interligados com diferentes tipos de hardware, o que torna a sua implementação um grande desafio. Em adição, a utilização de ROS facilita o trabalho em equipa, pois qualquer pessoa ou entidade pode desenvolver um nó que facilmente se integra com outros nós ou sistemas, pois o *framework* realiza todos os processos de comunicação entre eles.

Na base do ROS está um conjunto de conceitos fundamentais de todo o *framework*. Numa visão geral, os sistemas desenvolvidos em ROS consistem em vários programas que comunicam entre si por forma a atingir determinados objetivos comportamentais.

3.2.1 Nós

Um nó¹³ é um processo que executa um determinado código - programa. Os nós são os blocos fundamentais de qualquer sistema ROS que fazem parte de um gráfico de comunicações com arquitetura ponto-a-ponto, comunicando entre si através de tópicos, serviços, ações ou a partir

¹¹<http://wiki.ros.org/indigo>

¹²<http://wiki.ros.org/>

¹³<http://wiki.ros.org/Nodes>

do servidor de parâmetros. Como o ROS funciona como um sistema distribuído, os nós não têm necessariamente de operar no mesmo computador, podendo ser executados a partir de várias plataformas que comunicam entre si. Entre os nós, destaca-se o mestre (*roscore*) responsável por fornecer as informações necessárias para as comunicações entre os restantes nós (utilizando XML-RPC (Extensible Markup Language Remote Procedure Call)), sem que seja necessário que estes conheçam o endereço IP (Internet Protocol) e a porta onde os outros nós estão a ser executados.

Quando um novo nó aparece no sistema, o *roscore* providencia os dados necessários para que este possa formar uma ligação ponto-a-ponto com os restantes nós que publicam e subscrevem os mesmos tópicos [45]. Para além disto, o *roscore* também gere o servidor de parâmetros, atualizando as informações transmitidas pelos nós no início da sua execução, ou a partir de reconfigurações em *runtime*, utilizando-se a API *dynamic_reconfigure*¹⁴.

3.2.2 Tópicos

Os tópicos são barramentos de comunicação (identificados por nomes únicos) pelos quais os nós trocam mensagens entre si, seguindo um modelo de *publisher/subscriber* que separa a produção e o consumo de dados. Os nós simplesmente subscrevem os tópicos de que necessitam e, quando geram informações, publicam-nas para os tópicos relevantes¹⁵. Com esta arquitetura é possível que múltiplos nós subscrevam e publiquem para os mesmos tópicos, podendo processar a mesma informação para aplicações diferentes. Atualmente o ROS suporta transporte de mensagens através de TCP/IP (Transmission Control Protocol/Internet Protocol), conhecido por TCPROS¹⁶ e UDP (User Datagram Protocol), nomeado de UDPROS¹⁷. Por *default* é utilizado o transporte por TCP/IP, que é mais fidedigno, garantindo a entrega de todas as mensagens, em oposição ao UDP, que apesar de produzir menos *overhead* e diminuir a latência, não garante a entrega de todos os pacotes de dados.

3.2.3 Serviços

Os serviços são uma forma de comunicação do tipo de pedido/resposta (*request/reply*), normalmente utilizados quando um nó necessita de consultar alguma informação referente a outro nó ou quando é necessário iniciar a execução de uma determinada atividade. Um nó (servidor) dispõe um determinado serviço, identificado por um nome único, que pode ser requisitado por outro nó (cliente) que chama esse serviço e espera pela resposta¹⁸.

3.2.4 Ações

As ações são outra forma de comunicação entre os nós, muito úteis nos casos em que a ação requisitada por um determinado nó não é algo imediato e que se prolonga pelo tempo. As ações

¹⁴http://wiki.ros.org/dynamic_reconfigure

¹⁵<http://wiki.ros.org/Topics>

¹⁶<http://wiki.ros.org/ROS/TCPROS>

¹⁷<http://wiki.ros.org/ROS/UDPROS>

¹⁸<http://wiki.ros.org/Services>

são definidas por um objetivo final, um resultado e por um conjunto de *feedbacks* dados ao longo da sua execução. O nó que requisita a ação indica o objetivo final pretendido e o nó que a executa vai informando do desenrolar incremental do processo (através de *feedbacks*), até à devolução do resultado final da ação¹⁹.

3.2.5 Arquitetura geral

Por forma a que um robô desempenhe um conjunto de funções, é necessário haver vários grupos de ligações relacionando diversos programas. No caso concreto do robô Clever, foi necessário integrar vários nós, cada um com a sua função, com vista ao bom funcionamento do sistema final. Para que a execução do *software* seja realizada de uma forma modular e simplificada, utilizam-se vários níveis de lançamento dos nós de ROS (ficheiros *launch*), destacando-se três principais: nível de *hardware*, nível de navegação e nível de decisão.

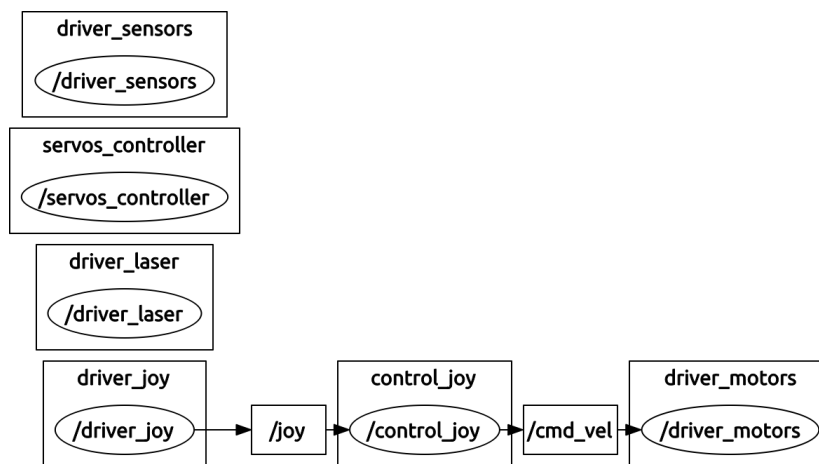


Figura 3.22: Arquitetura de ROS, visualizada através do *rqt_graph*, do nível de lançamento de *hardware*.

Na figure 3.22 é possível visualizar graficamente toda a hierarquia ROS relacionada com o *hardware* que é executada no robô Clever, com este nível de lançamento. Esta representação foi obtida a partir de uma das ferramentas de visualização do ROS: o *rqt*²⁰, utilizando especificamente o *plugin rqt_graph*²¹ que demonstra graficamente toda a hierarquia de nós, interligados pelos tópicos que os relacionam, formando o sistema geral de execução.

A partir da análise da imagem verifica-se que a este nível apenas são lançados os nós de ROS referentes ao controlo de *hardware*, que incluem os *drivers* dos motores, dos sensores, do laser, do controlo por *joystick* e o nó de controlo dos servomotores. Isto permite a verificação e validação das funcionalidades de todas as componentes de controlo do *hardware*, permitindo uma mais fácil resolução de problemas. Para além disso, é muito útil utilizar o controlo por *joystick* tanto para

¹⁹<http://wiki.ros.org/actionlib>

²⁰<http://wiki.ros.org/rqt>

²¹http://wiki.ros.org/rqt_graph

a movimentação do robô de umas áreas para outras, assim como para a utilização de algumas ferramentas de mapeamento (*e.g.*, SLAM) aquando da criação de mapas do ambiente.

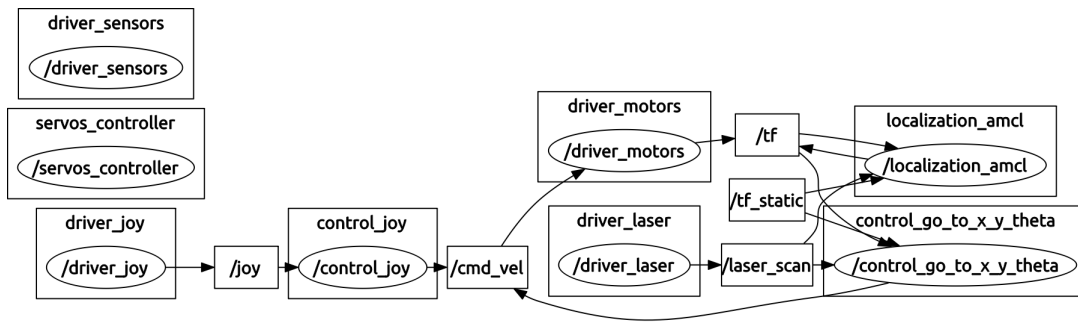


Figura 3.23: Arquitetura de ROS, visualizada através do *rqt_graph*, do nível de lançamento de navegação.

Neste caso, representado na figura 3.23 foram lançados todos os nós que operam desde o *hardware* (figura 3.22) até ao nível da navegação (que incluiu os nós de localização e um nó de controlo de posição para a navegação até aos pontos pretendidos).

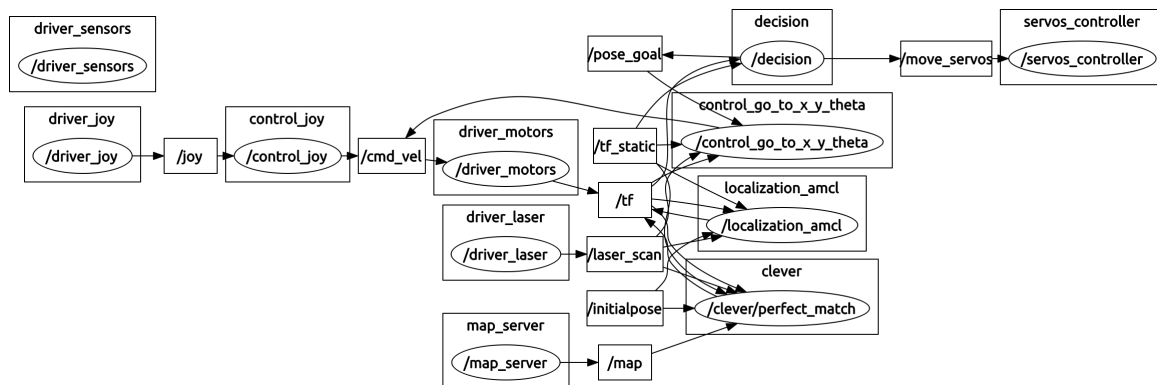


Figura 3.24: Arquitetura de ROS, visualizada através do *rqt_graph*, do nível de lançamento de decisão.

Em adição aos dois anteriores níveis de lançamentos de programas de ROS, na figura 3.24 está incluído um nó de decisão, cuja função é proporcionar uma navegação completamente autónoma ao robô, sem ser necessário a utilização de um controlo remoto (nível de *hardware*), nem o fornecimento manual de pontos de destino pelo utilizador (no caso do nível de lançamento da navegação). Na secção seguinte, irá fazer-se uma descrição mais pormenorizada dos programas visualizados nas imagens anteriores, fornecendo todas as informações necessárias para a sua utilização e compreensão.

3.2.6 Descrição dos nós de ROS

A análise será efetuada desde o nível mais baixo, do *hardware* até ao nível mais alto de tomada de decisões.

1. *Driver* do laser

O *driver* para o laser URG-04LX-UG01 já existia nas bibliotecas oficiais do ROS (*hokuyo_node* ²²) e por isso foi utilizado diretamente, tendo apenas sido necessário configurar os parâmetros referentes às características do laser. Na figura 3.25 está o *dynamic_reconfigure* do *driver* do laser, onde estão listados todos os parâmetros configuráveis deste nó, podendo ser alterados durante a sua execução.

Por forma a ser possível utilizar corretamente este programa, tal como em todos os outros nós, é necessário conhecer os tópicos necessários para o seu funcionamento²³.

Nó [/driver_laser]

Publicações:

- /driver_laser/parameter_updates [dynamic_reconfigure/Config]
- /driver_laser/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
- /laser_scan [sensor_msgs/LaserScan]

Subscrições: Nenhuma

Serviços:

- /driver_laser/set_parameters

²²http://wiki.ros.org/hokuyo_node

²³Irão omitir-se os tópicos de logging (/rosout, */set_logger_level, */get_loggers, */self_test, etc) e de diagnóstico de *hardware* (/diagnostics), pois estes são comuns a todos os nós e não são relevantes para efeitos de documentação.



Figura 3.25: Parâmetros ajustáveis em *runtime*, com o *dynamic_reconfigure* do *driver* do laser.

Como já foi referido, este nó tem implementado a possibilidade de reconfiguração dinâmica, em *runtime*, através do *dynamic_reconfigure*, sendo as duas primeiras publicações: `/driver_laser/parameter_updates`, `/driver_laser/parameter_descriptions` e o serviço `/driver_laser/set_parameters` para este propósito. No final de cada varrimento do laser, este nó publica o conjunto do valor das distâncias aos obstáculos detetados para o tópico `/laser_scan`, que é do tipo `sensor_msgs/LaserScan`.

Parâmetros:

- `"port"`: Porta USB onde está ligado o laser.
- `"frame_id"`: ID do referencial do laser do robô.
- `"min_ang"`: Ângulo mínimo do varrimento do laser, em radianos.
- `"max_ang"`: Ângulo máximo do varrimento do laser, em radianos.

2. Driver do joystick

Nó `[/driver_joy]`

Publicações:

- `/joy` [`sensor_msgs/Joy`]

Subscrições: Nenhuma

Serviços: Nenhum

O programa do *driver* do *joystick* adquire, no recetor conectado a uma das portas USB do computador, os dados enviados pelo comando remoto. De seguida, converte os dados recebidos para o formato de mensagens `sensor_msgs/Joy`, publicando no tópico `/joy`. As

mensagens publicadas apenas indicam os botões premidos no *joystick* por parte do utilizador, que depois serão processadas pelo nó de controlo `/control_joy`. Este programa também pertence às bibliotecas oficiais de ROS²⁴, tendo-se apenas configurado os parâmetros de ajuste do nó.

3. Controlo do *joystick*

Nó `/control_joy`

Publicações:

- `/cmd_vel` [geometry_msgs/Twist]

Subscrições:

- `/joy` [sensor_msgs/Joy]

Serviços: Nenhum

Parâmetros:

- `"axis_linear"`: Definição do botão que dará os comandos de velocidade linear.
- `"axis_angular"`: Definição do botão que dará os comandos de velocidade angular.
- `"axis_deadman"`: Definição do botão com função de *deadman switch*.
- `"scale_linear"`: Fator de multiplicação para a velocidade linear máxima do robô.
- `"scale_angular"`: Fator de multiplicação para a velocidade angular máxima do robô.

Este programa, pertencente às bibliotecas de ROS²⁵ tem como principal função interpretar o conteúdo das mensagens publicadas no tópico `/joy`, pelo nó `/driver_joy`, e convertê-las para comandos de velocidade que serão posteriormente enviadas para os motores, pelo nó do respetivo *driver*.

4. Controlo dos servomotores

Nó `/servos_controller`

Publicações: Nenhuma

Subscrições:

- `/move_servos` [std_msgs/String]

Serviços:

- `/get_servos_status`

²⁴<http://wiki.ros.org/joy>

²⁵http://wiki.ros.org/turtlebot_teleop

O programa de controlo dos servomotores do robô funciona a partir de um comando enviado através do tópico `/move_servos` que indica a ação pretendida por parte do controlador. Ao enviar a *string* "Open", os 4 servomotores rodam por forma a que a caixa das baterias seja solta, deixando-a cair. Por outro lado, no momento do encaixe da bateria, é necessário publicar a *string* "Close" para que os servos elevem e segurem a bateria, com o objetivo de colocar os contactos elétricos na posição correta para fornecer a energia necessária ao robô. Para além disto, é possível saber a posição momentânea de cada servomotor através de uma chamada ao serviço `/get_servos_status` deste nó.

Este programa foi totalmente desenvolvido neste projeto de dissertação, tendo-se depois integrado no restante sistema.

Parâmetros:

- "num_servos": Número total de servomotores a controlar.
- "servos_velocity": Velocidade nominal de rotação dos servomotores, em graus/s.
- "tolerance_to_pos": Máximo erro permitido para a posição final indicada.
- "initial_pos": Posição inicial dos servomotores, de 0 (0°) a 1023 (300°).
- "final_pos": Posição final dos servomotores, de 0 (0°) a 1023 (300°).

5. Drivers dos sensores

Nó [/driver_sensors]

Publicações:

- /driver_sensors/sonar0 [sensor_msgs/Range]
- /driver_sensors/sonar1 [sensor_msgs/Range]
- /driver_sensors/sonar2 [sensor_msgs/Range]
- /driver_sensors/sonar3 [sensor_msgs/Range]
- /driver_sensors/sharp0 [sensor_msgs/Range]
- /driver_sensors/sharp1 [sensor_msgs/Range]
- /driver_sensors/sharp2 [sensor_msgs/Range]
- /driver_sensors/sharp3 [sensor_msgs/Range]
- /driver_sensors/opt_rr [sensor_msgs/Range]
- /driver_sensors/opt_fl [sensor_msgs/Range]
- /driver_sensors/opt_fr [sensor_msgs/Range]
- /driver_sensors/opt_rl [sensor_msgs/Range]

Subscrições:

- /driver_sensors/command [unknown type]

Serviços: Nenhum

Este é o nó de ROS que comunica, via porta USB, com o Arduino, que, por sua vez, recebe os dados dos sensores do robô (sonares, LEDs). Como é possível verificar, publica para 12 tópicos, sendo 4 referentes aos sonares (*/driver_sensors/sonar0*, */driver_sensors/sonar1*, */driver_sensors/sonar2* e */driver_sensors/sonar3*), outros 4 referentes aos LEDs do robô (*/driver_sensors/opt_rr*, */driver_sensors/opt_fl*, */driver_sensors/opt_fr*, */driver_sensors/opt_rl*) e outros 4 tópicos referentes aos sharps (*/driver_sensors/sharp0*, */driver_sensors/sharp1*, */driver_sensors/sharp2*, */driver_sensors/sharp3*).

Este programa não faz parte das bibliotecas de ROS, tendo sido desenvolvido pelo grupo de robótica do INESC TEC/FEUP e utilizado para o presente projeto de dissertação, apesar de neste momento ainda não estar integrado no resto do sistema. Para que o programa seja integrado é apenas necessário criar um nó ou utilizar um já existente que publique para o tópico */driver_sensors/command*, indicando o tipo de ação que se pretende deste.

6. *Drivers* dos motores

Nó [/driver_motors] <u>Publicações:</u> • /tf [tf2_msgs/TFMessage] <u>Subscrições:</u> • /cmd_vel [geometry_msgs/Twist] <u>Serviços:</u> Nenhum
--

Parâmetros:

- "port": Porta USB onde está ligado o *driver* dos motores.
- "wheel_radius": Raio das rodas do robô, em metros.
- "dist_between_wheels": Distância entre as rodas do robô, em metros.
- "enc_pulses_per_revolution": Número de pulsos do *encoder* por cada rotação completa das rodas.
- "odom_frame_id": ID do referencial da odometria.
- "base_frame_id": ID do referencial da base do robô.
- "laser_frame_id": ID do referencial do laser do robô.
- "laser_x_offset": Distância (em x) do laser em relação à origem do referencial de base do robô ("base_frame_id").
- "laser_z_offset": Distância (em z) do laser em relação à origem do referencial de base do robô ("base_frame_id").

Este programa também foi desenvolvido pelo grupo de robótica do INESC TEC/FEUP, antes do início deste projeto. Essencialmente tem como função fazer a comunicação com os motores, subscvendo o tópico */cmd_vel*, no qual são recebidos os comandos de velocidade, e enviá-los para os *drivers*, através do protocolo de comunicações especificado. Para além disto, este nó é responsável pela publicação das transformadas entre o referencial do laser (*laser_frame_id*) e o referencial da base do robô (*base_frame_id*), a partir das informações fornecidas através dos parâmetros *laser_x_offset* e *laser_z_offset*, referentes ao posicionamento relativo entre o laser e a base do robô, necessárias para o restante sistema ROS. Como este nó também é responsável pela comunicação com os *encoders* dos motores, publica a transformada entre o referencial de base do robô (*base_frame_id*) e o referencial da odometria (*odom_frame_id*).

7. Localização AMCL

Nó [/localization_amcl]

Publicações:

- /amcl_pose [geometry_msgs/PoseWithCovarianceStamped]
- /tf [tf2_msgs/TFMessage]
- /particlecloud [geometry_msgs/PoseArray]

Subscrições:

- /tf [tf2_msgs/TFMessage]
- /tf_static [tf2_msgs/TFMessage]
- /initialpose [geometry_msgs/PoseWithCovarianceStamped]
- /map [nav_msgs/OccupancyGrid]
- /scan [sensor_msgs/LaserScan]

Serviços:

- /global_localization
- /request_nomotion_update
- /set_map

O nó de localização baseado na localização Monte Carlo (conhecido por filtro de partículas) faz parte da biblioteca de navegação *standard* do ROS. Este subscreve os tópicos referentes à pose inicial do robô (no momento de início de execução do programa - */initialpose*), as transformações de referencial (*/tf* e */tf_static*) e os dados de varrimento do laser (*/scan*). O mapa do ambiente (*/map*) é recebido através de um pedido (*request*) ao serviço */static_map*, disponibilizado pelo nó */map_server*. Com estas informações e aplicando o algoritmo original da localização Monte Carlo juntamente com algumas extensões ao modelo, devolve a pose estimada do robô (no tópico */amcl_pose*).

Este nó também pode publicar a transformada entre o referencial do mapa (*frame: /map*) e o referencial da odometria (*frame: /odom*), sendo outra forma de devolver a pose estimada pelo algoritmo de localização, para posterior controlo do robô. Em adição, também publica para o tópico */particlecloud*, onde estão todas as hipóteses atuais das possíveis poses do robô (partículas) que o algoritmo mantém na sua execução.

Quanto aos serviços disponibilizados, é possível iniciar uma localização global (*/global_localization*) distribuindo uniformemente todas as partículas pelo mapa, com vista a convergir para a pose correta do robô. Por outro lado, o algoritmo AMCL é executado de acordo com a movimentação do robô e não de forma periódica. Por isso, quando o robô está parado o

nó não calcula uma nova posição, comportamento que pode ser alterado quando se requisita o serviço */request_nomotion_update*.

O nó *amcl* tem um vasto número de parâmetros de configuração (que também podem ser alterados em *runtime* com o *dynamic_reconfigure*) que serão pormenorizadamente descritos na Secção 5.2.

8. Localização Perfect Match

Nó */localization_perfect_match*

Publicações:

- */LocalisationPoseResult* [geometry_msgs/PoseStamped]
- */LocalisationPoseWithCovResult* [geometry_msgs/PoseWithCovarianceStamped]
- */tf* [tf2_msgs/TFMessage]
- */debug/TimeCycle* [std_msgs/Int16]
- */debug/TimeLoad* [std_msgs/Int16]
- */debug/LaserScanfiltered* [sensor_msgs/PointCloud]
- */debug/LaserScan* [sensor_msgs/PointCloud]
- */debug/RPropIterationNumber* [std_msgs/Int16]
- */debug/LaserBeam* [sensor_msgs/PointCloud]
- */parameter_updates* [dynamic_reconfigure/Config]
- */parameter_descriptions* [dynamic_reconfigure/ConfigDescription]

Subscrições:

- */tf* [tf2_msgs/TFMessage]
- */tf_static* [tf2_msgs/TFMessage]
- */initialpose* [geometry_msgs/PoseWithCovarianceStamped]
- */scan* [sensor_msgs/LaserScan]
- */map* [nav_msgs/OccupancyGrid]

Serviços:

- */perfect_match/set_parameters*

O nó */localization_perfect_match* é outra implementação de um algoritmo de localização, o Perfect Match, baseado em ajuste de mapa, em oposição aos modelos probabilísticos do filtro de partículas. Este programa possuiu a mesma função do nó *amcl* e a estrutura é idêntica: subscreve os tópicos do mapa (*/map*), transformações de referenciais (*/tf* e */tf_static*),

posição inicial (*/initialpose*) e os dados publicados pelo nó do laser (*/scan*). De igual modo, com base nestes parâmetros de entrada, estima a pose atual do robô, publicando-a para os tópicos */LocalisationPoseResult* e */LocalisationPoseWithCovResult* (pose sem e com covariância associada, respetivamente).

Assim como acontece com o */localization_amcl* também é dada a possibilidade deste nó devolver a pose calculada publicando a transformada entre o referencial do mapa (*frame: /map*) e o referencial da odometria (*frame: /odom*), sendo outra forma de disponibilizar a pose estimada pelo algoritmo. Em adição, e para além dos tópicos relacionados com o *dynamic_reconfigure*, devolve vários dados para auxílio no *debugging* de eventuais maus comportamentos por parte do programa (*/debug/**).

Os parâmetros de configuração do nó */localization_perfect_match* serão descritos de forma mais aprofundada na Secção 5.1.

Ambos os algoritmos de localização têm na sua base a utilização de um mapa para estimar a pose do robô. Assim, é sempre necessário lançar um nó que funciona como servidor (*map_server*²⁶) e que lhes fornece o mapa.

Esta implementação do Perfect Match foi desenvolvida pelo grupo de robótica do INESC-TEC/FEUP, tendo-se estudado com detalhe o seu funcionamento e configurado os parâmetros necessários para a correta utilização nas aplicações concretas. Para além disto, explorou-se algumas possibilidades de melhoramento do algoritmo, expostas na Secção 5.1.2.

9. Controlo de posição e navegação

Nó */control_go_to_x_y_theta*

Publicações:

- */cmd_vel* [geometry_msgs/Twist]

Subscrições:

- */tf* [tf2_msgs/TFMessage]
- */tf_static* [tf2_msgs/TFMessage]
- *pose_goal* [geometry_msgs/PoseStamped]
- */base_scan* [sensor_msgs/LaserScan]

Serviços: Nenhum

²⁶http://wiki.ros.org/map_server

Este programa de controlo de posição para navegação, também desenvolvido pelo grupo de robótica do INESC TEC/FEUP, integra um conjunto de algoritmos para o desvio de obstáculos durante a movimentação do robô, enquanto este se desloca para a posição final pretendida. Nomeadamente, possuiu a opção de utilizar uma rotina de simples paragem da movimentação, no caso de ser detetado um obstáculo no interior da zona de segurança do robô (definida por um retângulo com dimensões configuráveis através dos parâmetros do nó); um algoritmo de desvio de obstáculos com base em campos de potencial; e um algoritmo de desvio de obstáculos com base em histograma polar (*Vector Field Histogram*).

A ideia fundamental dos campos de potencial passa por considerar que o robô é um ponto afetado por um campo de potencial, em que o objetivo (o ponto de destino) exerce forças de atração enquanto que todos os obstáculos do mapa exercem forças de repulsão ao robô, resultando num trajeto até ao ponto final. Apesar deste método possuir vantagens, principalmente relacionadas com o peso computacional e o número de cálculos necessários até chegar à solução final, está sujeito ao efeito de mínimos locais ao longo do trajeto, "prendendo" o robô e impedindo que este encontre um caminho para o objetivo [46].

O algoritmo de desvio de obstáculos com base em histograma polar utiliza o varrimento do laser para calcular a direção apropriada para onde o robô se deve deslocar (com espaço suficiente para que este passe), no caso do aparecimento de um obstáculo na zona de segurança do robô. Cada ponto do laser é considerado como uma célula na grelha do histograma, criando uma distribuição de probabilidade com pouco *overhead* computacional [47]. Esta direção resulta da opção com valor de custo mais baixo dada por $a \times \text{desvio_orientacao_destino} + b \times \text{desvio_orientacao_atual}$. Atualmente, está a utilizar-se o método de desvio de obstáculos com base em histograma polar neste nó de controlo e navegação.

Na Figura 3.26 está representado o funcionamento deste nó, através de uma máquina de estados, com o objetivo de clarificar o comportamento do programa, utilizando a mesma nomenclatura dos parâmetros configuráveis do mesmo.

Parâmetros:

- "FWD_DIST": Distância de segurança dianteira, em metros.
- "LAT_DIST": Distância de segurança lateral, em metros.
- "STEP_HIST": Passo do histograma no algoritmo de desvio de obstáculos.
- "ANG_HIST_INI": Ângulo inicial do histograma, em radianos.
- "MAX ETF": Máximo erro angular permitido para a rotação inicial do robô na direção da posição de destino, em radianos.
- "HIST ETF": Máximo erro angular permitido para a rotação inicial do robô na direção da posição de destino utilizando o desvio por histograma, em radianos.
- "DIST_NEWPOSE": Distância superior à qual o robô volta a rodar sobre ele mesmo para se alinhar com a posição final.

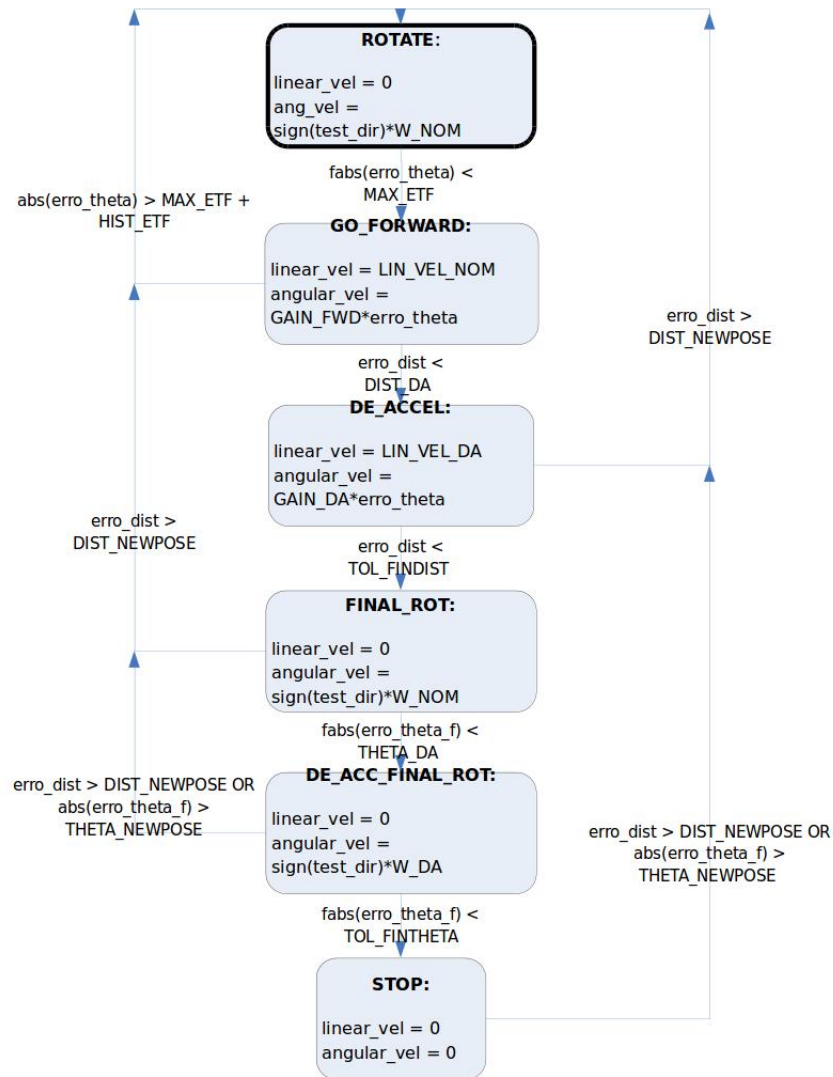


Figura 3.26: Máquina de estados do funcionamento do nó /control_go_to_x_y_theta para controlo de posição e navegação do robô.

- **"DIST_DA"**: Distância da posição de destino a partir da qual o robô começa a desacelerar.
- **"TOL_FINDIST"**: Máximo erro de posição permitido para se considerar que o robô chegou à posição de destino, em metros.
- **"THETA_NEWPOSE"**: Diferença angular superior à qual o robô volta a rodar sobre ele mesmo para se alinhar com a posição final.
- **"THETA_DA"**: Diferença entre o ângulo do robô e o ângulo final pretendido a partir da qual o robô começa a desacelerar na rotação sobre ele mesmo.

- "TOL_FINTHETA": Máximo erro angular permitido para se considerar que o robô chegou à posição de destino, em radianos.
- "LIN_VEL_NOM": Velocidade linear nominal do robô, em m/s.
- "LIN_VEL_DA": Velocidade linear do robô quando este está em modo de desaceleração, em m/s.
- "W_NOM": Velocidade angular nominal do robô, em radianos/s.
- "W_DA": Velocidade angular do robô este está em modo de desaceleração na rotação, em radianos/s.
- "GAIN_FORW": Factor de multiplicação pelo erro de orientação em relação à posição final, para se obter a velocidade angular, quando o robô se está a movimentar à velocidade nominal.
- "GAIN_DA": Factor de multiplicação pelo erro de orientação em relação à posição final, para se obter a velocidade angular, quando o robô está em modo de desaceleração.

10. Decisão

Nó [/decision] <u>Publicações:</u> • /pose_goal [geometry_msgs/PoseStamped] <u>Subscrições:</u> • /tf [tf2_msgs/TFMessage] • /tf_static [tf2_msgs/TFMessage] <u>Serviços:</u> Nenhum	
--	--

O programa de decisão é a camada de mais alto nível em todo o sistema ROS que é executada neste robô, tendo como função a atribuição de objetivos de navegação ao mesmo, ao longo do seu período de operação. Atualmente, este nó carrega um conjunto de poses-objetivo para onde o robô se deve deslocar ao longo do seu percurso, autonomamente.

Parâmetros:

- "min_position_error": Máximo erro de posição permitido para que a chegada ao ponto de destino seja considerada terminada, em metros.
- "min_orientation_error": Máximo erro angular permitido para que a chegada à posição de destino seja considerada terminada, em graus.

3.2.7 Criação de mapas

Como já foi referido anteriormente, para que os algoritmos de localização utilizados (AMCL e Perfect Match) possam estimar a pose do robô, é necessário que tenham por base um mapa do ambiente onde o robô vai navegar. O mapa é representado por uma imagem que descreve o estado de ocupação de cada célula no mundo, através do valor da cor atribuída a cada pixel da imagem. Os pixels de cor próxima a branco são considerados como "livres", os pixels de cor próxima a preto são considerados como "ocupados" e os pixels com valores intermédios (cor cinzenta) são considerados como "desconhecidos". Os limiares considerados para estes três estados são definidos num ficheiro com a *metadata* do mapa, sendo a probabilidade de ocupação de uma determinada célula calculada por $\frac{(255 - \text{nível_cinza_célula})}{255}$, em que os valores por *default* consideram uma determinada célula como ocupada se a probabilidade for superior a 0.65 e completamente livre se a probabilidade for inferior a 0.196; os valores intermédios classificam as células com estado desconhecido.

A criação do mapa para a navegação do robô Clever foi realizada recorrendo ao próprio laser do robô (URG-04LX-UG01 - 3.1.5). Visto que este laser é o utilizado durante a sua normal operação, considera-se de grande relevância que a perceção do meio no momento da criação do mapa e durante a navegação seja o mais similar possível. Para a movimentação do robô durante a geração do mapa utilizou-se o controlo por *joystick*, com os nós descritos na secção 3.2.5, referentes ao lançamento apenas do nível de *hardware*. O programa de ROS escolhido para esta tarefa foi o *hector_slam*²⁷ [48], devido ao seu baixo peso computacional (fator relevante na aplicação concreta ao robô Clever) e aos seus bons resultados, dando a possibilidade de se realizar o mapeamento sem a necessidade da odometria. Outra opção seria a utilização do *gmapping*²⁸, que requer necessariamente dados (precisos) da odometria e de um laser montado horizontalmente no robô.

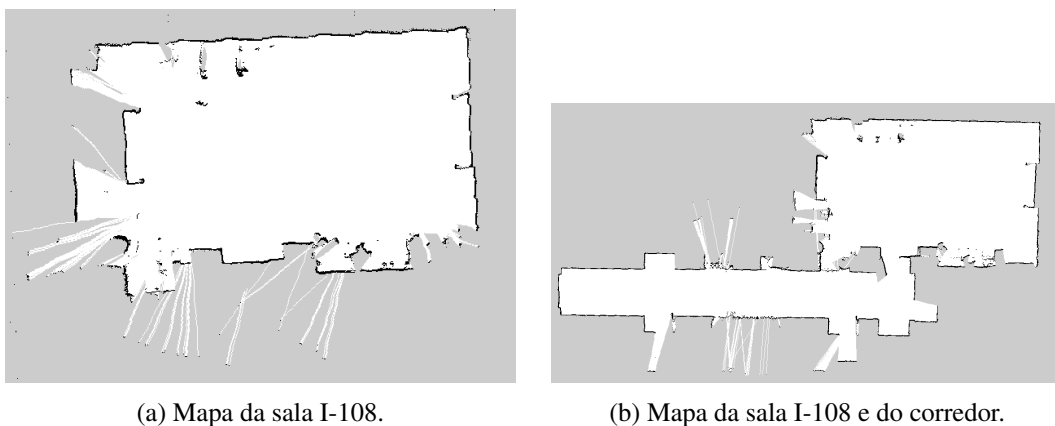


Figura 3.27: Mapas criados para a navegação e localização do robô, utilizando o laser URG-04LX-UG01.

Na figura 3.27 estão os mapas resultantes do procedimento descrito até aqui. Um dos mapas foi criado apenas na sala I-108 (figura 3.27a) enquanto que o outro também incluiu o mapeamento

²⁷http://wiki.ros.org/hector_slam

²⁸<http://wiki.ros.org/gmapping>

do corredor (figura 3.27b).

3.3 Aplicações da plataforma robótica

Este projeto tem em vista a utilização deste robô como uma plataforma genérica para a sua aplicação em diversas atividades distintas. O objetivo principal passou pelo desenvolvimento de um sistema robótico de base que conseguisse navegar autonomamente e que possuísse mecanismos de auto-localização robustos, para que a sua utilização em diversas atividades fosse adaptada com a maior facilidade possível.

Visto que este robô possui uma baixa altura e todo o *hardware* e *software* do sistema de navegação e localização se encontram a essa baixa altura, abrem-se possibilidades para a colocação de diversas extensões por cima do robô, sendo possível acrescentar-se, de forma modular, um conjunto de funcionalidades.

Essencialmente, manteve-se em vista as seguintes aplicações para a plataforma robótica:

- **Monitorização e vigilância**

Ao colocar uma câmara de vigilância e diversos sensores (*e.g.*, temperatura, monóxido de carbono, fumo, fogo, etc) por cima do robô, é possível controlar o ambiente onde este irá operar.

- **Publicidade**

Com vista a despertar a atenção e a curiosidade das pessoas, podem ser integradas colunas de som, LEDs, painéis ou cartazes por cima do robô, que servirão para efeitos publicitários, enquanto este se desloca num determinado espaço público.

- **Logística**

Aproveitando as pequenas dimensões e a pequena altura do robô, este pode ser aplicado em operações de logística e transporte de objetos. Mais especificamente, desenvolveu-se um sistema de transporte de prateleiras móveis (com rodas), em que o robô se desloca para debaixo destas e as arrasta para os locais pretendidos. Para se movimentar as prateleiras, integrou-se uma placa com solenóides, que atuam um conjunto de cilindros metálicos, fazendo-os movimentar-se verticalmente: nos momentos de encaixe na prateleira os cilindros sobem, e quando estas já estão na posição pretendida e o robô quer soltá-las movem-se os cilindros para baixo. Este sistema está descrito com maior detalhe na Secção 3.1.9.

Capítulo 4

Troca automática de baterias

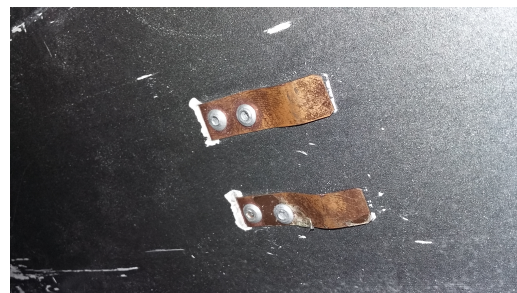
No sentido de se desenvolver um sistema capaz de trocar de baterias automaticamente e de forma autónoma, *i.e.*, sem a intervenção de nenhum humano, desenvolveu-se um conjunto de ferramentas e pensou-se em algumas estratégias com vista a esse objetivo.

4.1 Hardware

A alimentação elétrica do robô será realizada através de uma caixa com 2 baterias de 12 V ligadas em série, no seu interior, perfazendo 24 V. Na superfície superior da caixa existem 3 contactos metálicos (Figura 4.1a): o contacto central tem o pólo positivo (+) e os dois contactos laterais têm o pólo negativo (-). Esta disposição tem a vantagem de permitir que a caixa das baterias seja acoplada indiferentemente das duas possíveis orientações, visto que na parte inferior do robô existem apenas 2 contactos metálicos (Figura 4.1b), um positivo no centro do robô, e outro negativo, num dos lados. Assim, não há o risco de curto circuito na eletrônica do robô, caso o pólo positivo fosse ligado acidentalmente ao pólo negativo.



(a)



(b)

Figura 4.1: (a) Caixa com as baterias externas do robô, com três contactos de cobre. (b) Dois contactos de cobre na parte inferior do robô, a partir dos quais é recebida a alimentação elétrica para o funcionamento de todos os componentes do robô.

Para o encaixe automático das baterias desenvolveu-se de raiz um sistema que inclui 4 servomotores, dispostos paralelamente 2 a 2, cada um com uma peça de alumínio aparafusada, com a

forma de uma garra (Figura 4.3), para entrar em contacto com as saliências (cabeças de parafusos - Figuras 4.2a e 4.2b) da caixa de baterias e conseguir elevá-la. A orientação das garras são simétricas duas a duas, em que as 4 garras têm o gancho a apontar para o centro do robô, com vista a otimizar a distribuição do peso e a reduzir a força exercida por cada um dos servomotores (Figuras 4.2c e 4.2d). Ainda assim, depois de se realizarem um conjunto de testes a este mecanismo, verificou-se que os 4 servomotores não tinham binário suficiente para a elevação da caixa das baterias, a partir do chão. No caso desta estar elevada a uma altura superior (5-7 cm) do nível do chão, os servomotores já são capazes de elevar a caixa até esta contactar com o robô, e segurá-la permanentemente.

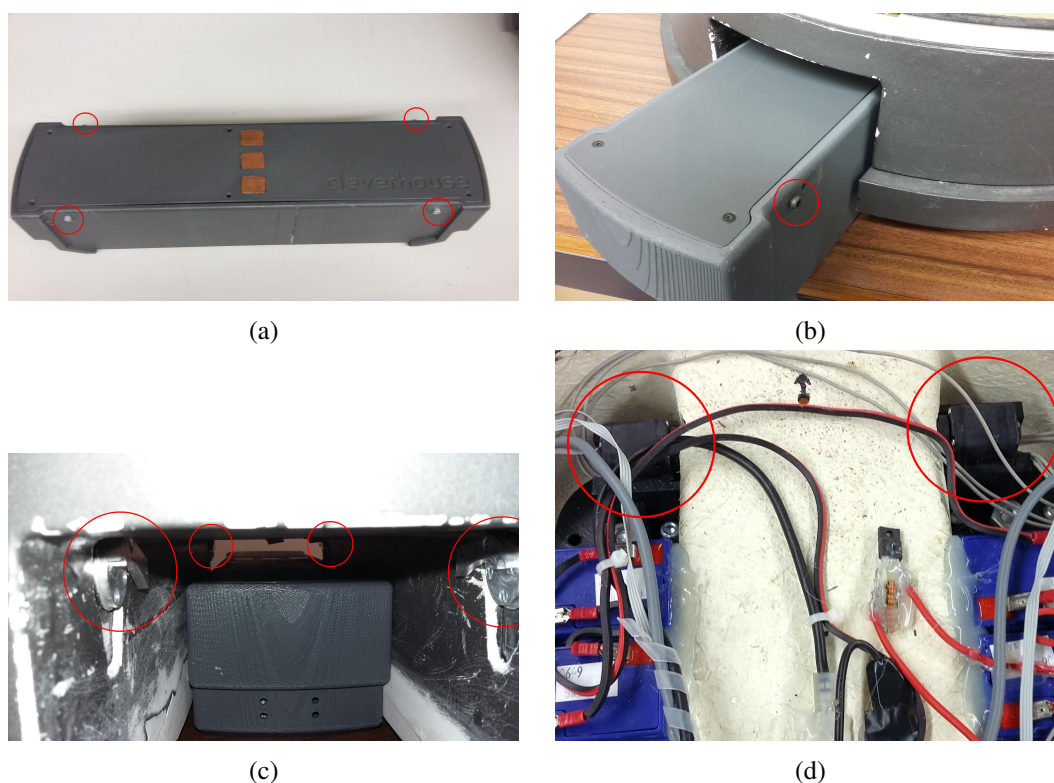


Figura 4.2: (a) e (b) Caixa das baterias com quatro pinos de encaixe destacados a vermelho. (c) Estrutura onde é realizado o encaixe das baterias, possuindo quatro servomotores laterais (realçados a vermelho) com uma peça de alumínio integrada. (d) Vista interior de dois dos quatro servomotores integrados no andar inferior do robô.

Deste modo, foi necessário pensar-se numa estação que facilitasse o processo de elevação e o encaixe das baterias, auxiliando e minimizando a força necessária que os servomotores teriam de exercer, colocando a caixa das baterias a uma altura favorável para o robô. A ideia será dotar a estação de uma entrada com guias: mais largas na extremidade oposta à estação, ficando cada vez mais estreitas à medida que se aproximam do ponto da troca de bateria.



Figura 4.3: Peça de alumínio acoplada aos servomotores, para o levantamento automático da caixa de baterias.

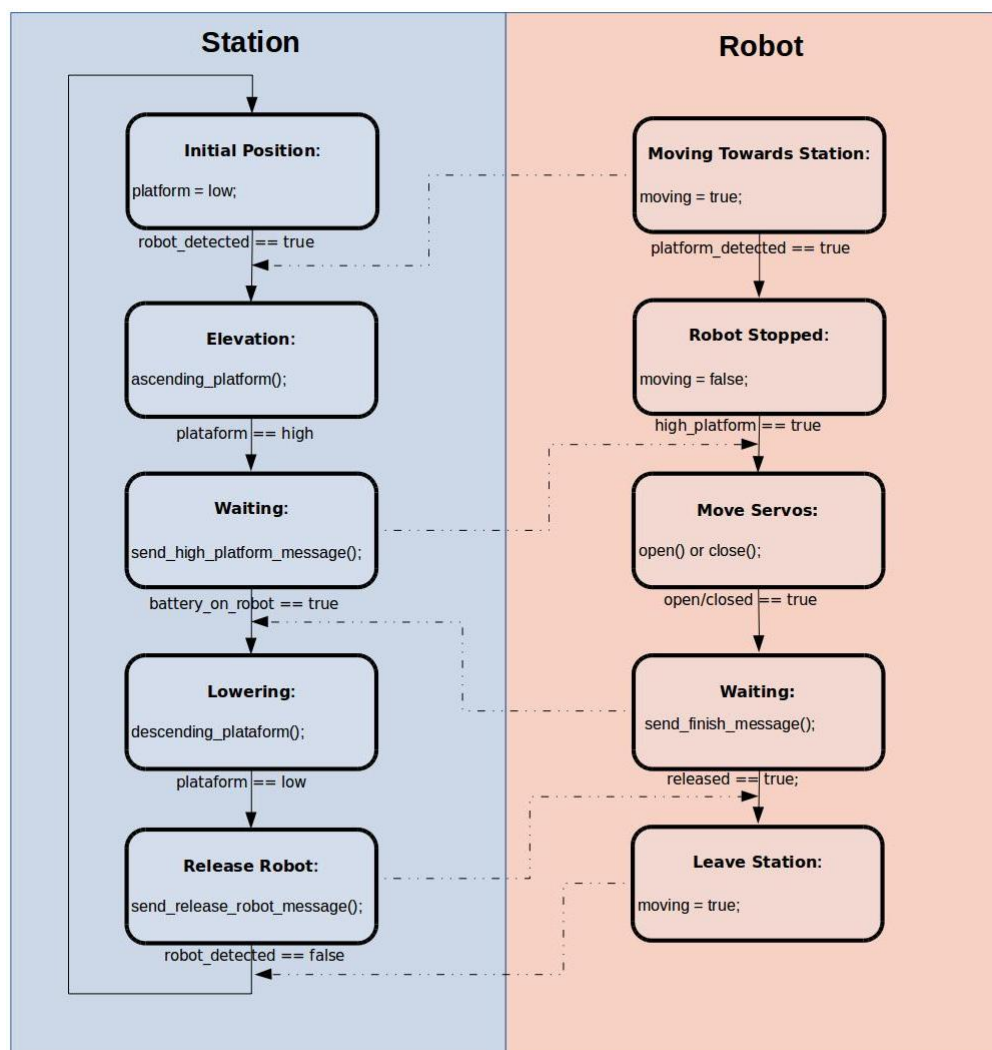


Figura 4.4: Diagrama representativo do processo de troca de baterias, com comunicação entre a estação e o robô.

Desta forma, é possível corrigir um eventual desajuste de posição do robô em relação à estação,

caso a sua localização possua algum erro, conduzindo o robô para a posição correta. Depois, ao colocar um contacto mecânico na estação, esta será capaz de detetar o momento em que o robô chega à posição da troca de bateria, simultaneamente travando-o. Neste momento, poderá ligar os motores para que estes elevem a plataforma onde se colocará a bateria descarregada, enviando um comando de controlo ao robô, via *wireless*, para que este atue os servomotores e solte a bateria, informando a estação no momento em que esta ação termine. Ao receber este comando, a estação voltará a descer a plataforma para a posição baixa, libertando o robô. O robô, após ter colocado a caixa com as baterias na estação, ficando estas a carregar, terá de se deslocar até ao outro ponto de troca, onde, por um processo idêntico, encaixará outra caixa de baterias, previamente carregadas, e continuará a sua operação. A movimentação do robô sem a alimentação elétrica principal é assegurada por um conjunto de pequenas baterias internas que fornecem a energia necessária para este processo. Quando uma nova caixa com baterias carregadas é colocada no robô estas também carregam as baterias internas para que o próximo processo de troca automática volte a realizar-se de igual forma.

Na Figura 4.4 está uma representação esquemática genérica do processo de troca de baterias até aqui descrito, sendo muito similar tanto no caso de encaixe da bateria carregada como no caso de reposição da bateria descarregada.

4.2 Programa de controlo dos servomotores

Com vista a integrar o mecanismo de rotação dos servomotores com o resto do *software* do sistema, foi necessário desenvolver um programa, em C++, para o ambiente ROS. Parte da documentação referente a este programa já foi apresentada na Secção 3.2.6.

Na figura 4.5 está representado o diagrama de classes em C++ do controlador dos servomotores. Este programa possui apenas uma classe ("ServosController") que coordena todo o sistema, tendo-se utilizado a biblioteca USB2Dynamixel¹ que fornece um conjunto de funções para a comunicação via porta série com os servomotores.

O programa recebe como parâmetros:

- O número de servomotores;
- A velocidade nominal de rotação dos servomotores, em graus/s;
- O erro máximo para se considerar que chegaram à posição de destino;
- A posição inicial dos servomotores, referente ao estado de "aberto", *i.e.*, quando as garras estão na posição de soltar as baterias.
- A posição final dos servomotores, referente ao estado de "fechado", *i.e.*, quando as garras estão na posição de prender as baterias.

¹http://support.robotis.com/en/software/dynamixel_sdk/usb2dynamixel/usb2dxl_linux.htm

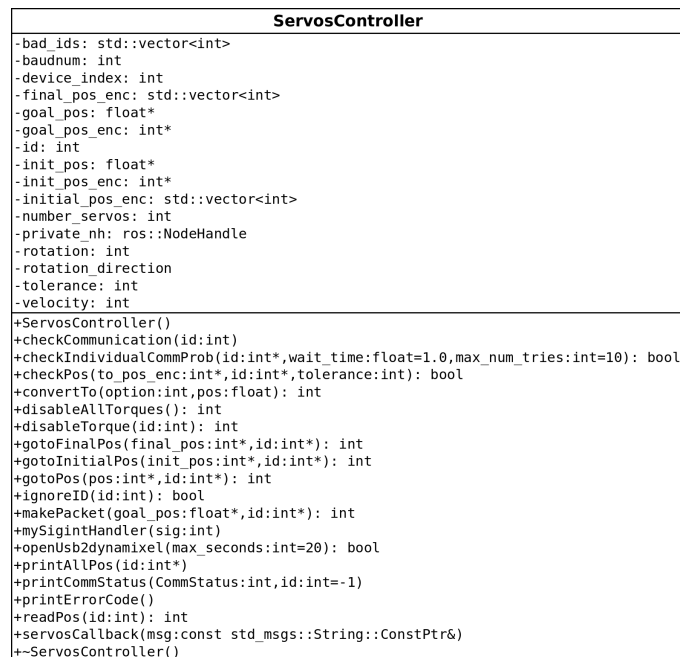


Figura 4.5: Diagrama de classes UML do programa de controlo dos servomotores, desenvolvido em C++ e aplicado em ROS.

Para esta aplicação, é importante que a movimentação dos motores seja realizada de forma síncrona e rodando todos à mesma velocidade. Assim, quando se requisita a ação de abertura ou fecho das garras, os servos terão de rodar para os respetivos lados (*e.g.*, dois dos servomotores rodarão no sentido horário e os outros dois no sentido anti-horário), de forma síncrona, para que a movimentação e a força exercida pelos motores seja o mais uniforme possível. Deste modo, é apenas necessário enviar um comando de abertura ("Open") ou fecho ("Close") das garras para o tópico respetivo e os 4 servomotores rodam de forma a completar a ação, sendo considerada finalizada só quando todos chegam à posição de destino (idealmente chegam ao mesmo tempo).

Durante o processo de comunicação com os motores não é incomum haverem falhas nas transmissão dos pacotes, ou estes chegarem corrompidos. Com isto em mente, o programa faz um constante controlo desses erros antes e durante a movimentação dos servomotores, retransmitindo os pacotes sempre que essas falhas se verificam. Por outro lado, os servos AX-12 da Dynamixel possuem *bits* de erro (alarmes) cujos estados podem ser monitorizados e, de acordo com o *bit* ativo, agir em conformidade. Como exemplo, quando os servomotores estão a atuar no processo de encaixe das baterias, estas podem ficar presas ou estarem numa posição que não permita a sua elevação, o que ativa o alarme de carga excessiva (*overload*). Quando este alarme é detetado, o programa desliga automaticamente o torque dos motores, impedindo que estes fiquem a ser forçados e reinicia esse bit de erro por *software*.

Para além do que já foi descrito, este nó disponibiliza um serviço que permite a "consulta" momentânea da posição dos servomotores, durante o seu processo de rotação, permitindo algum

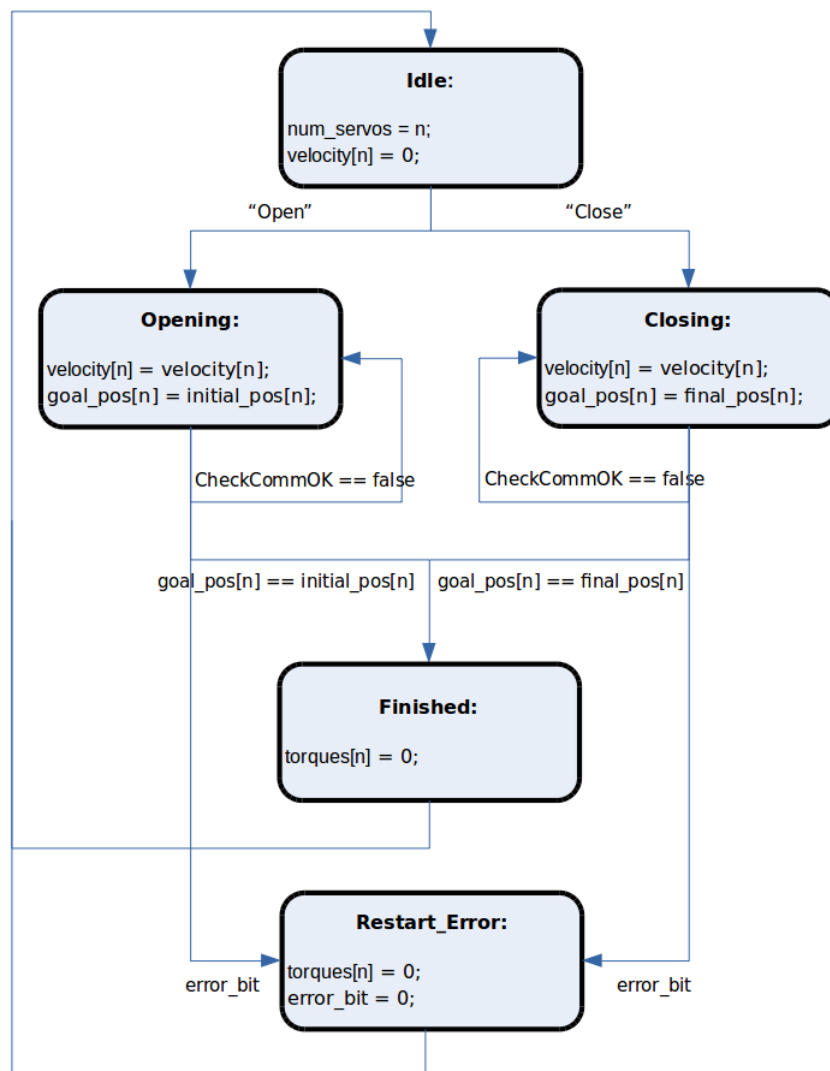


Figura 4.6: Máquina de estados do programa de controlo dos servomotores.

tipo de ação, por parte de outro nó de decisão ou supervisão, que obtenha esta informação.

Na Figura 4.6 está um esquema representativo, na forma de máquina de estados, do funcionamento do programa de controlo dos servomotores. Na condição de verificação da chegada à posição de destino está implícita a tolerância fornecida como parâmetro de entrada.

Capítulo 5

Análise dos métodos de localização

5.1 Análise e comparação de várias configurações do Perfect Match

Nesta secção será feita uma análise pormenorizada do algoritmo Perfect Match utilizado, explicando os parâmetros de entrada do programa e as extensões ao algoritmo original, ainda não documentadas. De seguida, irá realizar-se uma comparação e análise prática de diferentes configurações do algoritmo, com vista a achar um bom conjunto de parâmetros que torne o modelo o mais robusto e preciso possível.

5.1.1 Análise dos parâmetros do Perfect Match

Parâmetros gerais do Perfect Match

- "GlobalFrameId": Referencial global.
- "ScanFrameId": Referencial do laser do robô.
- "OdomFrameId": Referencial da odometria do robô.
- "LaserMeasuresNum": Número máximo (total) de pontos devolvidos pelo laser.
- "SaveLocalizationStatePeriod": Período com que o programa guarda a localização atual para o disco do computador.
- "write_pose_result_to_tf": Booleano para ativar ou desativar a publicação da estimativa da localização a partir da transformada entre o referencial global e a odometria.
- "OperationMode": Modos de operação: apenas com a odometria ("OdomOnly"), apenas com os sensores ("SensorsOnly"), com odometria e sensores mas sem fusão sensorial através do Filtro de Kalman, utilizando a odometria apenas para a previsão da propagação da pose do robô antes da otimização do RPROP ("SensorsOdomNoFusion"), com odometria e sensores, fundindo-os com implementação do Filtro de Kalman ("SensorsOdomFusion").

- "MaxErrorDistFilterEnabled1": Booleano que ativa ou desativa a primeira rejeição de *outliers*, utilizando o valor do parâmetro *LaserFilterMaxDist1*.
- "LaserFilterMaxDist1": Limiar (menos exigente) da distância máxima permitida a um ponto detetado pelo laser e o objeto no mapa mais próximo, para decidir se esse ponto é descartado (*outlier*), não entrando no algoritmo.
- "MaxErrorDistFilterEnabled2": Booleano que ativa ou desativa a segunda rejeição de *outliers*, utilizando o valor do parâmetro *LaserFilterMaxDist2*.
- "LaserFilterMaxDist2": Limiar (mais exigente) da distância máxima permitida a um ponto detetado pelo laser e o objeto no mapa mais próximo, para decidir se esse ponto é descartado (*outlier*), não entrando no algoritmo.
- "WeightenMeasureDistEnable": Booleano que ativa ou desativa a função de custo do erro com pesos diferentes para os pontos mais próximos e mais distantes.
- "Lc": Coeficiente para a rejeição de pontos com grandes erros (*outliers*), na função de custo.
- "MinPercentValideSensorData": Percentagem mínima de pontos do laser válidos ($\frac{n_pontos_laser_validos}{n_pontos_laser_total} \times 100\%$) para que o algoritmo estime a posição do robô.
- "initial_pose_x": Coordenada x da posição inicial do robô, utilizada para iniciar o algoritmo.
- "initial_pose_y": Coordenada y da posição inicial do robô, utilizada para iniciar o algoritmo.
- "initial_pose_ori": Orientação da posição inicial do robô, utilizada para iniciar o algoritmo.

Parâmetros para o algoritmo de otimização RPROP

- "NumIterationMax": Número máximo de iterações permitidas para o algoritmo de otimização RPROP (Resilient Back-Propagation).
- "OptimizationStopTranslation": Condição de paragem do algoritmo RPROP, quando o passo de translação for inferior ao valor deste parâmetro.
- "OptimizationStopRotation": Condição de paragem do algoritmo RPROP, quando o passo de rotação for inferior ao valor deste parâmetro.
- "DeltaUpdateInitPosition": Passo inicial para a componente da posição do algoritmo RPROP.
- "DeltaUpdateInitOrientation": Passo inicial para a componente da orientação do algoritmo RPROP.

Parâmetros para a fusão sensorial do Filtro de Kalman

- "SensorsKErrorXY": Factor do ruído de posição dos sensores, para o cálculo da segunda derivada.
- "SensorsKErrorTheta": Factor do ruído de orientação dos sensores, para o cálculo da segunda derivada..
- "DriftTranslationDeltaD": Desvio na translação da odometria pela componente de translação, para a propagação do erro.
- "DriftTranslationDeltaRot": Desvio na translação da odometria pela componente de rotação, para a propagação do erro.
- "DriftRotationDeltaRot": Desvio na orientação da odometria pela componente de rotação, para a propagação do erro. .
- "StateMinVarXY": Mínima covariância ($x \times y$) permitida para os estados do Filtro de Kalman.
- "StateMinVarTheta": Mínima covariância ($\theta \times \theta$) permitida para os estados do Filtro de Kalman.

O algoritmo do Perfect Match utilizado para o presente trabalho de dissertação é baseado no artigo [5] que implementa um conjunto de melhoramentos e estende o algoritmo original [3] para outro tipo de aplicações: robôs de serviços, industriais, utilizando outro tipo de sensores (*e.g.*, lasers), em oposição às técnicas de visão (câmeras). Nesta implementação, em vez de se utilizar um número fixo de iterações para o algoritmo de otimização RPROP, define-se um número máximo de iterações (*NumIterationMax*) ou, no caso do método convergir para um valor de erro inferior a um determinado limiar previamente definido (*OptimizationStopTranslation*, *OptimizationStopRotation*) o algoritmo pára.

Em adição, é definido um momento de pré-processamento dos dados recebidos pelo laser, fazendo-se uma rejeição dos pontos que estiverem mais distantes (que um determinado *threshold*) do obstáculo mais próximo no mapa, indicando a presença de objetos/pessoas externas ao mapa (*outliers*). Na realidade o processo de rejeição dos *outliers* é realizado duas vezes: uma primeira vez utilizando um valor de distância máxima menos exigente (*LaserFilterMaxDist1* - por exemplo 0.5 m), descartando-se os pontos mais distantes que este valor e de seguida executando-se o RPROP; na segunda vez, a partir do resultado do processo anterior, utiliza-se um valor de distância mais exigente (*LaserFilterMaxDist2* - por exemplo 0.1 m) e volta-se a correr o algoritmo do RPROP, obtendo-se o resultado final. A utilização de dois filtros consecutivos, com limiares diferentes, em oposição a apenas um filtro com um valor de distância mais exigente deve-se ao facto de, numa primeira fase, se o algoritmo estiver com uma posição desajustada (com algum erro) em relação à posição real do robô, pode descartar um número considerável de pontos que na realidade não correspondem a *outliers*. Isto porque todo o algoritmo tem como base a posição estimada pelo próprio, e no caso dessa posição estar com algum erro temporário, resulta num cálculo errado das

distâncias entre os pontos detetados pelo laser (5.1) e a posição esperada do obstáculo mais próximo no mapa, levando a uma rejeição errada de pontos corretos. Com a utilização de dois filtros, inicialmente rejeita-se os objetos muito longe do esperado, ajusta-se a posição estimada pelo algoritmo RPROP e, com base nesta nova estimativa, faz-se a rejeição dos pontos com o filtro mais exigente, havendo uma maior probabilidade de ter sido utilizada uma posição correta do robô.

Para além das funcionalidades descritas, o programa permite utilizar uma opção que atribui pesos diferentes aos pontos detetados pelo laser, de acordo com a sua distância ao robô (*Weighten-MeasureDistEnable*). Isto porque, um objeto que esteja próximo do robô é detetado por um maior número de pontos do laser, para o mesmo comprimento, do que se estivesse mais longe, devido ao efeito radial dos feixes do laser. Por consequência, na fase de atribuição dos pesos aos pontos utilizados pelo algoritmo, para o cálculo da função de custo, os objetos ou as características do ambiente (*e.g.*, paredes, mobília) que estiverem mais próximos do robô terão maior peso. Com vista a equilibrar este fenómeno, há uma compensação linear dos pesos dos pontos em função da distância, através de: $E_i = dist_i \cdot E_i$, com $dist_i = \sqrt{(p_y - p_{robot})^2 + (p_x - p_{robot})^2}$.

Numa visão geral, do início da execução do algoritmo até ao resultado final, há 3 fases distintas:

1. **Pré-cálculo da matriz das distâncias de cada célula do mapa ao obstáculo mais próximo, assim como as matrizes de gradiente (na direção do x e do y) e cálculo do erro de ajuste.**

$$P = \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} x_{robot} \\ y_{robot} \end{bmatrix} + \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} \quad (5.1)$$

Na Equação 5.1 está o método para calcular a posição de cada ponto detetado pelo laser em coordenadas globais (p_x, p_y) , a partir da posição (x_{robot}, y_{robot}) e orientação (θ) atual do robô e da distância relativa dos pontos do laser ao mesmo, (d_x, d_y) .

Utilizando a matriz das distâncias, para cada ponto P detetado pelo laser (em coordenadas globais), obtém-se a distância ($error_i$) desse ponto ao obstáculo do mapa mais próximo (na prática, numa imagem binária, é a distância à célula/pixel ocupada mais próxima), através da consulta a uma matriz pré-calculada.

2. **Execução do algoritmo de otimização RPROP, minimizando a função de custo.**

Após a determinação da distância entre os pontos detetados pelo laser e o obstáculo mais próximo presente no mapa, define-se uma função de custo (Figura 5.1), que, com base no erro de distância ($error_i$) para cada ponto i do laser, irá ser minimizada (através do método RPROP) e convergirá para a posição estimada do robô.

$$E = \sum_{i=1}^n E_i \quad (5.2)$$

Depois de calcular o valores da função de custo para cada ponto i do laser, através da expressão 5.3, utiliza-se o algoritmo RPROP para minimizar E (5.2).

$$E_i = \frac{L_c^2 \cdot error_i^2}{(L_c^2 + error_i^2)^2} \quad (5.3)$$

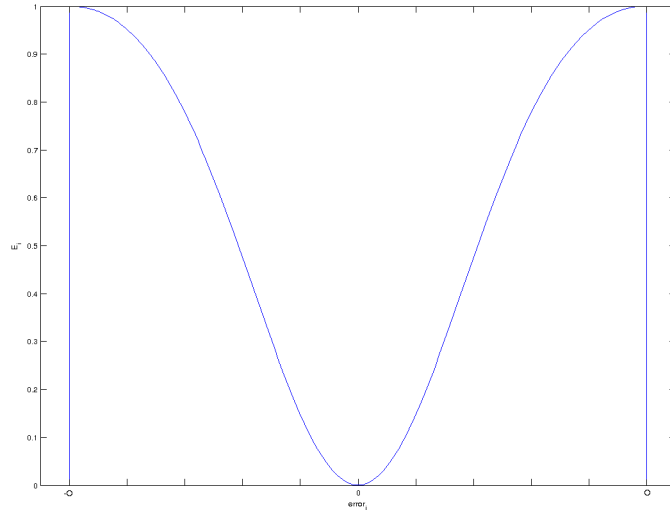


Figura 5.1: Gráfico da função de custo, com o limiares de rejeição de outliers (O).

3. Estimação da covariância, através do cálculo da segunda derivada.

Por fim, calcula-se a segunda derivada (Equações 5.4, 5.5 e 5.6) para, através de uma heurística, se obter a variância da estimativa de pose dada pelo algoritmo.

As variâncias da posição ($\sigma_{x,t}^2, \sigma_{y,t}^2$) e orientação do robô ($\sigma_{\theta,t}^2$), no instante de tempo t , são calculada a partir do conjunto de equações 5.7, em que $(\hat{x}_{t+1}, \hat{y}_{t+1}, \hat{\theta}_{t+1})$ são as estimativas da posição e orientação do robô, aplicando o modelo de movimentação do mesmo, com base nos dados recebidos pela odometria (velocidade linear v e velocidade angular ω). Estas variâncias serão depois utilizadas no processo de fusão sensorial do Perfect Match, quando se aplica o Filtro de Kalman.

$$\frac{\partial^2 E}{(\partial p_x)^2} \approx \sum_{i=1}^n E_i''(P_i) \cdot \left(\frac{\partial (error_i)}{\partial x} \right)^2 \quad (5.4)$$

$$\frac{\partial^2 E}{(\partial p_y)^2} \approx \sum_{i=1}^n E_i''(P_i) \cdot \left(\frac{\partial (error_i)}{\partial y} \right)^2 \quad (5.5)$$

$$\begin{aligned} \frac{\partial^2 E}{(\partial p_\theta)^2} \approx \sum_{i=1}^n (E_i''(P_i) \cdot \left(\frac{\partial(\text{error}_i)}{\partial x} (-\sin \theta - \cos \theta) P_i + \frac{\partial(\text{error}_i)}{\partial y} (\cos \theta - \sin \theta) P_i \right)^2 + \\ E_i' \cdot \left(\frac{\partial(\text{error}_i)}{\partial x} (-\cos \theta + \sin \theta) P_i + \frac{\partial(\text{error}_i)}{\partial y} (-\sin \theta - \cos \theta) P_i \right) \end{aligned} \quad (5.6)$$

$$\begin{cases} \sigma_{\hat{x},t+1}^2 = \sigma_{x,t}^2 + \alpha(\hat{x}_{t+1} - x_t)^2 \\ \sigma_{\hat{y},t+1}^2 = \sigma_{y,t}^2 + \alpha(\hat{y}_{t+1} - y_t)^2 \\ \sigma_{\hat{\theta},t+1}^2 = \sigma_{\theta,t}^2 + \alpha(\hat{\theta}_{t+1} - \theta_t)^2 \end{cases} \quad (5.7)$$

5.1.2 Ideias para melhoramento do desempenho do Perfect Match

O Perfect Match, apesar de ser um algoritmo de localização robusto e que devolve boas estimativas, ainda pode ser melhorado. Desde a sua implementação original veio sofrendo alterações e extensões com o objetivo de o tornar mais fiável, com menor sensibilidade a perturbações ou até para ser aplicado em áreas diferentes, com diferentes requisitos. Assim, ao longo deste projeto foram-se explorando alguns possíveis melhoramentos ao algoritmo Perfect Match utilizado.

Por vezes o algoritmo pode fazer um ajuste errado com uma parte do mapa que na realidade não está dentro do campo de visão do robô e é fisicamente impossível que este a detetar. Por exemplo, se o robô estiver a navegar por um local com zonas muito similares e indistinguíveis entre si (*e.g.*, paredes com formas idênticas, consecutivas, entre uma sala e um corredor), o processo de *matching* pode estar a ser realizado com a parede errada, a qual não é possível que o robô detete por existir outra no intermédio. Neste caso o robô iria devolver uma estimativa errada da posição do robô, uns metros ou centímetros mais deslocada da posição verdadeira. Para contrariar este fenómeno, desenvolveu-se um método de *beamtracking*, em que se verificava se ao longo do feixe do laser existiam obstáculos no mapa, indicando que, caso existissem, seria fisicamente impossível para o robô detetar algo para além desse ponto. Desta forma, o algoritmo alterava a posição do ponto detetado pelo laser para o obstáculo mais próximo do mapa, na direção desse feixe do laser, impedindo que o fenómeno descrito ocorresse. Apesar desta implementação poder revelar-se vantajosa em alguns casos, aumenta muito o peso computacional do algoritmo que tem como uma das suas principais características e vantagens a rapidez e leveza computacional. Isto porque a verificação das várias posições do mapa, ao longo de cada feixe de um varrimento do laser é um processo pesado e demorado.

Outra ideia, ainda não implementada, passa pela divisão do mapa global do robô em sub-mapas de zonas específicas, em que se ia comutando entre sub-mapas, de forma automática, à medida que o robô se ia deslocando ao longo do edifício. Este modo de operação traz vantagens para a convergência do processo de ajuste de mapa, pois há menor probabilidade de haver vários mínimos locais à medida que as dimensões do mapa diminuem. Por isso, a estimativa seria mais robusta e fiável para cada sub-mapa, impedindo também a ocorrência da situação referida anteriormente, em

que o algoritmo poderia fazer um *matching* com características do mapa que não seriam possíveis de detetar, eliminando também esse problema. Apesar de poder ser uma ideia promissora, a sua implementação não é trivial, devido à necessidade de sincronização dos vários sub-mapas com a movimentação do robô, o que é especialmente difícil nos locais de transição entre duas divisões (*e.g.*, passagem de uma sala para um corredor de um edifício). Assim, é um desafio para ser explorado no futuro.

5.1.3 Comparação prática de diferentes configurações do Perfect Match

Nesta secção serão demonstrados alguns testes práticos ao algoritmo de localização Perfect Match, realizados no robô Jarvis, com acesso a *groundtruth*¹ mas utilizando o laser URG-04LX-UG01 (integrado no robô Clever - 3.1.5), a 23 cm do chão (na mesmas condições que no robô Clever), como sensor para a recolha dos dados necessários para o algoritmo de localização. O robô movimentou-se ao longo de uma trajetória aproximadamente circular, a 0.1 m/s, representado na figura 5.2.

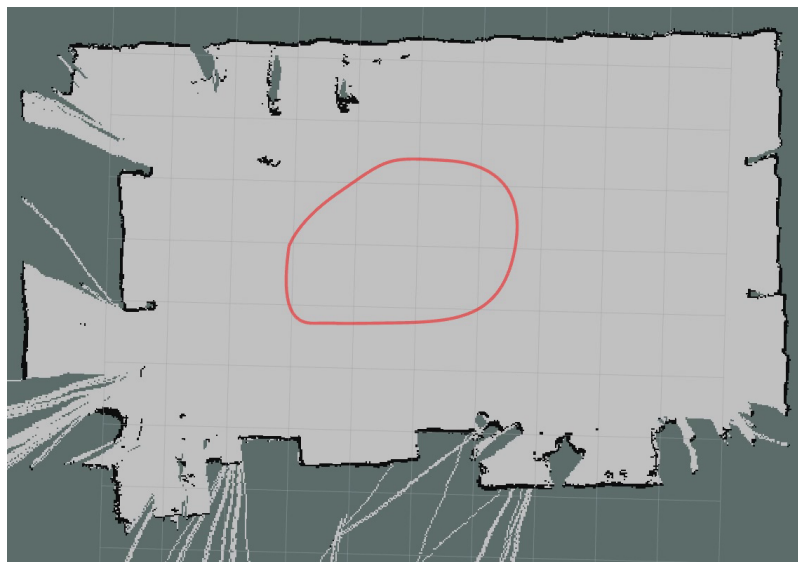


Figura 5.2: Trajetória descrita pelo robô durante os testes, sobreposta ao mapa utilizado para a navegação, visualizada através do *rviz*, em ROS.

Estes testes debruçar-se-ão na verificação da precisão e robustez do algoritmo, nestas circunstâncias, com diferentes parametrizações e num ambiente com variadas características: sem *outliers*, com diferentes níveis de obstáculos e sujeito a erros forçados de odometria.

As configurações terão como intuito demonstrar, na prática, o impacto real de alguns dos parâmetros do Perfect Match, mantendo outros, menos relevantes (ou com menos interesse para estes estudos), constantes. Todas as configurações terão os parâmetros com os valores apresentados na Tabela 5.1.

¹Para o *groundtruth* utilizou-se o laser SICK NAV350.

Tabela 5.1: Parâmetros comuns a todas as configurações desta secção.

Parâmetros	Valores
LaserMeasuresNum	683 pontos
NumIterationMax	50
OptimizationStopTranslation	0.0
OptimizationStopRotation	0.0
MinPercentValideSensorData	0.0 %
DeltaUpdateInitPosition	0.1 m
DeltaUpdateInitOrientation	0.1 graus
Lc	0.1
DriftRotationDeltaRot	0.001
DriftTranslationDeltaD	0.001
DriftTranslationDeltaRot	0.0003
SensorsKErrorTheta	0.001
SensorsKErrorXY	1.0
StateMinVarTheta	0.001
StateMinVarXY	0.01

Utilizou-se o número máximo de pontos devolvidos pelo laser URG-04LX-UG01: 683.

Definiu-se um número máximo de 50 iterações para o algoritmo RPROP, colocando o critério de paragem através dos passos mínimos do RPROP (*OptimizationStopTranslation* e *OptimizationStopRotation*) a 0, forçando a convergência total do algoritmo antes de devolver o resultado final (ou utilizar as 50 iterações), com vista a obter a melhor estimativa possível.

Não se coloca limite mínimo para o número de pontos totais utilizados pelo Perfect Match (que levaria à paragem temporária do programa), pois esta é uma medida de segurança e para o efeito destes testes não é relevante integrá-la, pois há interesse em verificar o comportamento do algoritmo em situações desfavoráveis.

Os parâmetros referentes ao passo inicial atribuído ao método RPROP (*DeltaUpdateInitPosition* e *DeltaUpdateInitOrientation*) deverão ser ajustados de acordo com o grau de *matching* esperado a cada iteração do algoritmo (cerca de alguns centímetros, para este caso), com vista a minimizar o número de iterações necessárias para que convirja. De qualquer modo, desde que não lhes sejam atribuídos valores demasiado altos ou demasiado baixos, esse desajuste inicial será traduzido em apenas algumas (muito poucas) iterações extra.

Os restantes parâmetros referentes ao Filtro de Kalman são intrínsecos ao robô e foram calibrados previamente a este projeto, resultando nos valores apresentados na Tabela 5.1.

5.1.3.1 Ambiente sem outliers

O conjunto de testes apresentados nesta secção foram realizados na sala I108, no campo RoboCup, sem outliers.

Configuração 1 e Configuração 2

As primeiras duas configurações serão denominadas por *Config1_OF* e *Config2_OF_F1F2* e terão os parâmetros apresentados na Tabela 5.2.

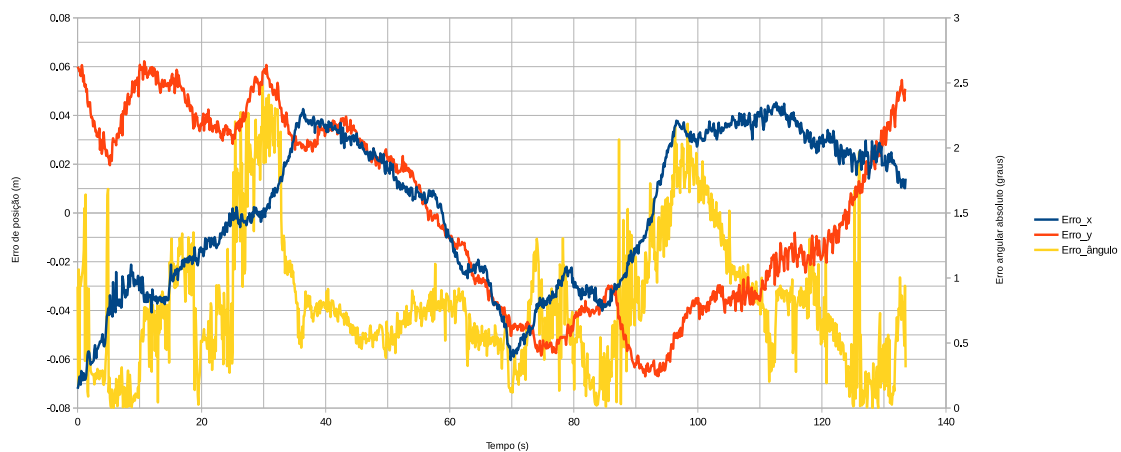
Tabela 5.2: Definição da *Config1_OF* e da *Config2_OF_F1F2*.

	<i>Config1_OF</i>	<i>Config2_OF_F1F2</i>
"OperationMode"	SensorsOdomFusion	SensorsOdomFusion
"MaxErrordistFilterEnabled1"	false	true
"LaserFilterMaxDist1"	-	0.5
"MaxErrordistFilterEnabled2"	false	true
"LaserFilterMaxDist2"	-	0.05
"WeightenMeasureDistEnable"	false	false

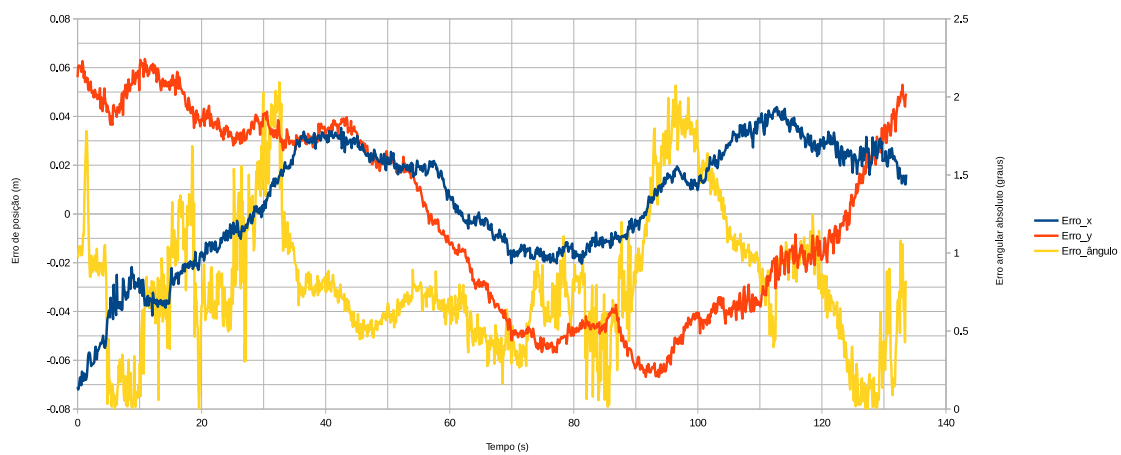
Tabela 5.3: Resultados estatísticos da *Config1_OF* e *Config2_OF_F1F2*, para um ambiente sem outliers.

	<i>Config1_OF</i>	<i>Config2_OF_F1F2</i>
Média do erro de distância (m)	0.0466	0.0445
Média do erro angular absoluto (°)	0.8010	0.8078
Erro máximo de distância (m)	0.0938	0.0940
Erro angular absoluto máximo (°)	2.482	2.094
Variância do erro da distância	$2.545 \cdot 10^{-4}$	$2.157 \cdot 10^{-4}$
Variância do erro angular	0.2546	0.1886

Estas duas configurações são muito similares: ambas utilizam o modelo de fusão sensorial, entre os sensores e a odometria para a estimação da posição. Por outro lado, uma utiliza o filtro de rejeição de outliers (*Config2_OF_F1F2*) e a outra não (*Config1_OF*). Visto que nas condições de teste não foram colocados objetos externos ao mapa, era expectável que com estas duas parametrizações os resultados fossem similares, que foi o que aconteceu. Isto é verificável através da observação das evoluções dos erros ao longo do percurso (Figuras 5.3 e 5.4), com uma ligeira melhoria na parametrização *Config2_OF_F1F2* (Tabela 5.3). Na sequência de imagens da Figura 5.5 a estimativa de localização dada pelo Perfect Match (seta azul) está sempre praticamente sobreposta à posição do *groundtruth* (seta verde), mantendo um reduzido erro ao longo da trajetória do robô.

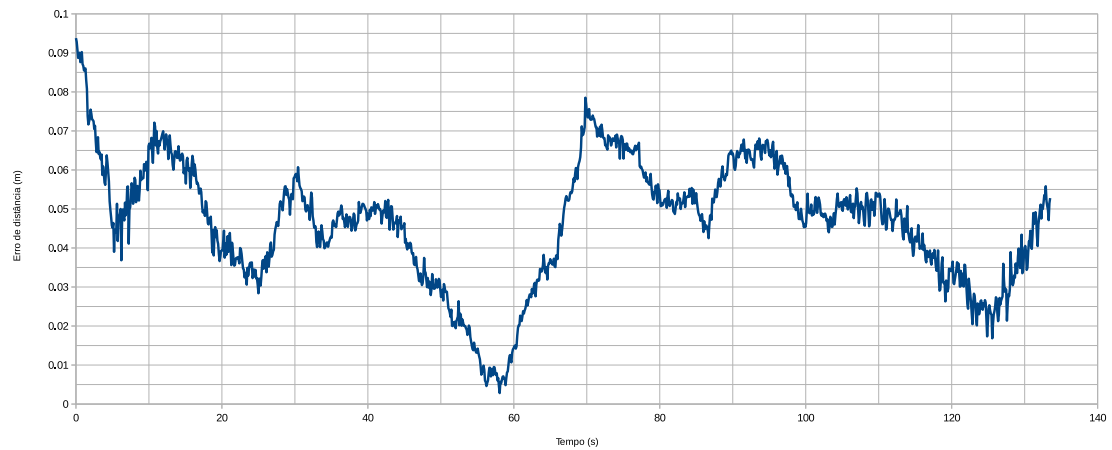


(a)

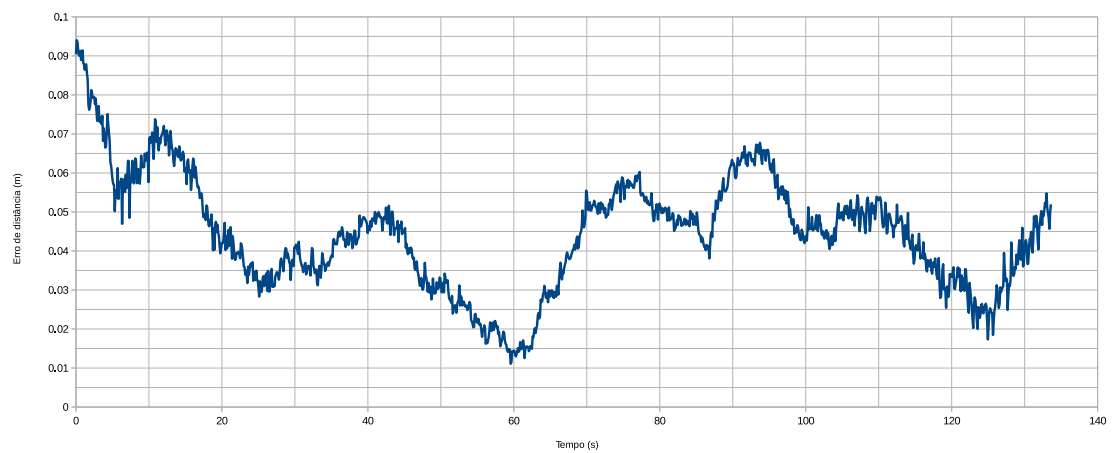


(b)

Figura 5.3: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config1_OF* e a (b) *Config2_OF_F1F2*, para um ambiente sem outliers.



(a)



(b)

Figura 5.4: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config1_OF* e a (b) *Config2_OF_FIF2*, para um ambiente sem *outliers*.

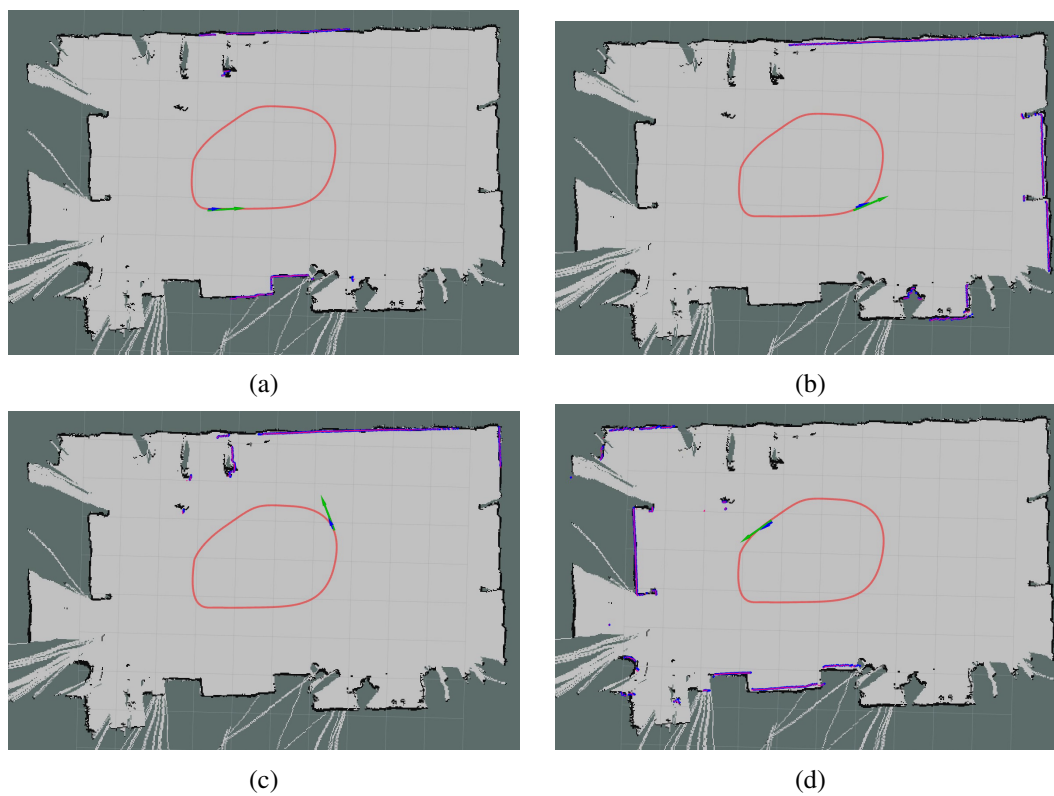


Figura 5.5: Sequência de posições estimadas pelo robô (seta azul) em relação ao *groundtruth* (seta verde), com os pontos do laser representados pelas cores rosa (todos os pontos) e azul (só os pontos utilizados pelo algoritmo).

Configuração 3 e Configuração 4

As Configurações 3 e 4 (*Config3_SO* e *Config4_SO_F1F2*) foram definidas com os parâmetros da Tabela 5.4.

Tabela 5.4: Definição da *Config3_SO* e da *Config4_SO_F1F2*.

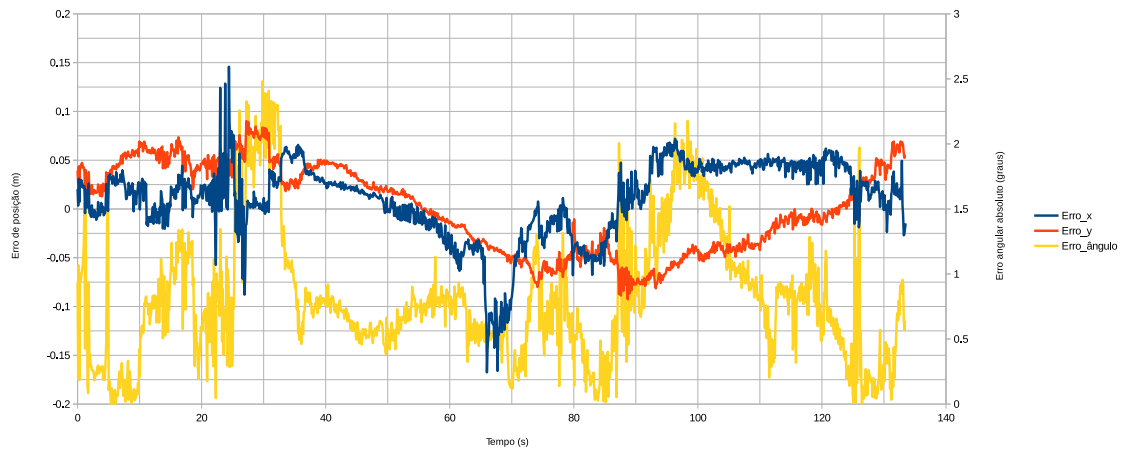
	<i>Config3_SO</i>	<i>Config4_SO_F1F2</i>
"OperationMode"	SensorsOnly	SensorsOnly
"MaxErrordistFilterEnabled1"	false	true
"LaserFilterMaxDist1"	-	0.5
"MaxErrordistFilterEnabled2"	false	true
"LaserFilterMaxDist2"	-	0.05
"WeightenMeasureDistEnable"	false	false

Para estas duas configurações, foi utilizado o modo do algoritmo que apenas utiliza os sensores do robô (laser), não integrando os dados fornecidos pela odometria. Assim, a cada iteração é calculada a posição que é mais ajustada com os dados fornecidos pelo laser naquele período, não utilizando a odometria para pesar a fase de propagação incremental da posição. Por isso, verifica-se uma elevada oscilação na estimativa devolvida pelo Perfect Match com erros absolutos máximos bastante superiores, visível nos gráficos das Figuras 5.6 e 5.7, mesmo num ambiente sem *outliers*. Para além disto, com a utilização deste parâmetro, o sistema fica muito sensível às medidas do laser, o que é crítico para aplicações do robô em ambientes dinâmicos e com os sensores a uma baixa altura, muito susceptíveis a ruído.

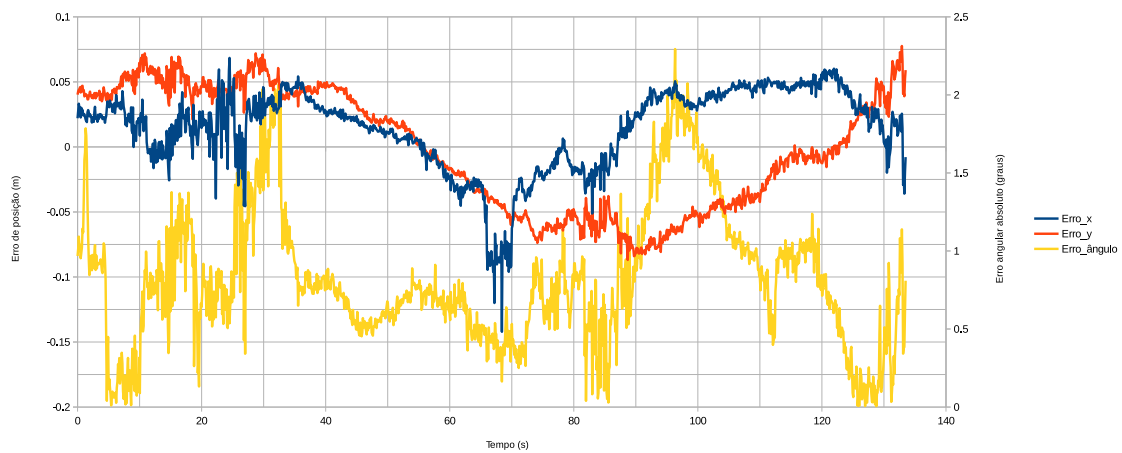
A partir da Tabela 5.5 verifica-se que as duas configurações devolveram valores similares, com uma ligeira melhoria na configuração que utiliza o filtro de rejeição de valores desajustados do laser (*Config4_SO_F1F2*). Isto é explicado porque, mesmo tendo-se utilizado um ambiente controlado e sem obstáculos, existem sempre alguns pontos do laser que não se ajustam ao mapa como seria de esperar e, ao serem descartados, tornam o algoritmo mais imune aos erros de medição.

Tabela 5.5: Resultados estatísticos da *Config3_SO* e *Config4_SO_F1F2*, na ausência de *outliers*.

	<i>Config3_SO</i>	<i>Config4_SO_F1F2</i>
Média do erro de distância (m)	0.0558	0.0544
Média do erro angular absoluto (°)	0.8119	0.8177
Erro máximo de distância (m)	0.1715	0.1498
Erro angular absoluto máximo (°)	2.482	2.294
Variância do erro da distância	$5.050 \cdot 10^{-4}$	$3.366 \cdot 10^{-4}$
Variância do erro angular	0.2542	0.1924

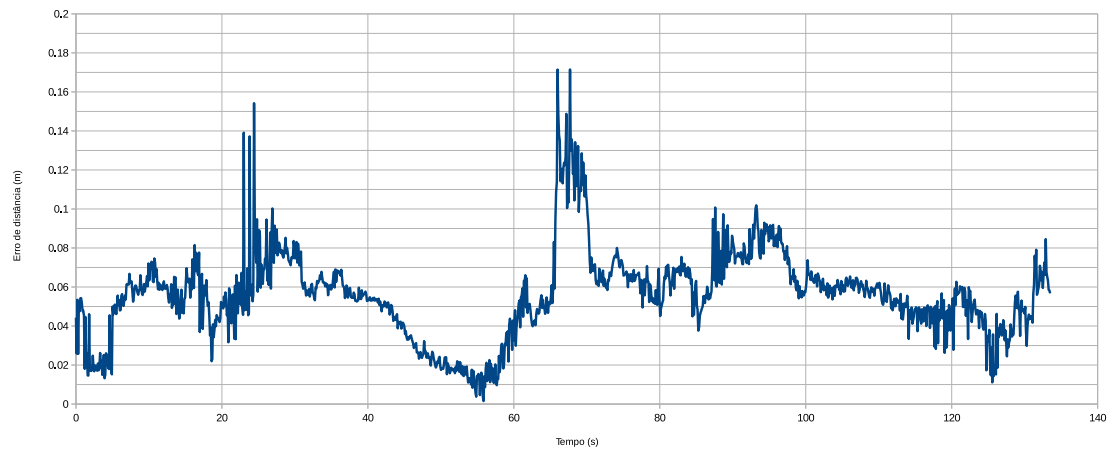


(a)

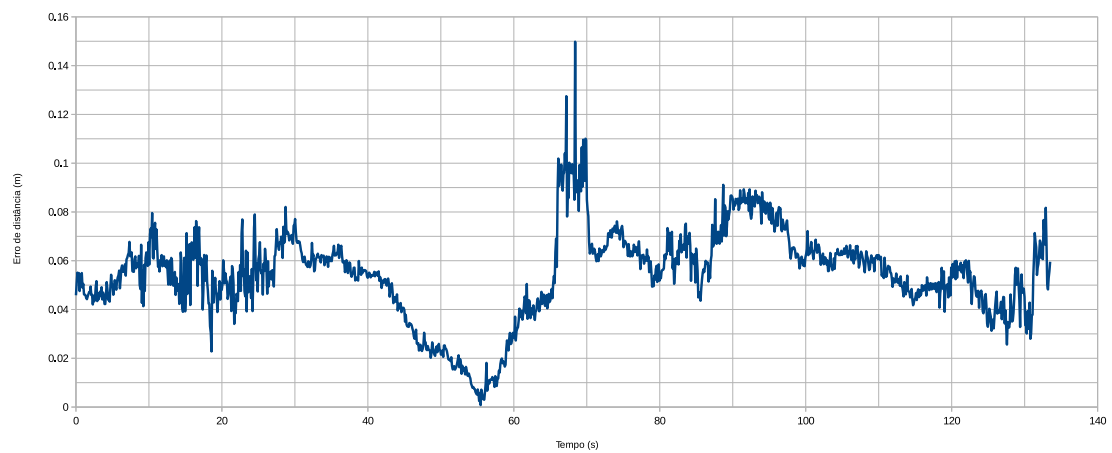


(b)

Figura 5.6: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config3_SO* e a (b) *Config4_SO_FIF2*, para o mesmo ambiente sem outliers.



(a)



(b)

Figura 5.7: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config3_SO* e a (b) *Config4_SO_FIF2*, para o mesmo ambiente sem *outliers*.

Configuração 5

A Configuração 5 (*Config5_OF_F1F2WD*) terá os parâmetros da Tabela 5.6.

Tabela 5.6: Definição da *Config5_OF_F1F2WD*.

"OperationMode"	SensorsOdomFusion
"MaxErrordistFilterEnabled1"	true
"LaserFilterMaxDist1"	0.5
"MaxErrordistFilterEnabled2"	true
"LaserFilterMaxDist2"	0.05
"WeightenMeasureDistEnable"	true

A utilização do parâmetro *WeightenMeasureDistEnable* para este exemplo, onde não existem *outliers*, não revelou ter impacto na melhoria do algoritmo em termos de precisão das suas estimativas (Tabela 5.7), tendo mantido uma muito similar evolução dos erros ao longo do tempo (Figuras 5.8 e 5.9) comparativamente com os gráficos e dados estatísticos das *Config1_OF* e *Config2_OF_F1F2*.

Tabela 5.7: Resultados estatísticos da *Config5_OF_F1F2WD*, na ausência de *outliers*.

Média do erro de distância (m)	0.0458 m
Média do erro angular absoluto (°)	0.8310 °
Erro máximo de distância (m)	0.0942 m
Erro angular absoluto máximo (°)	1.963 °
Variância do erro da distância	$2.451 \cdot 10^{-4}$
Variância do erro angular	0.1686

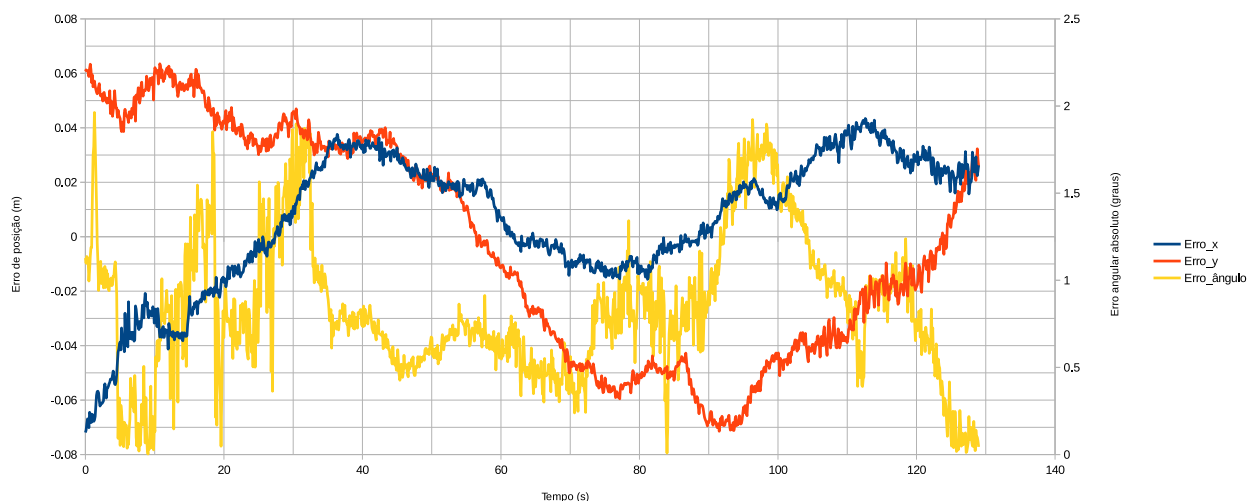


Figura 5.8: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a *Config5_OF_F1F2WD*, nas mesmas condições que as configurações anteriores.

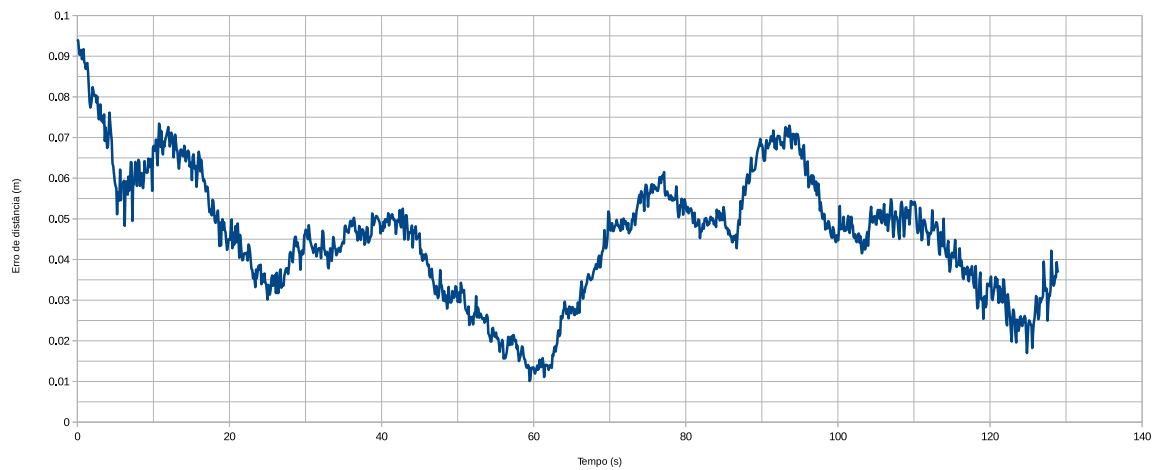
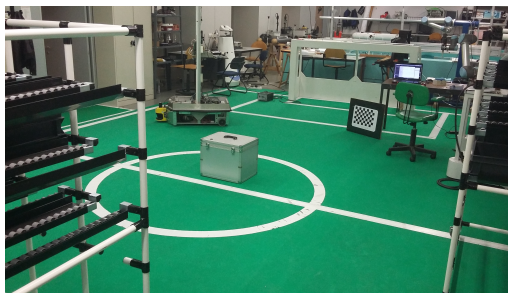


Figura 5.9: Evolução do erro em distância, ao longo do percurso, utilizando a *Config5_OF_F1F2WD*, nas mesmas condições que as configurações anteriores.

5.1.3.2 Ambiente com alguns outliers

Para esta secção, irão manter-se as Configurações já apresentadas anteriormente, testadas agora para um ambiente com alguma quantidade de objetos estranhos ao mapa, como se pode visualizar na Figura 5.10. À volta do percurso do robô colocaram-se duas estruturas móveis com prateleiras (visíveis do lado esquerdo e no canto inferior direito da Figura 5.10a, assim como no canto superior esquerdo da Figura 5.10b), uma caixa no centro da trajetória do robô e uma tábua (com padrão de xadrez, visível na Figura 5.10a) junto a uma das mesas.



(a)



(b)

Figura 5.10: Ambiente onde se realizaram os testes, tendo-se colocado alguns obstáculos junto às paredes e no centro da trajetória do robô.

Config1_OF

Utilizando a *Config1_OF*, agora para um ambiente com alguns objetos não pertencentes ao mapa obtiveram-se os resultados representados pelas Figuras 5.11, 5.12 e na Tabela 5.8. Como é perceptível pelos dados apresentados, o robô perdeu-se momentaneamente quando começou a integrar os pontos que correspondiam à detecção dos obstáculos do mapa, resultando num *matching* errado pelo algoritmo de otimização e consequentemente devolvendo uma posição errada. Após algum tempo, quando o robô se deslocou para uma zona em que já não havia grande número de *outliers*, acabou por voltar a convergir para a posição correta. Isto pode ser visualizado na Figura 5.13, onde está representada a posição estimada pelo Perfect Match (seta azul), a posição do *groundtruth* (seta verde) e os pontos do laser (a rosa são todos os pontos devolvidos pelo laser e a azul estão apenas os pontos que o algoritmo utiliza, após a rejeição de *outliers*). Neste exemplo em concreto, é possível verificar que o algoritmo utiliza todos os pontos do laser (a rejeição de *outliers* está desativada nesta configuração), levando à sua momentânea perda de localização.

Tabela 5.8: Resultados estatísticos da *Config1_OF*, com alguns obstáculos no mapa.

Média do erro de distância (m)	0.4957
Média do erro angular absoluto (°)	4.718
Erro máximo de distância (m)	2.517
Erro angular absoluto máximo (°)	34.96
Variância do erro da distância	0.5072
Variância do erro angular	53.82

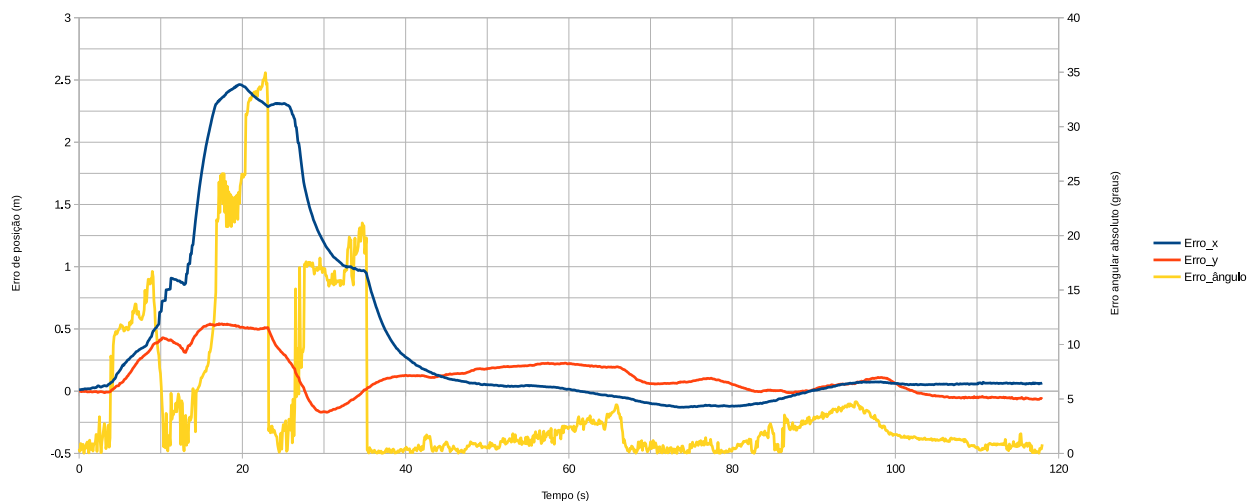


Figura 5.11: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a *Config1_OF*, para um ambiente com alguns *outliers*.

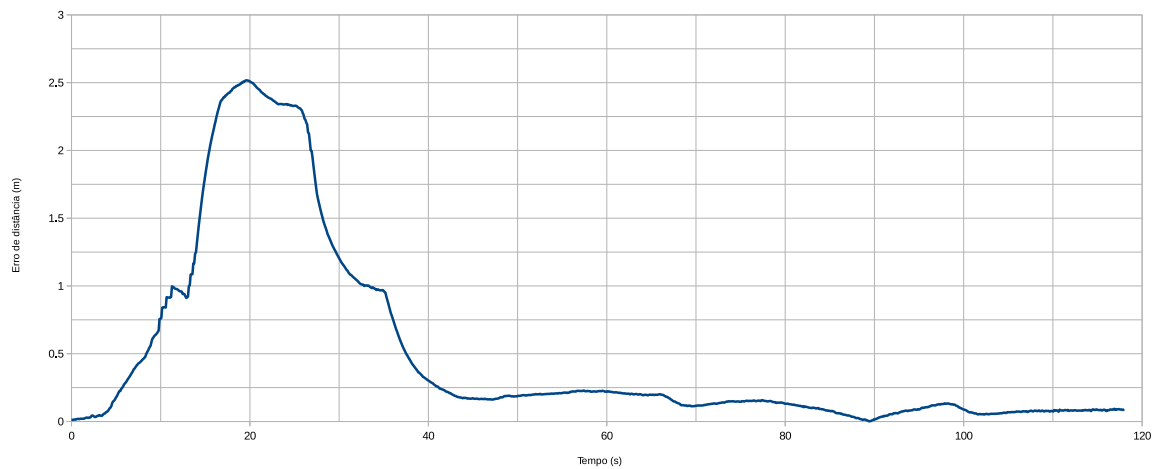


Figura 5.12: Evolução do erro de distância, ao longo do percurso, utilizando a *Config1_OF* para um ambiente com alguns *outliers*.

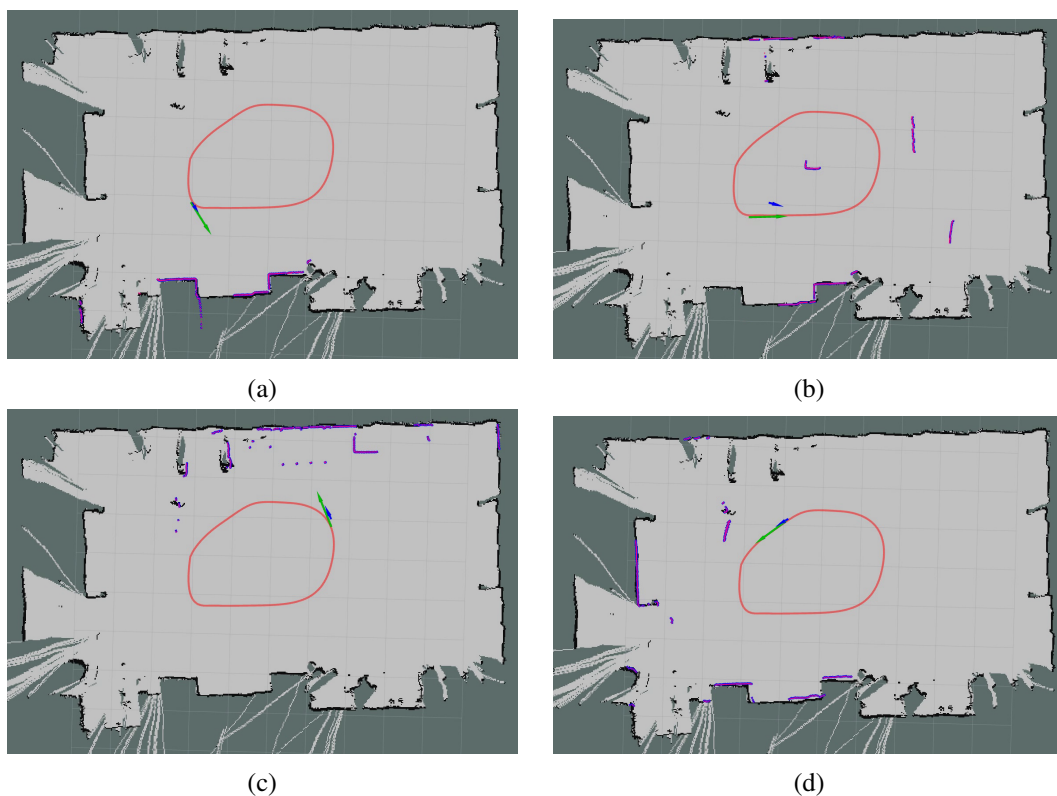


Figura 5.13: (a) Robô na posição inicial, bem localizado. (b) A integração de pontos do laser correspondentes à detecção de obstáculos externos ao mapa resulta num aumento momentâneo do erro de localização. (c) e (d) Numa zona com menos *outliers* o robô volta a conseguir recuperar a posição correta.

Config2_OF_F1F2

Com a utilização de uma configuração com o filtro de rejeição de *outliers* ativo, aumenta a robustez do algoritmo a objetos ou pessoas inesperadas no ambiente de ação do robô. Na Figura 5.16, é possível verificar que o Perfect Match descarta os pontos referentes à detecção de *outliers* (a cor de rosa), conseguindo manter uma boa localização da posição do robô durante todo o percurso, notório pela sobreposição entre a estimativa do Perfect Match (seta azul) e a posição do *ground-truth* (seta verde). Ao observar-se as Figuras 5.14, 5.15 e os dados fornecidos pela Tabela 5.9, é possível notar uma elevada precisão das estimativas de localização do Perfect Match, obtendo um erro médio inferior a 3 cm e um erro máximo pouco superior a 5 cm.

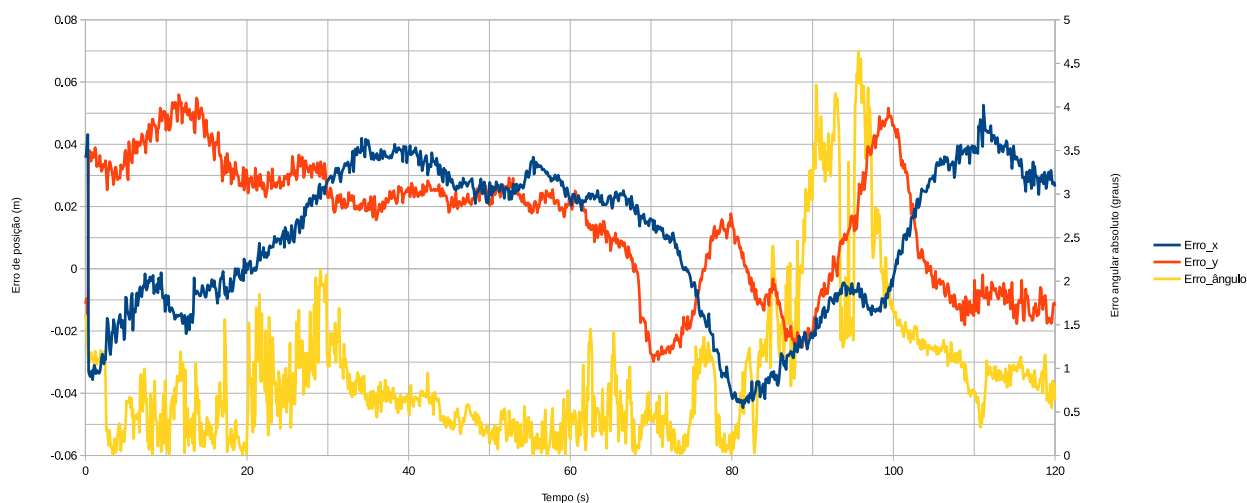


Figura 5.14: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a *Config2_OF_F1F2*, para um ambiente com alguns *outliers*.

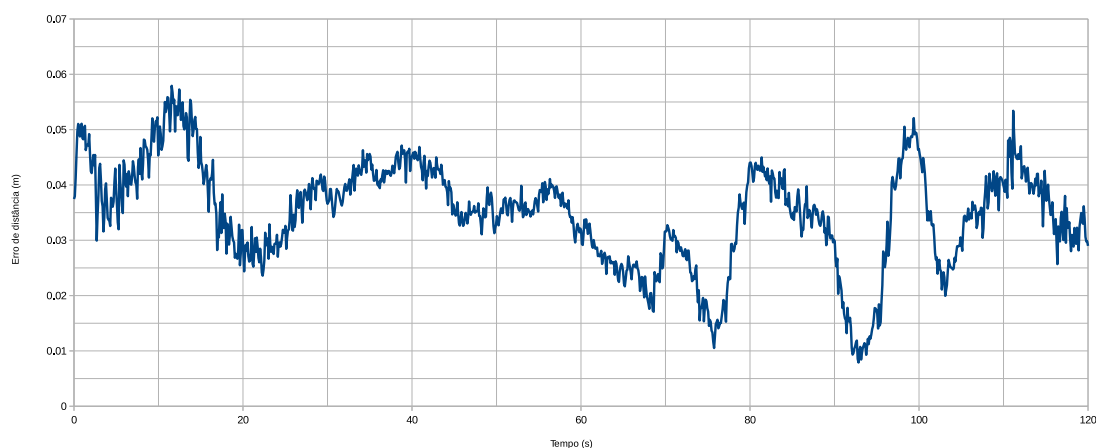


Figura 5.15: Evolução do erro de distância, ao longo do percurso, utilizando a *Config2_OF_F1F2* para um ambiente com alguns *outliers*.

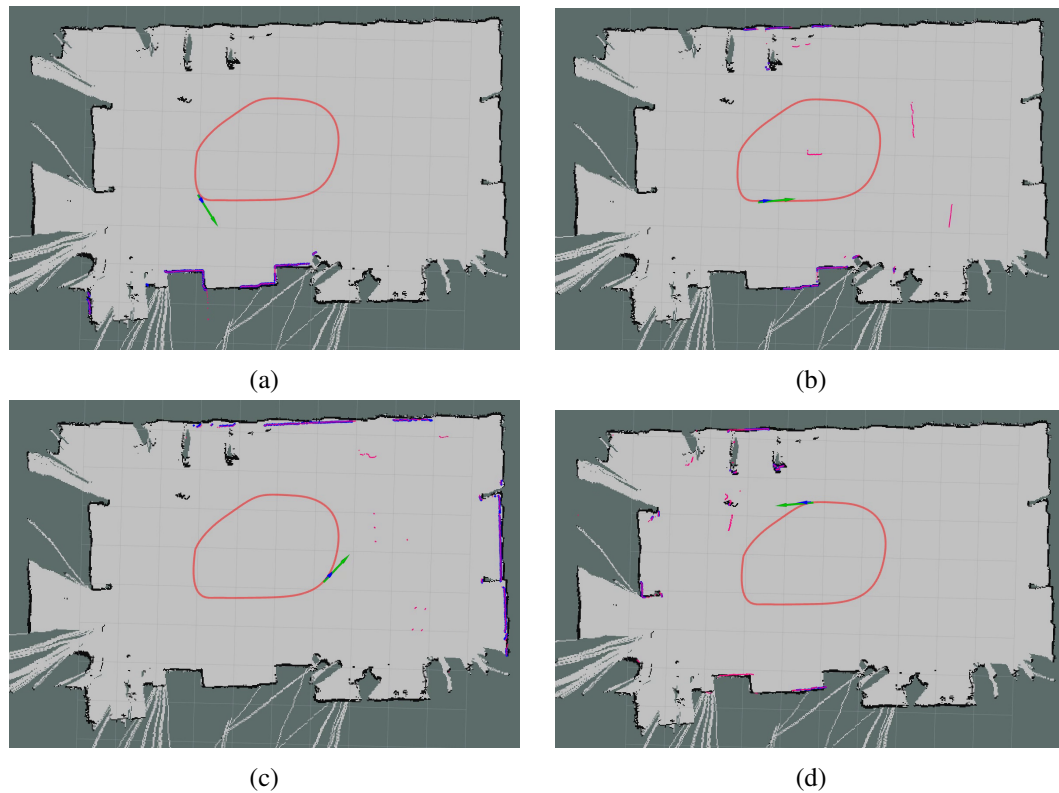


Figura 5.16: (a) Robô na posição inicial, bem localizado. (b) Os pontos a cor de rosa (*outliers*) são descartados pelo algoritmo. (c) e (d) O Perfect Match consegue manter uma boa estimativa da localização do robô (a seta azul corresponde à estimativa da localização e a seta verde ao *groundtruth*).

Tabela 5.9: Resultados estatísticos da *Config2_OF_F1F2*, com alguns obstáculos no mapa.

Média do erro de distância (m)	0.0291
Média do erro angular absoluto (°)	0.9107
Erro máximo de distância (m)	0.0549
Erro angular absoluto máximo (°)	4.639
Variância do erro da distância	$1.346 \cdot 10^{-4}$
Variância do erro angular	0.6941

Configuração 6

Para ilustrar o efeito prático dos dois filtros de *outliers*, preparou-se uma nova configuração (*Config6_OF_F2*), descrita na Tabela 5.10. Aqui, utiliza-se apenas o filtro mais exigente do Perfect Match, descartando todos os pontos que estão a uma distância superior a 0.05 m da posição esperada, sem a utilização do primeiro filtro que rejeita *outliers* mais desajustados.

Tabela 5.10: Definição da *Config6_OF_F2*.

"OperationMode"	SensorsOdomFusion
"MaxErrordistFilterEnabled1"	false
"LaserFilterMaxDist1"	-
"MaxErrordistFilterEnabled2"	true
"LaserFilterMaxDist2"	0.05
"WeightenMeasureDistEnable"	false

Nas Figuras 5.17 (c) e (d) é possível verificar que alguns pontos do laser correspondentes a secções das paredes não foram integrados pelo algoritmo (pontos que não estão a azul), por serem considerados a uma distância superior ao limiar especificado (0.05 m), causado por um desajuste temporário da estimativa de localização dada pelo Perfect Match, levando à rejeição desses pontos. Este tipo de comportamento pode ser crítico em determinados ambientes de operação, podendo chegar ao limite de descartar grande parte dos pontos e levar à perda do robô.

Através da análise dos gráficos das Figuras 5.18, 5.19 e dos resultados estatísticos da Tabela 5.11, verifica-se que a estimativa calculada pelo Perfect Match resultou num aumento temporário do erro de localização, devido à integração de pontos correspondentes a obstáculos presentes no ambiente e à não inclusão de alguns pontos corretos (*e.g.*, da detecção de paredes), em algumas zonas do percurso do robô.

Tabela 5.11: Resultados estatísticos da *Config6_OF_F2*, com alguns obstáculos no mapa.

Média do erro de distância (m)	0.1913
Média do erro angular absoluto (°)	3.406
Erro máximo de distância (m)	0.9419
Erro angular absoluto máximo (°)	25.74
Variância do erro da distância	0.0812
Variância do erro angular	32.23

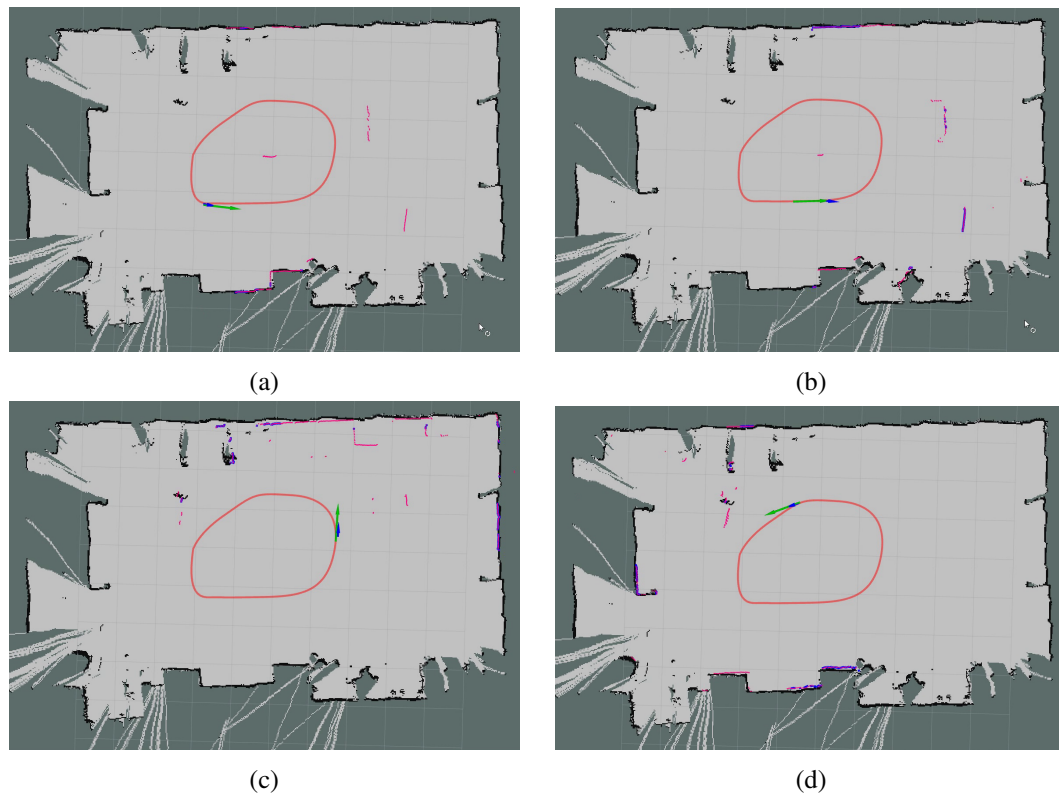


Figura 5.17: (a) Robô na posição inicial, bem localizado. (b) O algoritmo integra alguns pontos referentes aos obstáculos que estão no mapa e aumenta o erro de localização. (c) Numa zona onde há menos *outliers* e mais dados referentes ao mapa, o robô começa a convergir para a posição correta. (d) Robô bem localizado.

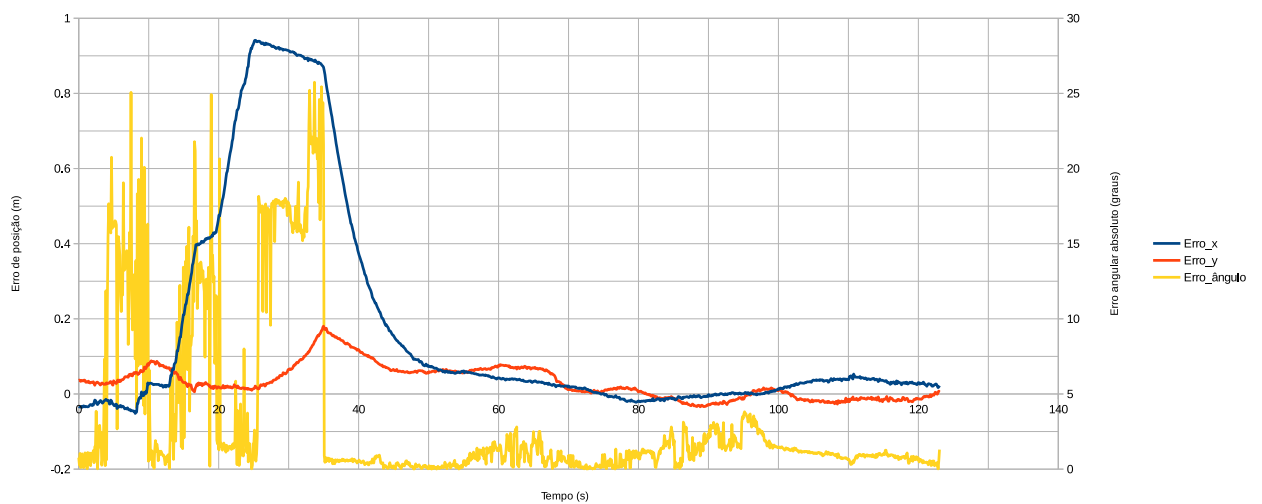


Figura 5.18: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a *Config6_OF_F2*, para um ambiente com alguns *outliers*.

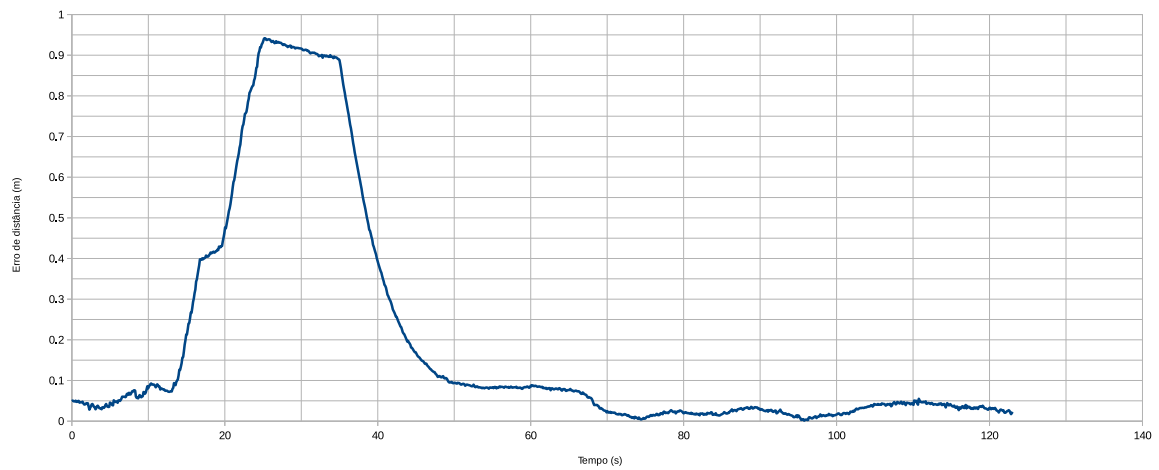


Figura 5.19: Evolução do erro de distância, ao longo do percurso, utilizando a *Config6_OF_F2* para um ambiente com alguns *outliers*.

Configuração 7

Para o caso inverso ao da *Config6_OF_F2*, desligou-se o filtro mais exigente para a rejeição de *outliers* e ligou-se o filtro menos exigente (com o valor 0.5 m) para esta nova configuração (*Config7_OF_F1*), visível na Tabela 5.12.

Tabela 5.12: Definição da *Config7_OF_F1*.

"OperationMode"	SensorsOdomFusion
"MaxErrorDistFilterEnabled1"	true
"LaserFilterMaxDist1"	0.5
"MaxErrorDistFilterEnabled2"	false
"LaserFilterMaxDist2"	-
"WeightenMeasureDistEnable"	false

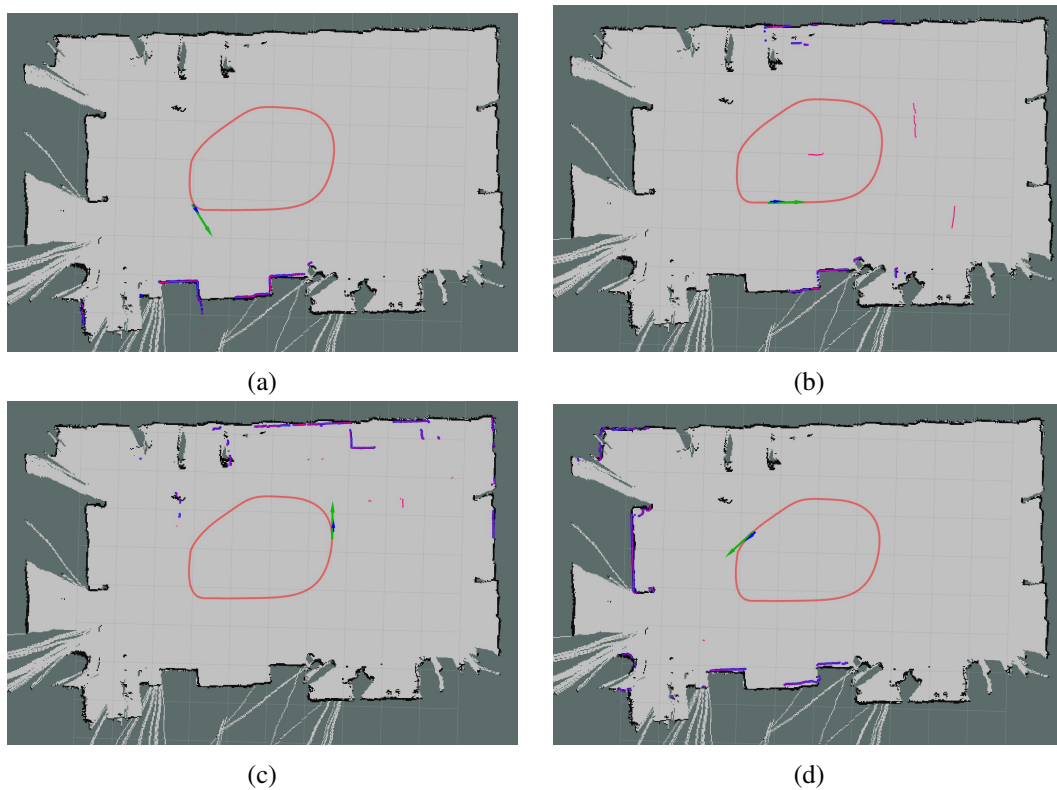


Figura 5.20: (a) Robô na posição inicial (seta azul), bem localizado. (b) O algoritmo descarta o conjunto de pontos a cor de rosa (*outliers* a mais de 0.5 m dos pontos esperados). (c) Os obstáculos próximos da parede (a menos de 0.5 m) são integrados no algoritmo (pontos azuis), levando a um aumento do erro de localização. (d) Na zona sem *outliers* o robô volta a recuperar a posição correta.

Tendencialmente, para este tipo de configuração, apesar do algoritmo não atingir valores de erro tão elevados como para a *Config6_OF_F2*, acaba por integrar conjuntos de pontos do laser referentes a obstáculos localizados a uma distância inferior à especificada (para este caso, 0.5 m), o que leva a um aumento do erro nessas zonas, devido a um ajuste errado por parte do algoritmo

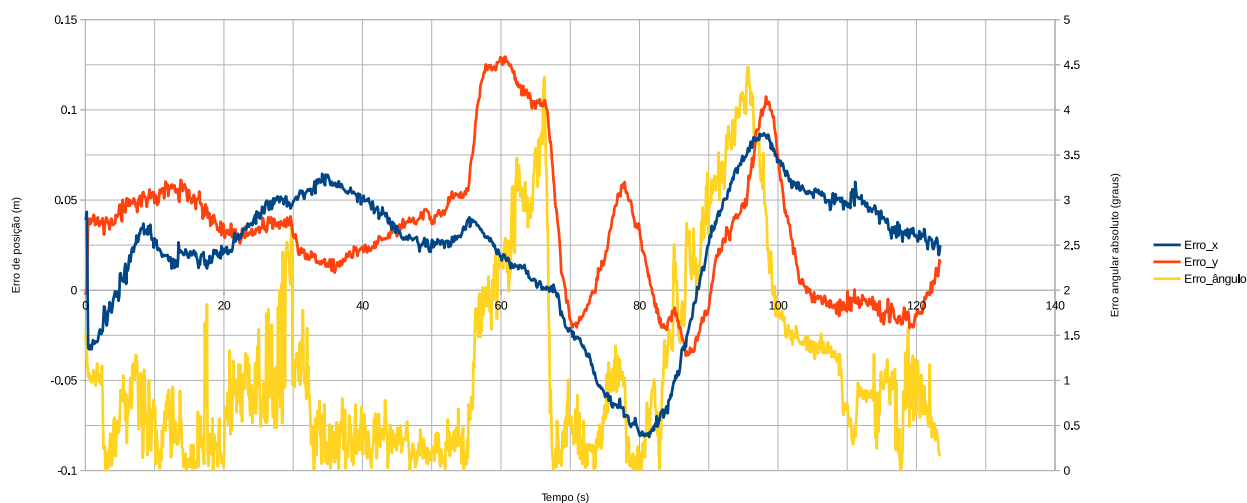


Figura 5.21: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a *Config7_OF_F1*, para um ambiente com alguns *outliers*.

ao considerar esses objetos como parte do mapa. Este fenómeno é claramente verificado na Figura 5.20 (c), onde a deteção de uma caixa localizada junto à parede é integrada no algoritmo (indicado pelos pontos azuis do laser), levando a um temporário desajuste na estimativa de localização, aumentando o erro (Figuras 5.21 e 5.22).

Assim, através de experiências reais, fica demonstrada a importância da utilização dos dois filtros de rejeição de *outliers*, tendo estes efeitos complementares e devendo ser integrados conjuntamente para um melhor desempenho do algoritmo. Isto porque os filtros são aplicados em iterações consecutivas do algoritmo: numa primeira fase com um limiar menos exigente no erro de distância dos pontos em relação ao obstáculo mais próximo (para estes exemplos 0.5 m) e, depois de um ajuste da estimativa da posição, através do algoritmo RPROP, volta-se a descartar um conjunto de pontos com base num limiar menos exigente (0.05 m, nestes casos) antes de voltar a executar de novo o algoritmo de otimização.

Tabela 5.13: Resultados estatísticos da *Config7_OF_F1*, com alguns obstáculos no mapa.

Média do erro de distância (m)	0.0609
Média do erro angular absoluto (°)	1.129
Erro máximo de distância (m)	0.1376
Erro angular absoluto máximo (°)	4.477
Variância do erro da distância	$6.539 \cdot 10^{-4}$
Variância do erro angular	1.096

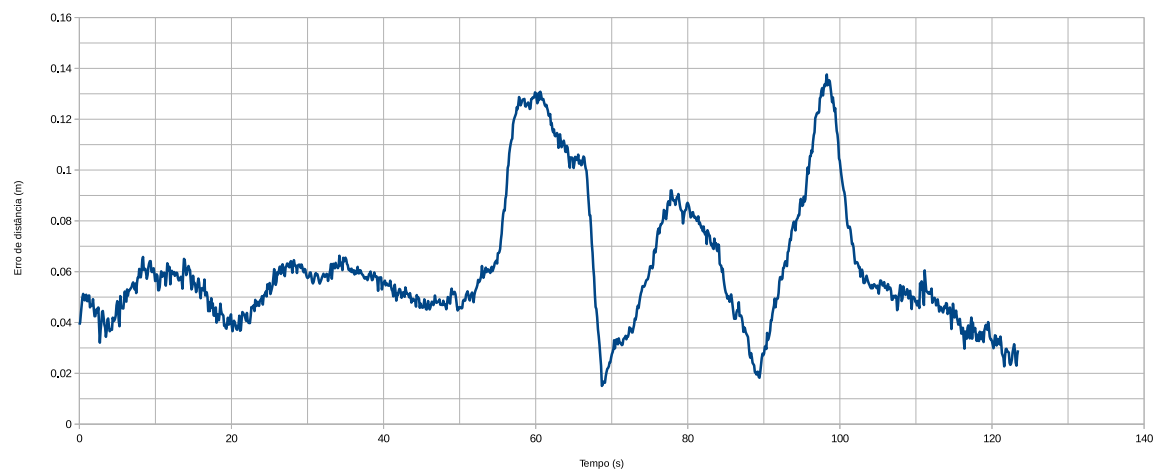


Figura 5.22: Evolução do erro de distância, ao longo do percurso, utilizando a *Config7_OF_F1* para um ambiente com alguns *outliers*.

5.1.3.3 Ambiente com grande quantidade *outliers*

Depois de terem sido analisados os resultados das estimativas de localização obtidas através do algoritmo Perfect Match com diferentes configurações e para dois tipos de ambientes: sem *outliers* e com alguns obstáculos, apresentar-se-á os resultados obtidos para um ambiente com grande quantidade de objetos colocados ao longo do percurso do robô, com vista a perturbar a sua localização.

Na Figura 5.23 podemos observar o local utilizado e as condições dos testes do robô. Para além das estruturas móveis com prateleiras já colocadas para os testes com menos número de *outliers*, utilizou-se um conjunto de placas e tábuas (visíveis no lado esquerdo das Figuras 5.23a e 5.23b), colocou-se uma caixa junto à parede e manteve-se um robô a tapar uma parte da baliza do campo do RoboCup.



Figura 5.23: Ambiente onde se realizaram os testes, tendo-se aumentado a quantidade de objetos externos ao mapa, com vista a perturbar a percepção do ambiente por parte do robô.

É de notar que se prevê que o robô Clever opere em situações com alguma quantidade de pessoas, mobília e obstáculos à sua volta, circunstâncias similares às apresentadas nesta secção de testes.

Com base nos resultados das secções anteriores, aqui será feita uma análise apenas das configurações que apresentaram valores consistentes e que conseguiram manter uma boa estimativa da posição do robô, excluindo as configurações que já demonstraram falhar em condições menos exigentes.

Assim, utilizar-se-ão a [Config2_OF_F1F2](#) e a [Config5_OF_F1F2WD](#), ambas com os dois filtros de rejeição de *outliers* ativos e fusão da odometria com os dados do laser. A única diferença é a utilização do parâmetro *WeightenMeasureDistEnable* na [Config5_OF_F1F2WD](#).

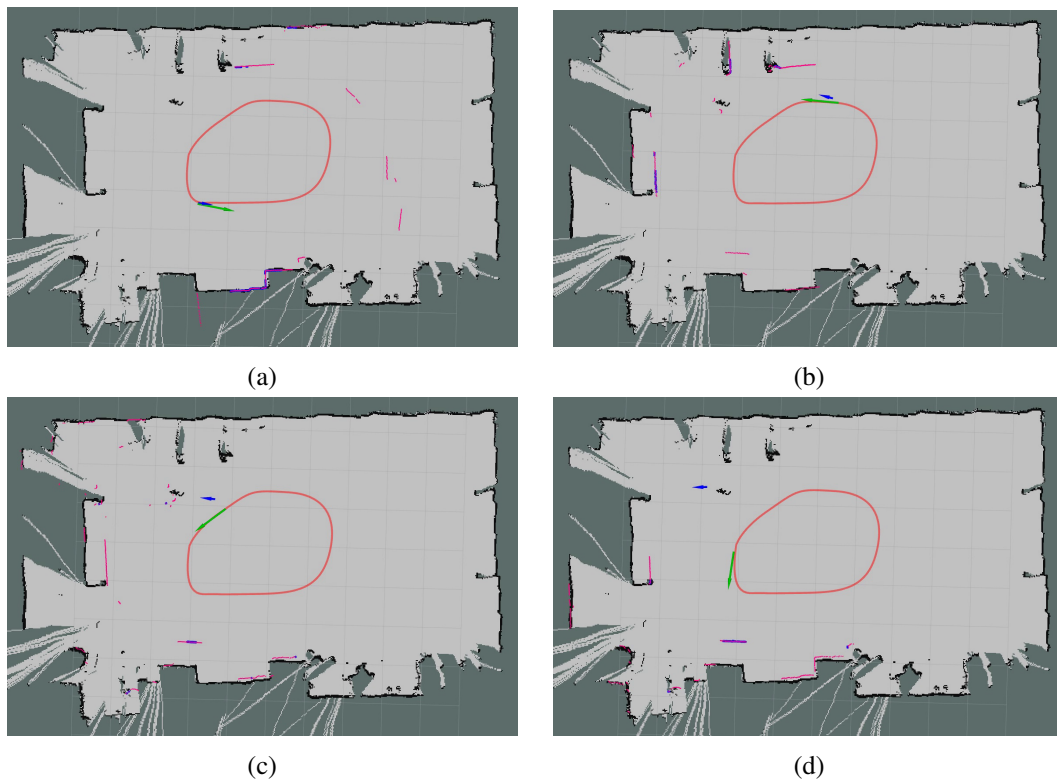
Config2_OF_F1F2 e Config5_OF_F1F2WD

Figura 5.24: Resultados para a *Config2_OF_F1F2*: (a) Robô na posição inicial (seta azul), bem localizado em relação ao *groundtruth*. (b) O algoritmo, ao utilizar alguns pontos do laser referentes a *outliers* começa a divergir da posição correta. (c) e (d) O robô acaba por se perder.

Tabela 5.14: Resultados estatísticos das *Config2_OF_F1F2* e *Config5_OF_F1F2WD*, com grande quantidade de obstáculos no mapa.

	<i>Config2_OF_F1F2</i>	<i>Config5_OF_F1F2WD</i>
Média do erro de distância (m)	0.3483	0.0896
Média do erro angular absoluto ($^{\circ}$)	13.53	2.188
Erro máximo de distância (m)	2.488	0.1776
Erro angular absoluto máximo ($^{\circ}$)	80.69	15.94
Variância do erro da distância	0.3625	$2.925 \cdot 10^{-3}$
Variância do erro angular	506.6	5.502

A partir da análise dos resultados apresentados nas Figuras 5.26 e 5.27, verifica-se que, para esta experiência em concreto, a utilização de uma atribuição de pesos diferentes para os pontos do laser de acordo com as suas distâncias ao robô, provou fazer uma enorme diferença no desempenho e robustez do algoritmo de localização. Como foi verificado em secções anteriores, a utilização desta opção não melhora muito a precisão na estimativa de localização do Perfect Match, mas pode fazer a diferença em momentos em que o número de pontos utilizados pelo algoritmo é muito reduzido. É de notar que, neste exemplo, a quantidade de *outliers* é muito elevada, sendo

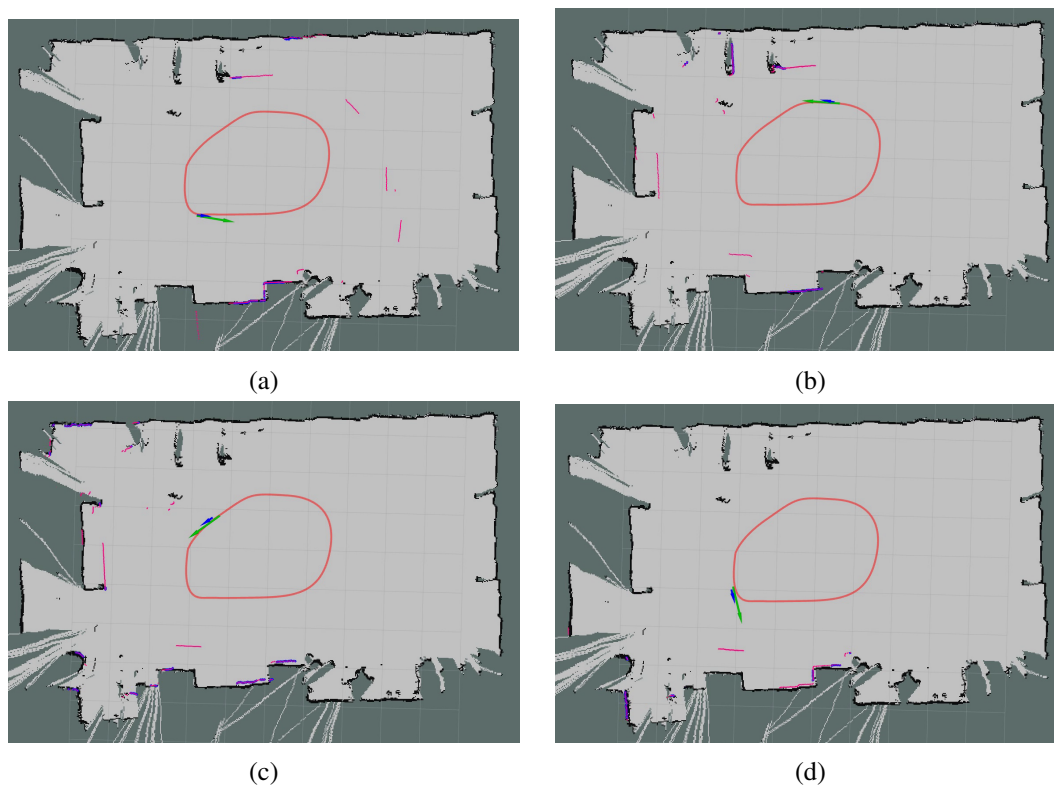
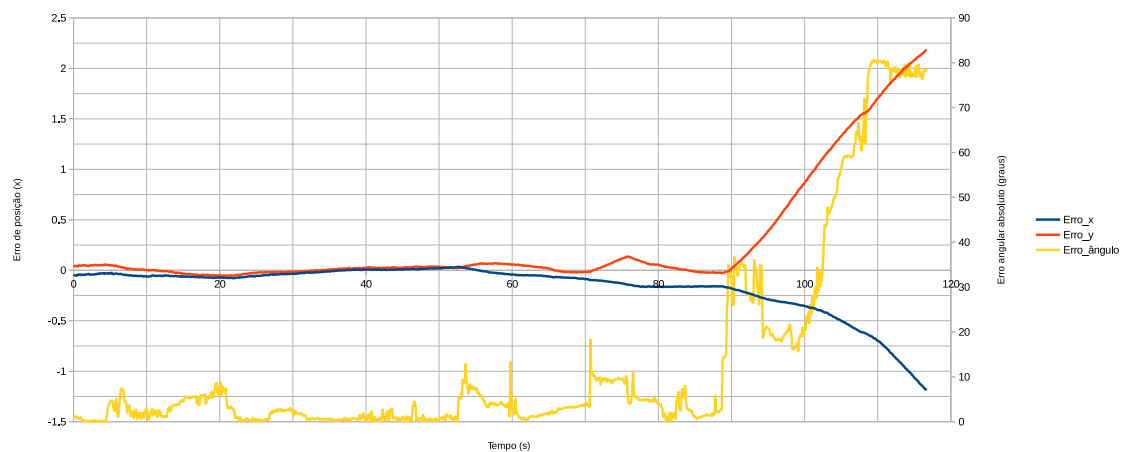
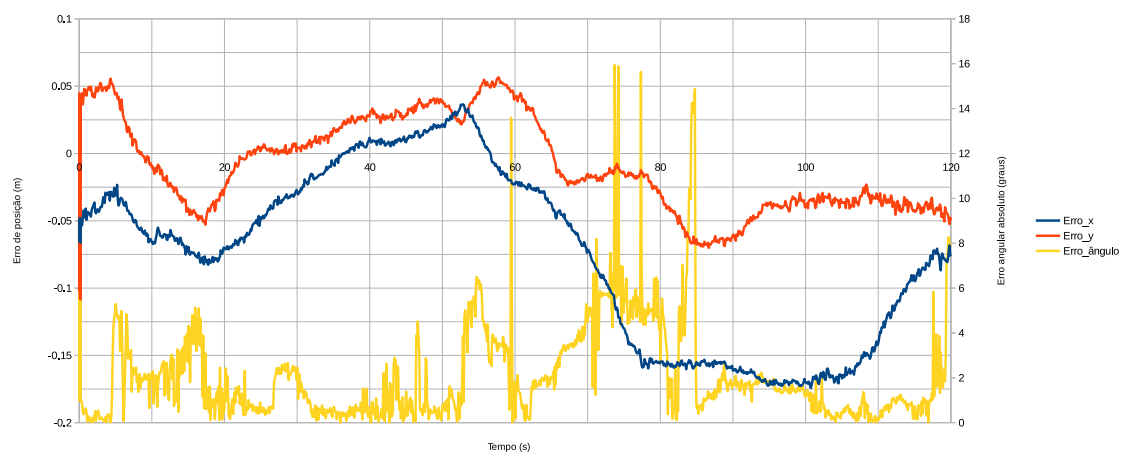


Figura 5.25: Resultados para a *Config5_OF_F1F2WD*: (a) Robô na posição inicial (seta azul), bem localizado em relação ao *groundtruth*. (b), (c) e (d) Com esta configuração o Perfect Match consegue manter uma boa estimativa de localização ao longo do tempo, ganhando algum erro a meio do percurso, devido a uma grande quantidade de *outliers*.

que o algoritmo, em alguns momentos, utiliza muito poucos pontos do laser (após as rejeições dos pontos desajustados) no processo de *matching*, podendo levar a um aumento do erro ou até à sua perda, que foi o que aconteceu no caso da *Config2_OF_F1F2* (Figuras 5.26a e 5.27a). Numa situação normal de operação do robô, utilizar-se-ia o parâmetro que limita a percentagem mínima de pontos do laser (*MinPercentValideSensorData*) a ser utilizada pelo algoritmo, fazendo o robô parar no caso do número de pontos ficar abaixo do limiar especificado.

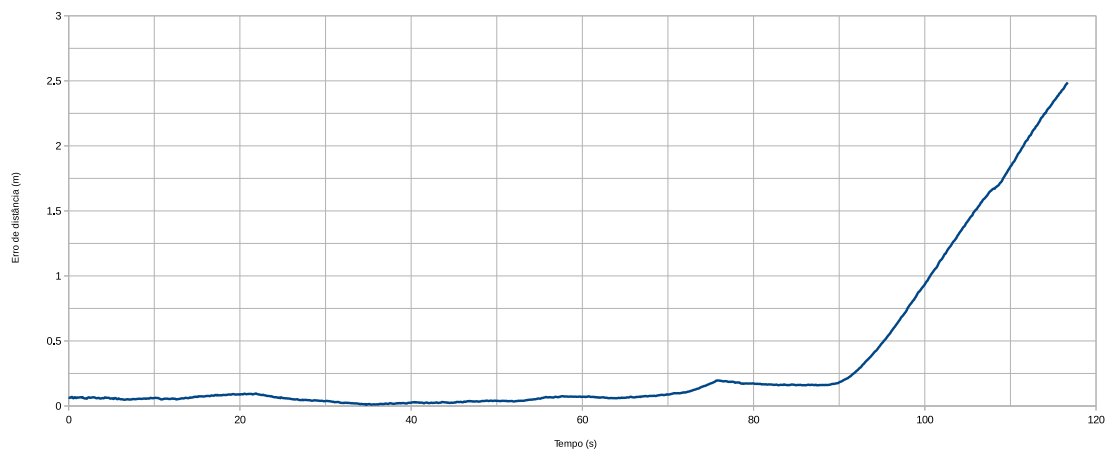


(a)

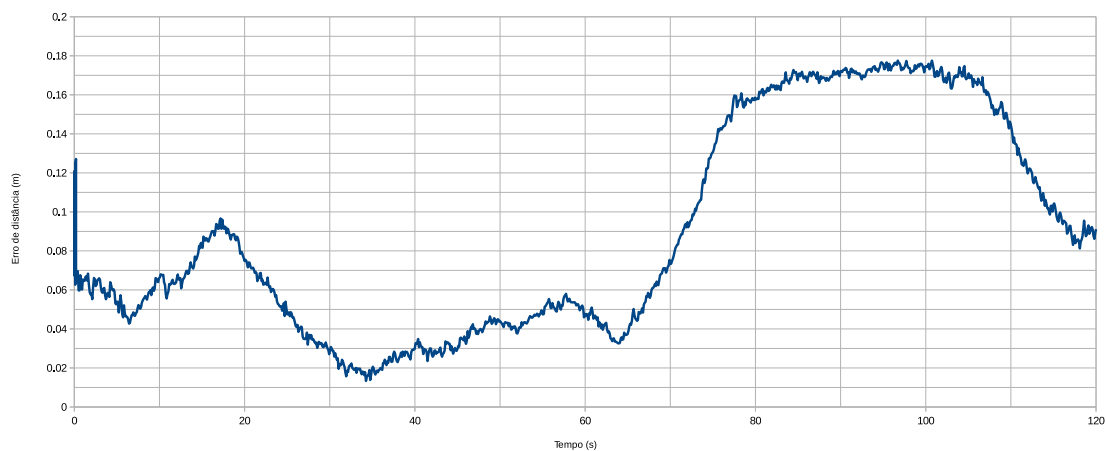


(b)

Figura 5.26: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config2_OF_F1F2* e a (b) *Config5_OF_F1F2WD*, para um ambiente com muitos obstáculos.



(a)



(b)

Figura 5.27: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config2_OF_F1F2* e a (b) *Config5_OF_F1F2WD*, para um ambiente com grande quantidade de *outliers*.

5.1.3.4 Erros forçados de odometria

Por fim, irá testar-se a robustez do algoritmo Perfect Match, utilizando as mesmas duas configurações da secção anterior (*Config2_OF_F1F2* e *Config5_OF_F1F2WD*) em condições de erros de odometria. Este caso em estudo tenta refletir algumas situações onde as condições do piso, no ambiente onde o robô vai operar, se tornam adversas. Por exemplo, em ambientes em que possa haver líquidos no chão (*e.g.*, água, óleo), pequenos objetos ou até zonas com piso irregular, os dados da odometria serão afetados por estas condições e devolverão dados que não traduzem a realidade.

Assim, é importante testar a robustez dos algoritmos de localização para estes casos. Como simulação de algumas das situações descritas anteriormente, levantou-se o robô durante 2-3 segundos em dois momentos do percurso, para que os *encoders* das rodas dessem indicação de movimento do robô quando na realidade este não saiu do sítio. O efeito prático da existência de líquidos no chão é muito similar à situação simulada, em que as rodas derrapam e o robô não se desloca tanto como as informações fornecidas pelos *encoders* indicam.

Config2_OF_F1F2 e *Config5_OF_F1F2WD*

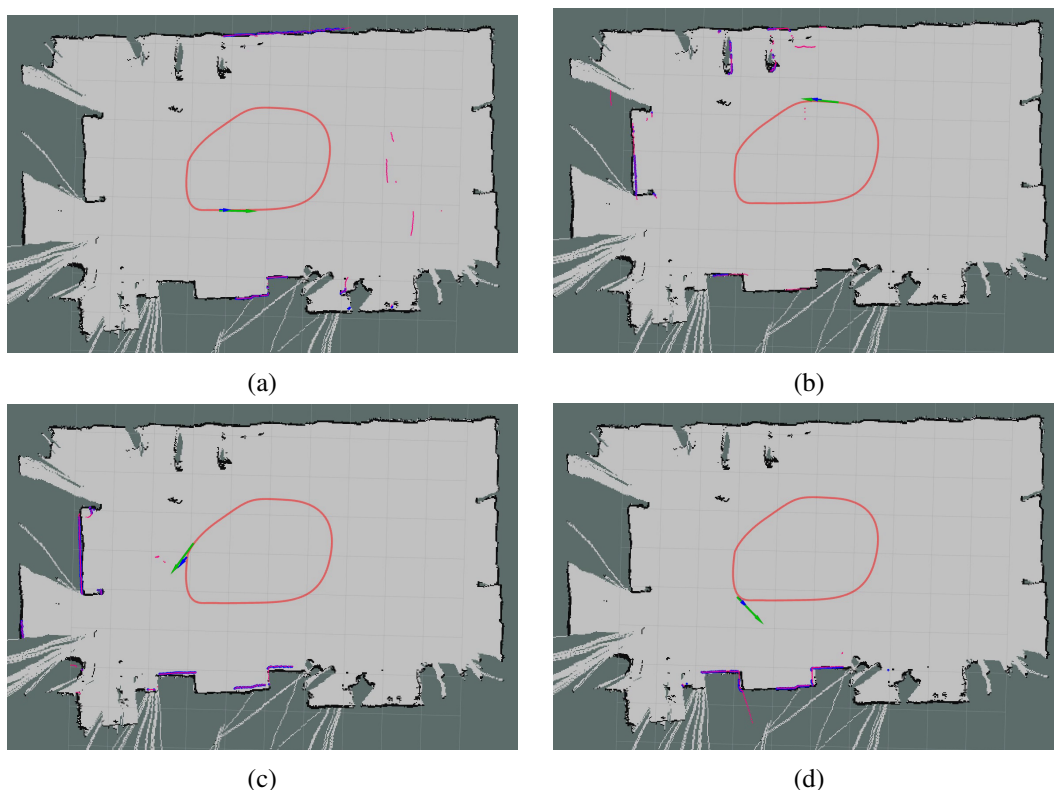
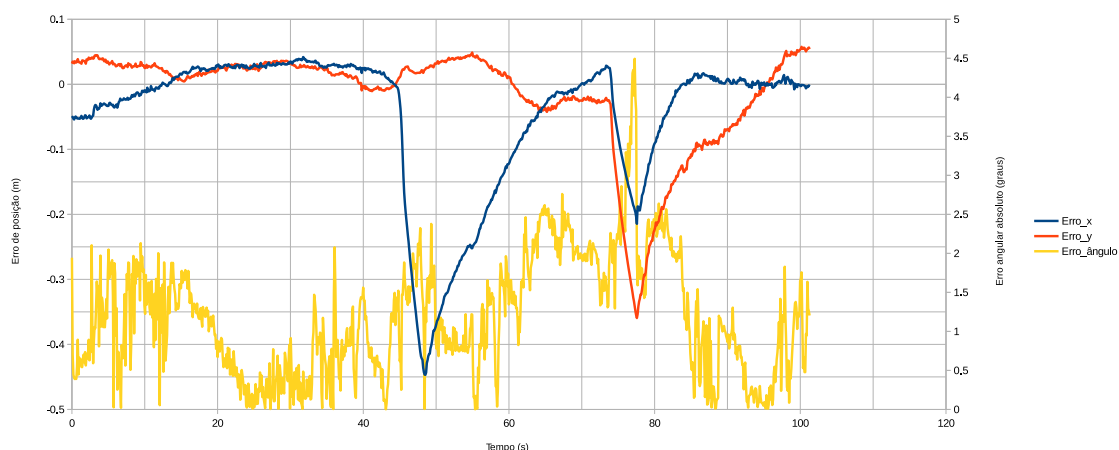
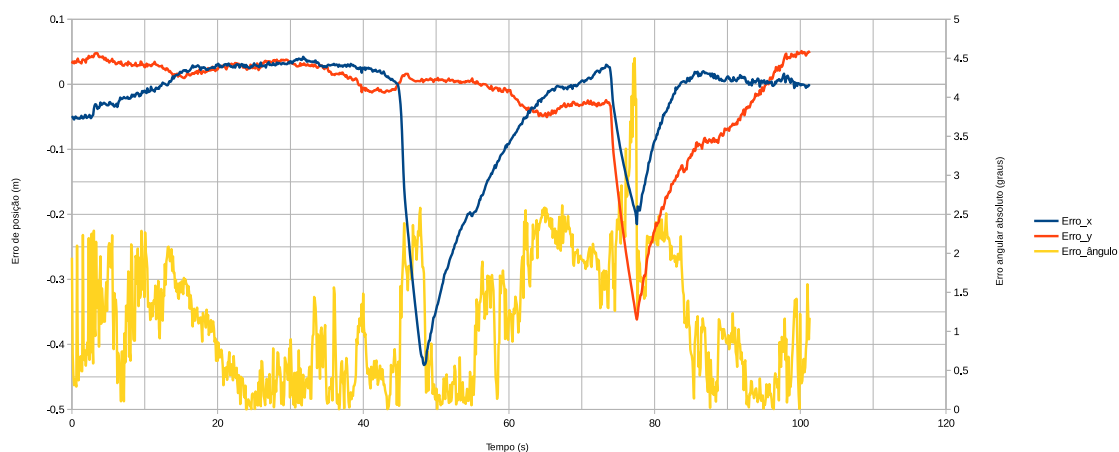


Figura 5.28: (a) Robô na posição inicial (seta azul), bem localizado em relação ao *groundtruth*. (b) e (c) Primeiro e segundo locais, respetivamente onde se forçou um deslizamento das rodas do robô, levando a um aumento do erro da estimativa de localização devido aos dados errados da odometria. (d) O robô volta a convergir para a posição correta.



(a)

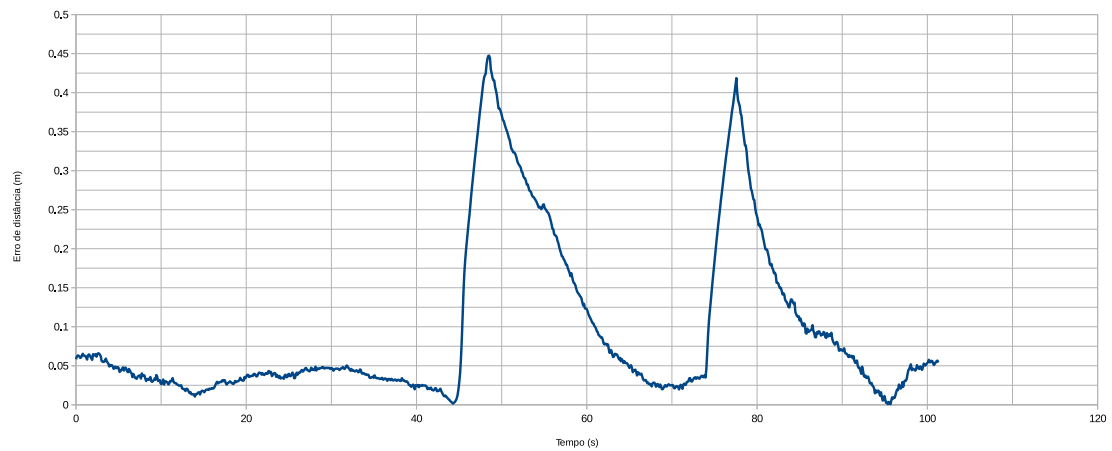


(b)

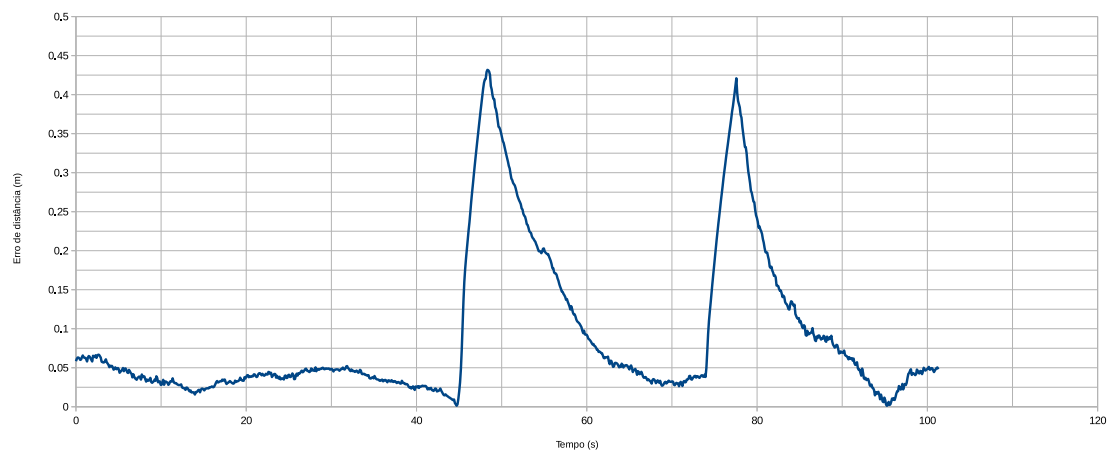
Figura 5.29: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config2_OF_F1F2* e a (b) *Config5_OF_F1F2WD*, com dois momentos onde se forçou o erro da odometria (correspondentes aos dois aumentos de erro visíveis no gráfico).

Na Figura 5.28 encontra-se uma única sequência de imagens ilustrativas do percurso do robô e da estimativa da sua posição (seta azul), comparando-a com o *groundtruth* (seta verde), para as duas configurações (dado a grande semelhança na evolução das duas visualizações ao longo do tempo).

Os resultados para estas duas configurações foram muito similares, como é possível verificar pelos gráficos das Figuras 5.29, 5.30 e pelos dados da Tabela 5.15. Ambas as configurações revelaram ser bastante robustas a erros de odometria, tendo conseguido recuperar rapidamente dessa situação adversa, convergindo para a posição correta. Apesar da similaridade, a Configuração *Config5_OF_F1F2WD* revelou uma ligeira melhoria nos erros de posição e orientação ao longo do percurso.



(a)



(b)

Figura 5.30: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config2_OF_F1F2* e a (b) *Config5_OF_F1F2WD*, com dois momentos onde se forçou o erro da odometria (correspondentes aos dois picos no gráfico).

Tabela 5.15: Resultados estatísticos das *Config2_OF_F1F2* e *Config5_OF_F1F2WD*, em condições de falhas da odometria.

	<i>Config2_OF_F1F2</i>	<i>Config5_OF_F1F2WD</i>
Média do erro de distância (m)	0.0953	0.0911
Média do erro angular absoluto (°)	1.178	1.113
Erro máximo de distância (m)	0.4474	0.4318
Erro angular absoluto máximo (°)	4.494	4.502
Variância do erro da distância	$1.115 \cdot 10^{-2}$	$9.720 \cdot 10^{-3}$
Variância do erro angular	0.5846	0.6522

5.1.3.5 Conclusões

Numa visão geral de todas as configurações apresentadas, em diversos tipos de ambiente, destacaram-se as configurações que utilizam a fusão sensorial entre a odometria e o laser e os filtros de rejeição de *outliers*. O facto do algoritmo integrar os dados fornecidos pela odometria impede que a estimativa de posição ao longo da trajetória do robô oscile ou "salte" de umas posições para as outras e que dependa, nesse caso, unicamente do número de obstáculos presentes no mapa e dos dados fornecidos pelo laser. Mesmo utilizando as informações dadas pela odometria, o algoritmo Perfect Match revelou possuir uma elevada robustez e conseguir recuperar a localização correta, mesmo com dados errados de odometria.

Por outro lado, foi visível que, ao utilizar os filtros de rejeição de pontos desajustados do mapa torna o algoritmo muito menos sensível a ambientes com muitos obstáculos à volta do robô, aumentando a sua fiabilidade. Ainda assim, é importante ter em atenção o número total de pontos que resultam depois de se descartarem os referentes aos *outliers*, pois, se esse número for demasiado baixo, a qualidade da estimativa desce drasticamente e no limite o robô pode perder a sua localização. Deste modo, para aplicações reais, é aconselhável utilizar o parâmetro que analisa a percentagem de pontos totais considerados "bons".

5.2 Análise e comparação de várias configurações do AMCL

Nesta secção será feita uma análise detalhada do algoritmo de localização Monte Carlo Localization (MCL), na sua versão ROS, em que inclui várias extensões e modelos ao algoritmo, denominado de Adaptive Monte Carlo Localization (AMCL). Mais especificamente, serão explicados todos os parâmetros de ajuste disponibilizados por este programa², os quais têm um grande impacto na *performance* e robustez do algoritmo, tornando-se crucial para aplicações mais exigentes, como é o caso do projeto da presente dissertação.

5.2.1 Análise dos parâmetros do AMCL

Parâmetros gerais do filtro

- "min_particles": Número mínimo de partículas permitidas para o filtro.
- "max_particles": Número máximo de partículas permitidas para o filtro.
- "kld_err": Erro máximo entre a distribuição estimada e a real distribuição do filtro.
- "kld_z": Probabilidade do erro entre a distribuição estimada e a distribuição real seja inferior a kld_err.

Estes parâmetros fazem parte do modelo KLD-sampling, que utiliza o método de Kullback–Leibler divergence (KLD) com vista a estimar o erro da distribuição e consequentemente adaptar o número

²A análise matemática será baseada no livro "Probabilistic Robotics" [2]

de partículas ao estado de localização atual do robô. Como a qualidade da estimativa da posição calculada pelo algoritmo MCL é proporcional ao número de partículas mantidas ao longo da sua execução, é complicado encontrar um limiar que por um lado mantenha o peso computacional baixo e que por outro lado forneça uma boa estimativa. Assim, ao adaptar dinamicamente o número de partículas a cada iteração de acordo com as circunstâncias atuais, é possível tornar o algoritmo mais leve, não perdendo muita da sua robustez. No início da execução é necessário utilizar um número elevado de partículas para cobrir uma área suficientemente grande para que a estimativa convirja para a posição real do robô; enquanto que numa fase de seguimento da posição, em que as partículas já estão mais concentradas à volta de uma determinada posição, não justifica manter esse número tão elevado de amostras.

Com a adição do modelo do KLD-sampling ao MCL original, a cada iteração, é calculado um novo número de partículas M_z que estimará na iteração seguinte. O número M_z é calculado de forma a que, com probabilidade $(1 - kld_z)$, o erro entre a distribuição real e a aproximação calculada não exceda o valor kld_err .

- "update_min_d": Movimento de translação necessário para que seja realizado uma atualização do filtro.
- "update_min_a": Movimento de rotação necessário para que seja realizado uma atualização do filtro.
- "resample_interval": Número de atualizações do filtro necessárias para que seja realizado uma reamostragem.

```

1:   Algorithm MCL( $\mathcal{X}_{t-1}, u_t, z_t, m$ ):
2:      $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
3:     for  $m = 1$  to  $M$  do
4:        $x_t^{[m]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$ 
5:        $w_t^{[m]} = \text{measurement\_model}(z_t, x_t^{[m]}, m)$ 
6:        $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
7:     endfor
8:     for  $m = 1$  to  $M$  do
9:       draw  $i$  with probability  $\propto w_t^{[i]}$ 
10:      add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
11:    endfor
12:    return  $\mathcal{X}_t$ 

```

Figura 5.31: Pseudo-algoritmo de Monte Carlo Localization (MCL) [2].

No algoritmo MCL, representado na figura 5.31, é possível identificar 3 momentos principais no processo de cálculo das estimativas de posição. Inicialmente é aplicado o modelo da odometria

(propagação do deslocamento) a cada uma das partículas, calculando-se a posição previsível para cada uma (linha 4). De seguida, a cada uma destas partículas é atribuído um peso (linha 5) de acordo com o modelo das medições dadas pelos sensores do robô (*e.g.*, laser ou sonar - 5.2.1). Às partículas que estiverem numa posição reforçada pelas medições dos sensores ser-lhe-ão atribuídas um peso elevado, enquanto que as partículas localizadas em posições que não estão de acordo com as medições efetuadas terão um peso inferior. Desta forma é terminada a fase de atualização do filtro, obtendo-se um conjunto de partículas distribuídas pelo mapa com diversos pesos.

Por fim, é efetuada uma fase de reamostragem das partículas, em que se cria, com probabilidade proporcional ao peso de cada partícula, novas partículas e eliminando outras com pouca probabilidade. No final resulta o mesmo número de partículas. Ou seja, nesta última ação, as partículas com um peso mais baixo têm uma probabilidade reduzida de serem seleccionadas, desaparecendo as amostras desses locais e concentrando-se nas posições das partículas com maior peso.

Uma característica de grande relevância no algoritmo AMCL é que este é executado de acordo com a movimentação do robô e não de forma periódica, como poderia ser de esperar. As atualizações no filtro ocorrem apenas quando o robô se deslocou um determinado valor de distância (*update_min_d*) ou ângulo (*update_min_a*) desde a estimação anterior. Relacionado com estes dois parâmetros, a taxa de reamostragem (*resampling*) do algoritmo AMCL é dependente do número de atualizações realizadas ao filtro, através do parâmetro *resample_interval*.

Estes três parâmetros serão analisados com mais pormenor com exemplos práticos na Secção 5.2.2., considerando-se que têm um impacto não desprezável no desempenho global do algoritmo.

- "recovery_alpha_slow": Taxa de decaimento exponencial do filtro lento, de média pesada, integrado no modelo que determina a necessidade de adicionar partículas em posições aleatórias pelo mapa.
- "recovery_alpha_fast": Taxa de decaimento exponencial do filtro rápido, de média pesada, integrado no modelo que determina a necessidade de adicionar partículas em posições aleatórias pelo mapa.

Estes dois parâmetros integram o algoritmo Augmented MCL (figura 5.32), cuja ideia principal passa por adicionar partículas aleatórias ao mapa, para recuperar de uma situação de "perdido" ou "raptado", pesando as médias a curto e longo prazo da probabilidade das medições atuais z_t dado o estado atual do robô u_t , o mapa m e as medições efetuadas no ciclo anterior z_{t-1} .

Considera-se que, nesta situação em estudo, estes parâmetros devem ser desligados (atribuindo o valor 0). Isto porque, mesmo no caso em que o robô está perdido, a probabilidade de acertar na posição real deste apenas com algumas partículas distribuídas aleatoriamente pelo mapa é extremamente baixa.

- "initial_pose_x": Coordenada x da média da posição inicial do robô, utilizada para a inicialização do filtro com distribuição Gaussiana.
- "initial_pose_y": Coordenada y da média da posição inicial do robô, utilizada para a inicialização do filtro com distribuição Gaussiana.

```

1:  Algorithm Augmented_MCL( $\mathcal{X}_{t-1}, u_t, z_t, m$ ):
2:    static  $w_{\text{slow}}, w_{\text{fast}}$ 
3:     $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
4:    for  $m = 1$  to  $M$  do
5:       $x_t^{[m]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$ 
6:       $w_t^{[m]} = \text{measurement\_model}(z_t, x_t^{[m]}, m)$ 
7:       $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
8:       $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M} w_t^{[m]}$ 
9:    endfor
10:    $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$ 
11:    $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$ 
12:   for  $m = 1$  to  $M$  do
13:     with probability  $\max(0.0, 1.0 - w_{\text{fast}}/w_{\text{slow}})$  do
14:       add random pose to  $\mathcal{X}_t$ 
15:     else
16:       draw  $i \in \{1, \dots, N\}$  with probability  $\propto w_t^{[i]}$ 
17:       add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
18:     endwith
19:   endfor
20:   return  $\mathcal{X}_t$ 

```

Figura 5.32: Pseudo-algoritmo do modelo Augmented MCL [2].

- "initial_pose_a": Ângulo de orientação da média da posição inicial do robô, utilizada para a inicialização do filtro com distribuição Gaussiana.
- "initial_cov_xx": Covariância $x \times x$ inicial, utilizada para a inicialização do filtro com distribuição Gaussiana.
- "initial_cov_yy": Covariância $y \times y$ inicial, utilizada para a inicialização do filtro com distribuição Gaussiana.
- "initial_cov_aa": Covariância $\theta \times \theta$ inicial, utilizada para a inicialização do filtro com distribuição Gaussiana.

Os parâmetros *initial_pose_x*, *initial_pose_y* e *initial_pose_a* definem a posição e orientação inicial do robô, respectivamente, para a inicialização do filtro. As partículas concentram-se à volta da posição inicial especificada, com uma dispersão igual aos valores das covariâncias *initial_cov_xx*, *initial_cov_yy* e *initial_cov_aa* introduzidas.

Parâmetros do modelo do laser

- "laser_min_range": Alcance mínimo para ser considerado nas medidas do laser. Ao atribuir o valor -1, o programa considera que o alcance mínimo é igual ao valor de distância mínimo no conjunto de dados de cada varrimento.
- "laser_max_range": Alcance máximo para ser considerado nas medidas do laser. Ao atribuir o valor -1, o programa considera que o alcance máximo é igual ao valor de distância máximo no conjunto de dados de cada varrimento.
- "laser_max_beams": Define o número de feixes do laser, igualmente espaçados, a serem considerados na fase de atualização do algoritmo.

Na fase de atribuição de pesos a cada uma das partículas com base nos dados sensoriais do robô (5.31 - linha 5), o algoritmo AMCL apenas utiliza um determinado número de feixes do laser, igualmente espaçados, para os cálculos que efetua, valor especificado pelo parâmetro *laser_max_beams*, cujo valor máximo permitido é 100.

- "laser_z_hit": Peso dado à parte do modelo que considera os dados obtidos pelo laser como sendo corretos (*z_hit*).
- "laser_z_short": Peso dado à parte do modelo que considera os dados obtidos pelo laser como sendo mais curtos do que o esperado (*z_short*).
- "laser_z_max": Peso dado à parte do modelo que considera os dados obtidos pelo laser como sendo demasiado longos (*z_max*).
- "laser_z_rand": Peso dado à parte do modelo que considera uma componente aleatória nos dados obtidos pelo laser (*z_rand*).
- "laser_sigma_hit": Desvio padrão da distribuição Gaussiana usado no modelo *z_hit*.
- "laser_lambda_short": Fator do decaimento exponencial para o modelo *z_short*.
- "laser_likelihood_max_dist": Distância máxima de desvio com o mapa para que esses pontos detetados pelo laser não sejam considerados.
- "laser_model_type": Modelo do laser a utilizar: "beam" ou "likelihood_field".

Para o modelo do laser existem dois algoritmos possíveis, os quais utilizam os parâmetros anteriores, e que serão descritos pormenorizadamente de seguida.

5.2.1.1 Modelo "Beam Range Finder"

A opção "beam" para o parâmetro "laser_model_type" implementa o modelo "beam range finder" o qual calcula a probabilidade em cada medida do sensor correspondente a cada feixe do laser, z_t^k , de um determinado varrimento z_t . Com vista a modelar os possíveis cenários de um varrimento laser, a função de densidade de probabilidade é alcançada a partir de 4 densidades que definem um determinado tipo de erro:

1. Medidas corretas com ruído do laser:

Considerando que as medidas do laser, mesmo devolvendo o valor correcto, estão sujeitas a erros, a função de probabilidade destas medidas, p_{hit} é modelada (5.8) por uma curva Gaussiana de média z_t^{k*} e desvio padrão σ_{hit} (parâmetro do ruído do modelo das medições do laser: *laser_sigma_hit*), representada na Figura 5.33. Para se obter a distância "correta" ao obstáculo medido, utiliza-se o método de *ray casting*, em que é realizado um varrimento ao longo do feixe do laser, usando o mapa (m) e a posição atual estimada para o robô (x_t), com vista a detetar o obstáculo mais próximo ao longo do mesmo.

$$p_{hit}(z_t^k | x_t, m) = \begin{cases} \eta \lambda_{short} \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{1}{2} \frac{(z_t^k - z_t^{k*})^2}{\sigma_{hit}^2}} & 0 \leq z_t^k \leq z_{max} \\ 0 & else \end{cases} \quad (5.8)$$

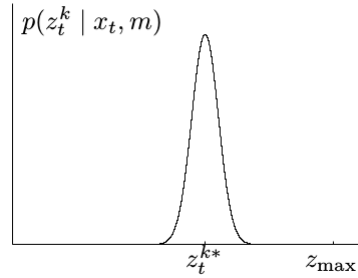


Figura 5.33: Distribuição da densidade de probabilidade Gaussiana p_{hit} [2].

2. Medidas curtas, devido a objectos inesperados:

Com objetos e pessoas constantemente a movimentar-se no ambiente circundante do robô, os dados recolhidos pelo laser são com frequência mais curtos do que se esperava, em comparação com o mapa. Deste modo, é importante modelar estes possíveis resultados como ruído, seguindo a estratégia do ponto 1. A probabilidade de deteção de objetos inesperados à volta do robô decresce com a distância ao mesmo, definindo-se esta probabilidade através de uma distribuição exponencial negativa, p_{short} , formulada em 5.9 e visualizada na Figura 5.34. Esta densidade de probabilidade depende de λ_{short} , um parâmetro intrínseco do modelo das medições do laser, recebido pelo programa através do *laser_lambda_short*.

$$p_{short}(z_t^k | x_t, m) = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short} z_t^k} & 0 \leq z_t^k \leq z_t^{k*} \\ 0 & else \end{cases} \quad (5.9)$$

3. Medidas falhadas:

No caso dos feixes do laser poderem ser absorvidos por superfícies escuras, ou serem refletidos noutras direções, o valor devolvido pelo laser é o máximo valor do alcance do mesmo.

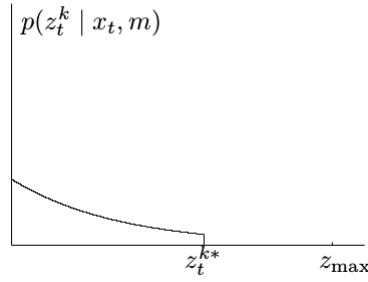


Figura 5.34: Distribuição da densidade de probabilidade exponencial p_{short} [2].

Visto que este é um evento frequente, é importante modelá-lo (5.10), como p_{max} (Figura 5.35), com vista à sua integração no modelo final de densidade de probabilidade do laser.

$$p_{max}(z_t^k | x_t, m) = \begin{cases} 1 & z = z_{\max} \\ 0 & else \end{cases} \quad (5.10)$$

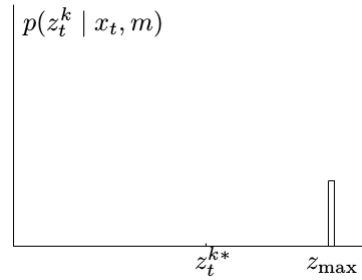


Figura 5.35: Distribuição da densidade de probabilidade uniforme p_{max} [2].

4. Medidas aleatórias:

Por vezes, os lasers devolvem medidas dificilmente explicadas (*e.g.*, por interferência com outros sensores), modelando-se este tipo de dados como eventos aleatórios, com uma distribuição uniforme, p_{rand} , ao longo do intervalo $[0; z_{\max}]$ 5.11, como é possível visualizar na Figura 5.36.

$$p_{rand}(z_t^k | x_t, m) = \begin{cases} \frac{1}{z_{\max}} & 0 \leq z_t^k \leq z_{\max} \\ 0 & else \end{cases} \quad (5.11)$$

Assim, obtém-se a função de densidade de probabilidade combinando estes 4 modelos, representada na Figura 5.37, através de uma média pesada controlada pelos parâmetros z_{hit} , z_{short} , $z_{\max}$

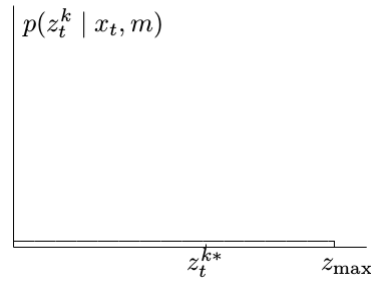


Figura 5.36: Distribuição da densidade de probabilidade uniforme p_{rand} [2].

e z_{rand} , com $z_{hit} + z_{short} + z_{max} + z_{rand} = 1$.

$$p(z_t^k | x_t, m) = \begin{bmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{bmatrix}^T \cdot \begin{bmatrix} p_{hit}(z_t^k | x_t, m) \\ p_{short}(z_t^k | x_t, m) \\ p_{max}(z_t^k | x_t, m) \\ p_{rand}(z_t^k | x_t, m) \end{bmatrix} \quad (5.12)$$

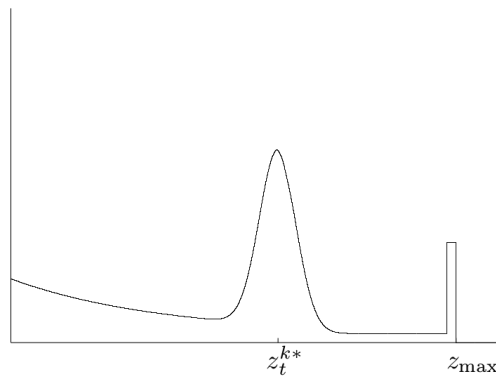


Figura 5.37: Distribuição da densidade de probabilidade combinada dos 4 modelos: p_{hit} , p_{short} , p_{max} e p_{rand} [2].

```

1:   Algorithm beam_range_finder_model( $z_t, x_t, m$ ):
2:        $q = 1$ 
3:       for  $k = 1$  to  $K$  do
4:           compute  $z_t^{k*}$  for the measurement  $z_t^k$  using ray casting
5:            $p = z_{hit} \cdot p_{hit}(z_t^k \mid x_t, m) + z_{short} \cdot p_{short}(z_t^k \mid x_t, m)$ 
6:                $+ z_{max} \cdot p_{max}(z_t^k \mid x_t, m) + z_{rand} \cdot p_{rand}(z_t^k \mid x_t, m)$ 
7:            $q = q \cdot p$ 
8:       return  $q$ 

```

Figura 5.38: Pseudo-algoritmo do modelo "beam range finder" [2].

Com vista ao cálculo da probabilidade de um determinado varrimento do laser, o programa executa o algoritmo representado na Figura 5.38. Inicialmente, para cada feixe do laser, z_t^k , do conjunto de feixes do varrimento, z_t , calcula-se a medida esperada para esse feixe, z_t^{k*} , utilizando-se o método de *ray casting*. Depois, calcula-se individualmente a probabilidade de cada feixe do laser, a partir das funções de probabilidade apresentadas em 5.8, 5.9, 5.10 e 5.11, afetadas pelos pesos z_{hit} , z_{short} , z_{max} e z_{rand} (ver Equação 5.12), que são introduzidas a partir dos parâmetros do programa *laser_z_hit*, *laser_z_short*, *laser_z_max* e *laser_z_rand*, respectivamente. Por fim, a probabilidade do varrimento do laser, p_{z_t} é calculada através do somatório da multiplicação das probabilidades individuais de cada feixe do laser, p_t^k . Na realidade, no programa do algoritmo AMCL, em ROS, a probabilidade do varrimento do laser, em vez de ser calculada por $q = q \cdot p$ é calculada por uma solução considerada *ad-hoc*, mas que devolve bons resultados: $q = q + p^3$.

No entanto, para aplicações de tempo real, o peso computacional deste modelo pode ser demasiado elevado, visto que para calcular a probabilidade da medida de cada feixe do laser é necessário utilizar *ray casting*, que é um algoritmo com grande exigência computacional.

5.2.1.2 Modelo "Likelihood Field"

Com vista a contornar estas limitações surgiu o modelo "likelihood_field", o qual já provou dar bons resultados na prática. Em vez de fazer o seguimento dos feixes do laser (*ray casting*), este método envolve apenas projetar os pontos de deteção de cada feixe no referencial global do mapa, inferindo as suas coordenadas espaciais. Para isso, é necessário conhecer a posição do robô, a distância e o ângulo do determinado ponto em relação ao robô.

De forma similar ao modelo "beam range finder", são modelados 3 densidades de probabilidade:

1. **Medidas corretas com ruído de medição:** Depois de projetar o ponto em coordenadas globais, é calculada a distância euclideana desse ponto detetado pelo laser ao objeto mais

próximo no mapa ($dist$), obtendo-se a densidade de probabilidade, p_{hit} dessa medida a partir de uma distribuição Gaussiana, modelada na Equação 5.13.

$$p_{hit}(z_t^k | x_t, m) = \epsilon_{\sigma_{hit}^2}(dist^2) \quad (5.13)$$

2. **Medidas falhadas:** A distribuição de probabilidade das medidas excessivamente longas, p_{max} é modelada da mesma forma que no caso anterior ("beam range finder").
3. **Medidas aleatórias:** De igual modo, as medições com valores inexplicáveis (aleatórios), p_{rand} , são distribuídas uniformemente pelo intervalo $[0, z_{max}]$.

```

1:   Algorithm likelihood_field_range_finder_model( $z_t, x_t, m$ ):
2:        $q = 1$ 
3:       for all  $k$  do
4:           if  $z_t^k \neq z_{max}$ 
5:                $x_{z_t^k} = x + x_{k,sens} \cos \theta - y_{k,sens} \sin \theta + z_t^k \cos(\theta + \theta_{k,sens})$ 
6:                $y_{z_t^k} = y + y_{k,sens} \cos \theta + x_{k,sens} \sin \theta + z_t^k \sin(\theta + \theta_{k,sens})$ 
7:                $dist^2 = \min_{x', y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \mid \langle x', y' \rangle \text{ occupied in } m \right\}$ 
8:                $q = q \cdot \left( z_{hit} \cdot \mathbf{prob}(dist^2, \sigma_{hit}^2) + \frac{z_{random}}{z_{max}} \right)$ 
9:       return  $q$ 

```

Figura 5.39: Pseudo-algoritmo do modelo "likelihood field range finder" [2].

Assim, na Figura 5.39 está representado o pseudo-algoritmo que descreve este modelo para cada leitura do laser associada a uma dado feixe. Inicialmente, calculam-se as coordenadas no referencial global para cada ponto final de cada feixe do laser ($x_{z_t^k}, y_{z_t^k}$). De seguida, obtém-se a distância dos referidos pontos ao objeto mais próximo existente no mapa, e, com base nesse valor, calcula-se a probabilidade desse feixe, combinando os 3 modelos de distribuição p_{hit} , p_{max} e p_{rand} . O cálculo desta distância é realizado através da consulta a uma *lookup table*, previamente calculada, com as distâncias de cada ponto do mapa ao obstáculo mais próximo. Por fim, a probabilidade do varrimento do laser é dado pelo somatório da multiplicação das probabilidades individuais de cada feixe do laser, à semelhança do modelo "beam".

Parâmetros do modelo da odometria

- "odom_model_type": Modelo da odometria: tração diferencial ("diff") ou tração omnidirecional ("omni").

- "odom_alpha1": Ruído na odometria da estimativa da rotação, a partir da componente da rotação do movimento do robô.
- "odom_alpha2": Ruído na odometria da estimativa da rotação, a partir da componente de translação do movimento do robô.
- "odom_alpha3": Ruído na odometria da estimativa da translação, a partir da componente de translação do movimento do robô.
- "odom_alpha4": Ruído na odometria da estimativa da translação, a partir da componente da rotação do movimento do robô.
- "odom_alpha5": Ruído na odometria da estimativa da translação, utilizada para o modelo omnidirecional.
- "odom_frame_id": Referencial da odometria do robô.
- "base_frame_id": Referencial da base do robô.
- "global_frame_id": Referencial global.
- "tf_broadcast": Booleano para ativar ou desativar a publicação da estimativa da localização a partir da transformada entre o referencial global e a odometria.

```

1:   Algorithm sample_motion_model_odometry( $u_t, x_{t-1}$ ):
2:        $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$ 
3:        $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$ 
4:        $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$ 
5:        $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \text{sample}(\alpha_1 \delta_{\text{rot1}} + \alpha_2 \delta_{\text{trans}})$ 
6:        $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \text{sample}(\alpha_3 \delta_{\text{trans}} + \alpha_4 (\delta_{\text{rot1}} + \delta_{\text{rot2}}))$ 
7:        $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \text{sample}(\alpha_1 \delta_{\text{rot2}} + \alpha_2 \delta_{\text{trans}})$ 
8:        $x' = x + \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot1}})$ 
9:        $y' = y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}})$ 
10:       $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$ 
11:      return  $x_t = (x', y', \theta')^T$ 

```

Figura 5.40: Pseudo-algoritmo do modelo da odometria "sample motion model" [2].

O programa AMCL implementa o modelo da odometria referente ao algoritmo "sample motion model", apresentado na Figura 5.40. Para processar a odometria de forma probabilística é

necessário considerar que a diferença entre duas poses é representada por uma sequência constituída por uma rotação (δ_{θ_1}), seguida de translação linear (δ_d) e outra rotação (δ_{θ_2}).

$$\begin{cases} \delta_{\theta_1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} \\ \delta_d = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2} \\ \delta_{\theta_2} = \bar{\theta}' - \bar{\theta} - \delta_{\theta_1} \end{cases} \quad (5.14)$$

No conjunto de equações 5.14 o algoritmo calcula a movimentação relativa a partir das leituras dos dados da odometria, com $u_t = (\bar{x}_{t-1} \ \bar{x}_t)^T$. As variáveis $(\bar{x}, \bar{y}, \bar{\theta})$ representam a posição e orientação do robô dadas pela odometria no instante $t - 1$ ($\bar{x}_{t-1}$); e as variáveis $(\bar{x}', \bar{y}', \bar{\theta}')$ representam a posição e orientação do robô dadas pela odometria no instante t ($\bar{x}_t$).

$$\begin{cases} \hat{\delta}_{\theta_1} = \delta_{\theta_1} - \varepsilon_{\alpha_1|\delta_{\theta_1}| + \alpha_2|\delta_t|} \\ \hat{\delta}_t = \delta_t - \varepsilon_{\alpha_3|\delta_t| + \alpha_4|\delta_{\theta_1} + \delta_{\theta_2}|} \\ \hat{\delta}_{\theta_2} = \delta_{\theta_2} - \varepsilon_{\alpha_1|\delta_{\theta_2}| + \alpha_2|\delta_t|} \end{cases} \quad (5.15)$$

Em 5.15, calcula-se o movimento relativo do robô, subtraindo as diferenças calculadas em 5.14 a uma função de erro, *sample*, modelada pelos parâmetros α_1 , α_2 , α_3 e α_4 . A função *sample* recebe o desvio padrão σ calculado através de 5.16 e desenha uma distribuição Gaussiana, de forma aleatória, com média nula e desvio padrão σ . Estes valores são controlados através dos parâmetros *odom_alpha1*, *odom_alpha2*, *odom_alpha3* e *odom_alpha4* do programa AMCL, respetivamente.

$$\begin{cases} \varepsilon_{\alpha_1|\delta_{\theta_1}| + \alpha_2|\delta_t|} = \text{sample}(\alpha_1\delta_{\theta_1} + \alpha_2\delta_t) \\ \varepsilon_{\alpha_3|\delta_t| + \alpha_4|\delta_{\theta_1} + \delta_{\theta_2}|} = \text{sample}(\alpha_3\delta_t + \alpha_4(\delta_{\theta_1} + \delta_{\theta_2})) \\ \varepsilon_{\alpha_1|\delta_{\theta_2}| + \alpha_2|\delta_t|} = \text{sample}(\alpha_1\delta_{\theta_2} + \alpha_2\delta_t) \end{cases} \quad (5.16)$$

Por fim, obtém-se a posição estimada do robô na referencial global (x', y', θ') , através das equações 5.17.

$$\begin{cases} x' = x + \hat{\delta}_t \cos(\theta + \delta_{\theta_1}) \\ y' = y + \hat{\delta}_t \sin(\theta + \delta_{\theta_1}) \\ \theta' = \theta + \hat{\delta}_{\theta_1} + \hat{\delta}_{\theta_2} \end{cases} \quad (5.17)$$

Considera-se que na prática, os parâmetros *odom_alpha1*, *odom_alpha2*, *odom_alpha3* e *odom_alpha4* têm um efeito muito relevante na robustez e *performance* do algoritmo, sendo um conjunto de parâmetros a ter em atenção para se obter uma boa configuração do modelo. Por isso, irão apresentar-se exemplos práticos do seu impacto em situações reais na Secção 5.2.2.

5.2.2 Comparação prática de diferentes configurações do AMCL

Os testes realizados têm como objetivo analisar a robustez e a precisão das estimativas nos seguintes conjuntos de situações: ambiente sem obstáculos estranhos ao mapa do robô, ambiente onde existem *outliers* e para o caso de erros de odometria. As experiências realizadas e as diferentes condições do ambiente foram exactamente as mesmas das descritas na Secção 5.1.3, tendo-se também utilizado os mesmos conjuntos de dados.

Com vista a demonstrar o impacto de alguns dos parâmetros configuráveis do programa AMCL, realizaram-se um conjunto de testes nos referidos ambientes, utilizando diferentes configurações para o filtro. Alguns dos parâmetros, com menor interesse para este estudo, serão mantidos constantes para o conjunto das diversas situações e configurações, enquanto que outros serão alterados e comparar-se-á o seu impacto prático.

Todas as configurações que se irão apresentar possuem os valores da Tabela 5.16, para esse conjunto de parâmetros.

Tabela 5.16: Parâmetros comuns a todas as configurações do algoritmo AMCL que serão utilizadas nesta secção.

Parâmetros	Valores
update_min_d	0.1 m
update_min_a	0.17453 rad
resample_interval	1
kld_err	0.01
kld_z	0.99
recovery_alpha_slow	0.0
recovery_alpha_fast	0.0
initial_cov_xx	0.0
initial_cov_yy	0.0
initial_cov_aa	0.0
laser_model_type	"likelihood_field"
laser_min_range	-1
laser_max_range	-1
laser_max_beams	100
laser_z_short	0.0
laser_z_max	0.0
laser_sigma_hit	0.1
laser_likelihood_max_dist	2.0 m
laser_lambda_short	0.1

Os parâmetros *update_min_d*, *update_min_a* e o *resample_interval* são referentes à frequência com que se quer que o programa calcule uma nova estimativa, tendo-se definido os valores de 0.1 m para a distância linear necessária para que o filtro faça uma atualização das partículas ou 0.17453 rad (10 graus) de rotação. Definiu-se que, por cada atualização das partículas é realizado uma reamostragem das mesmas, definindo o valor 1 para esse parâmetro.

Para estas configurações desligou-se a funcionalidade de recuperação do estado de "perdido" ou

”raptado” do robô, atribuindo o valor 0 aos parâmetros *recovery_alpha_slow* e *recovery_alpha_fast*. Assim, os valores dos parâmetros *kld_err* e *kld_z* não são relevantes, tendo-se deixado os *default*.

Um dado importante foi definir-se os parâmetros das covariâncias iniciais (*initial_cov_xx*, *initial_cov_yy* e *initial_cov_aa*) para 0, iniciando o algoritmo com a posição inicial pretendida (próxima da posição inicial do *groundtruth*), sem qualquer dispersão aleatória das partículas à volta dessa posição. Isto porque, havendo uma dispersão inicial, a partícula com maior peso (valor da estimativa do algoritmo) teria algum erro inicial até que o filtro convergisse. Esse erro, ao ser elevado, iria comprometer os resultados estatísticos de interesse para esta análise comparativa (*e.g.*, erro médio de posição, erro angular médio, etc). Para além disso, como a dispersão inicial das partículas é realizada de forma aleatória (dentro dos limites impostos pelas covariâncias definidas), em cada experiência o filtro poderia convergir mais ou menos rápido, beneficiando umas e prejudicando outras. Assim, é de interesse que o algoritmo inicie já com uma posição inicial bem definida, igual para todas as configurações, por forma a que a comparação seja o mais rigorosa possível.

Utilizou-se unicamente o modelo ”likelihood_field” devido ao elevado peso computacional do modelo ”beam” (explicado na Secção 5.2.1). Testes realizados com um processador Intel Core i5 (M 450 @ 2.40GHz) revelaram uma diferença de aproximadamente 10 vezes entre o tempo de ciclo do modelo ”likelihood_field” ($\approx 20\text{-}30$ ms) e o modelo ”beam” ($\approx 200\text{-}300$ ms).

Para os restantes parâmetros, atribuiu-se os valores -1 ao máximo e mínimo alcance do laser, indicando que, a cada varrimento, o valor mínimo e máximo corresponderá aos valores mínimo e máximo desse varrimento do laser, respectivamente. Por outro lado, utilizou-se o número máximo de pontos do laser, igualmente espaçados, permitidos para serem integrados no algoritmo (100). Visto que para o modelo ”likelihood_field” os parâmetros correspondentes às componentes do p_{short} (z_{short}) e p_{max} (z_{max}) não são utilizados, atribuiu-se o valor de 0. Os restantes parâmetros são intrínsecos ao laser.

5.2.2.1 Ambiente sem outliers

Aqui irá analisar-se essencialmente a precisão nas estimativas dadas pelo algoritmo AMCL, em função da alteração dos parâmetros da odometria (*odom_alphas*’s) e do laser (*laser_z_hit* e *laser_z_rand*), com ausência de *outliers* e utilizando um conjunto fixo de 300 partículas para o filtro. Este é o número limite máximo de partículas que poderá ser utilizado durante a operação do robô Clever, visto que o tempo de ciclo do programa é fortemente afetado por este parâmetro (ver Secção 5.3). Considera-se que os parâmetros referidos são os que mais têm impacto no desempenho do filtro, mantendo-se os restantes constantes, para uma melhor comparação.

Por fim, irá alterar-se o número de partículas para algumas das configurações que serão apresentadas, reduzindo-se para 100 e 200, com o objetivo de analisar os resultados estatísticos que possam variar com essa quantidade.

Configurações 1 e 2

Na Tabela 5.17 estão representados os valores dos parâmetros utilizados para as duas primeiras configurações (*Config1_OA_001* e *Config2_OA_02*). Ambas as configurações utilizam um número fixo de partículas (300) e possuem iguais pesos para os dois modelos do laser (p_{hit} e p_{rand}). A principal diferença encontra-se nos parâmetros referentes à odometria, em que a *Config1_OA_001* possui valores mais baixos (menor incerteza para a propagação da odometria) enquanto que a *Config2_OA_02* tem estes valores mais elevados. Estes parâmetros têm um grande impacto na dispersão das partículas (quanto maior o valor dos parâmetros maior a dispersão) ao longo da movimentação do robô. Com isto, quer-se verificar de que forma a precisão e a robustez do algoritmo depende da alteração destes parâmetros.

Tabela 5.17: Definição da *Config1_OA_001* e *Config2_OA_02*.

	<i>Config1_OA_001</i>	<i>Config2_OA_02</i>
min_particles	300	300
max_particles	300	300
laser_z_hit	0.85	0.85
laser_z_rand	0.15	0.15
odom_alpha1	0.01	0.2
odom_alpha2	0.01	0.2
odom_alpha3	0.01	0.2
odom_alpha4	0.01	0.2

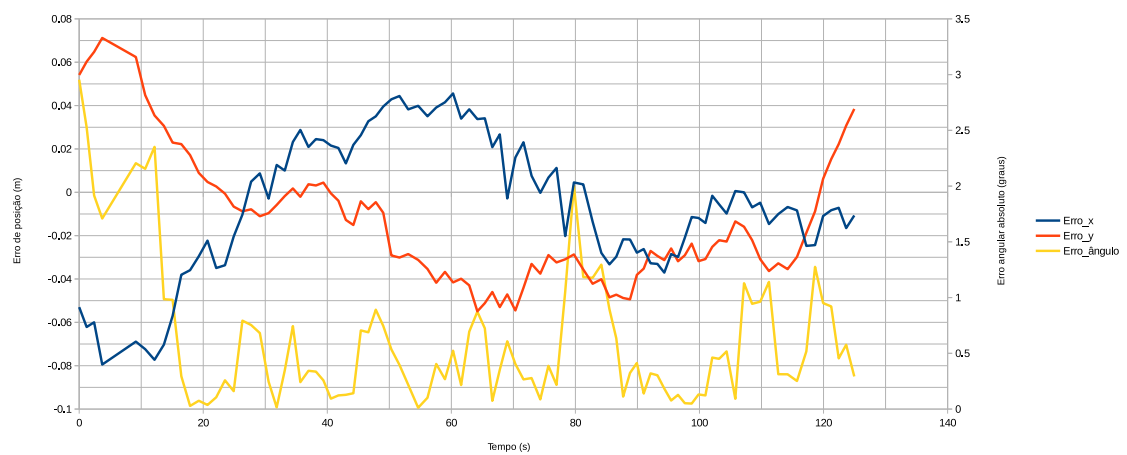
Tabela 5.18: Resultados estatísticos das *Config1_OA_001* e *Config2_OA_02*, num ambiente sem outliers.

	<i>Config1_OA_001</i>	<i>Config2_OA_02</i>
Média do erro de distância (m)	0.0363	0.0405
Média do erro angular absoluto (°)	0.6556	0.6043
Erro máximo de distância (m)	0.0995	0.1067
Erro angular absoluto máximo (°)	3.707	2.955
Variância do erro da distância	$5.015 \cdot 10^{-4}$	$4.039 \cdot 10^{-4}$
Variância do erro angular	0.6928	0.3711

Pelas Figuras 5.41 e 5.42 é possível comparar as evoluções dos erros em x, y, orientação e distância, em relação à posição referenciada pelo *groundtruth*. Os resultados são similares, notando-se um pequeno aumento geral destes erros ao longo do percurso do robô, no caso da *Config2_OA_02*, pelo facto dos parâmetros da odometria possuírem valores mais elevados, aumentando a incerteza na movimentação do robô e consequentemente dispersando um pouco mais as partículas, levando a uma maior oscilação nas estimativas.

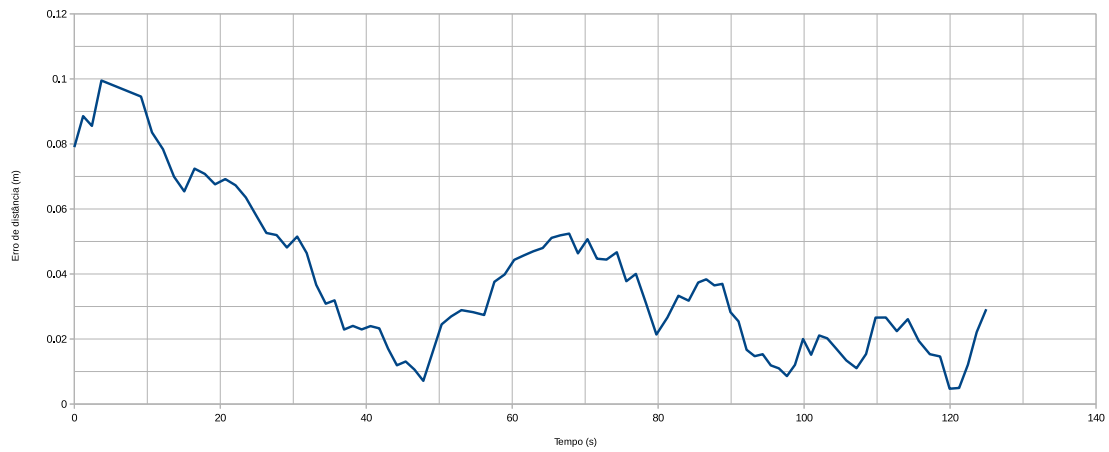


(a)

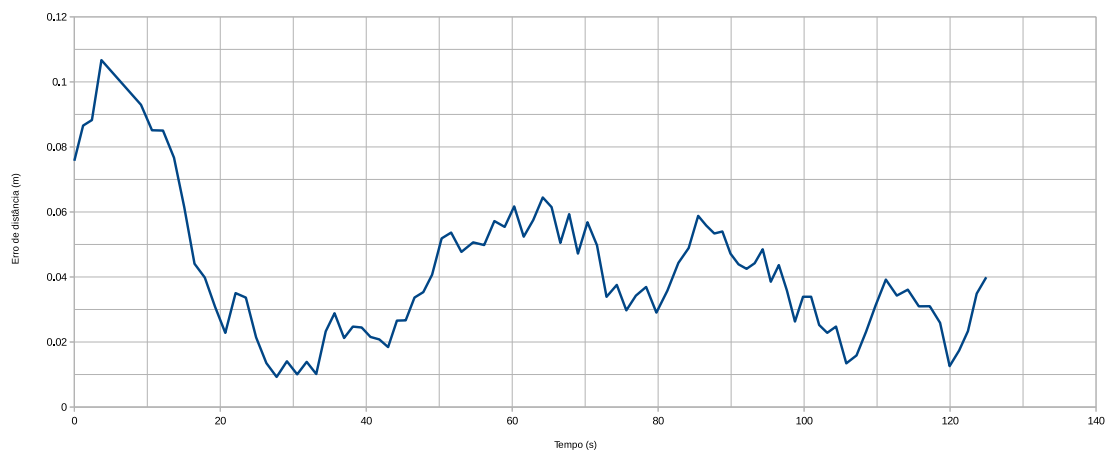


(b)

Figura 5.41: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config1_OA_001* e a (b) *Config2_OA_02*, na ausência de outliers.



(a)



(b)

Figura 5.42: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config1_OA_001* e a (b) *Config2_OA_02*, na ausência de *outliers*.

Configurações 3 e 4

Prepararam-se mais duas configurações (*Config3_OA_05* e *Config4_OA_05L*), com os valores especificados na Tabela 5.19. Para o caso da *Config3_OA_05*, mantiveram-se constantes os parâmetros das configurações anteriores, tendo-se apenas aumentado o valor dos coeficientes da odometria para 0.5, dispersando ainda mais as partículas com o movimento do robô. Na *Config4_OA_05L*, para além disto, alterou-se os pesos dados aos modelos do laser p_{hit} e p_{rand} , atribuindo o valor máximo ao primeiro e consequentemente eliminando a componente aleatória. Assim, todos os dados do laser serão processados de acordo com o modelo p_{hit} , dando, desta forma, um peso superior à componente do laser, comparativamente às configurações anteriores.

Tabela 5.19: Definição da *Config3_OA_05* e *Config4_OA_05L*.

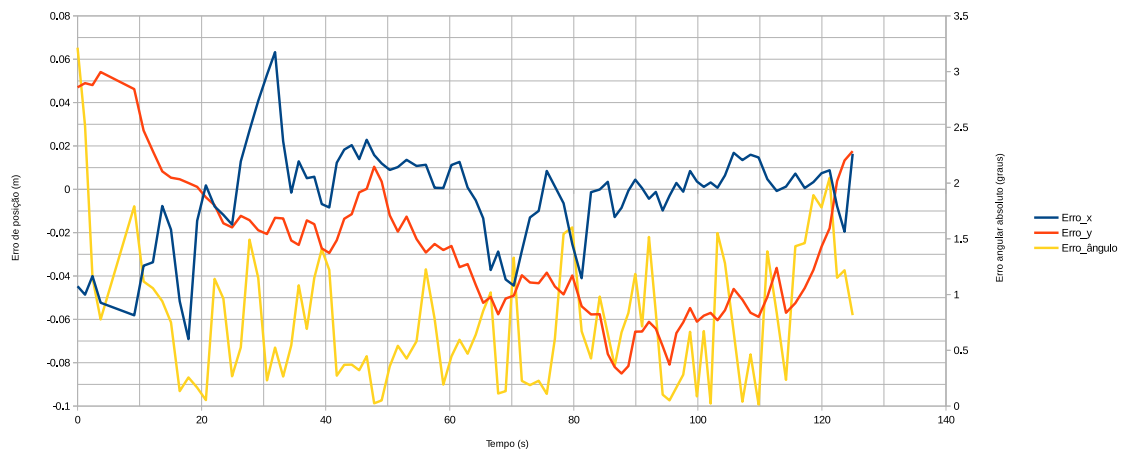
	<i>Config3_OA_05</i>	<i>Config4_OA_05L</i>
min_particles	300	300
max_particles	300	300
laser_z_hit	0.85	1.0
laser_z_rand	0.15	0.0
odom_alpha1	0.5	0.5
odom_alpha2	0.5	0.5
odom_alpha3	0.5	0.5
odom_alpha4	0.5	0.5

Pelo conjunto de resultados obtidos para estas 4 configurações (Tabelas 5.18 e 5.20), verifica-se que há um pequeno aumento (≈ 4 a 7 mm) no erro médio da estimativa de posição à medida que se aumenta os parâmetros referentes ao ruído esperado da odometria.

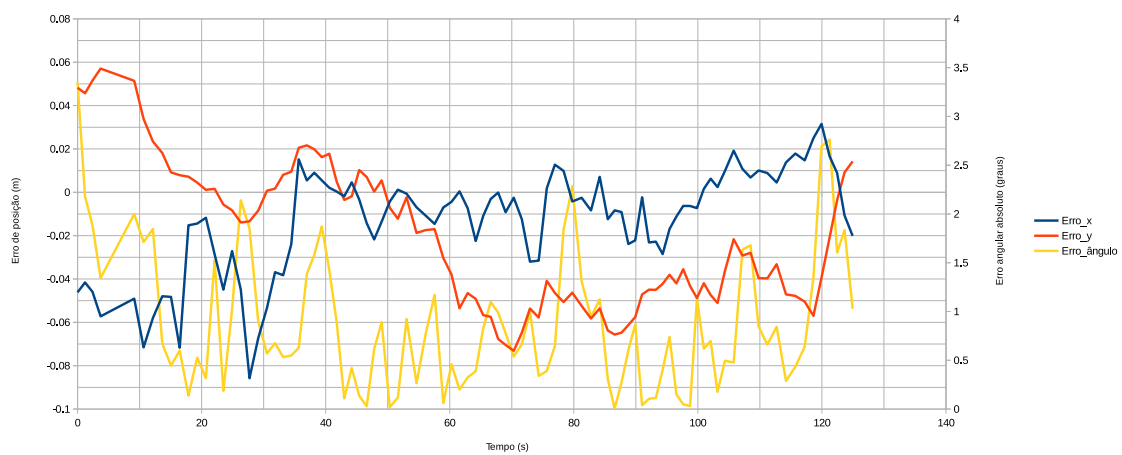
Com o aumento do peso dado ao z_{hit} do laser, notou-se uma ligeira melhoria no erro de posição (≈ 2 mm em média) ao longo do tempo (Figuras 5.43 e 5.44, visto que o ambiente utilizado para estes testes não possui objetos externos ao mapa e por isso o laser retorna essencialmente dados corretos. Mesmo assim, os resultados obtidos com estas 4 configurações são bastante similares para um ambiente sem *outliers*, como seria de esperar.

Tabela 5.20: Resultados estatísticos das *Config3_OA_05* e *Config4_OA_05L*, com grande quantidade de obstáculos no mapa.

	<i>Config3_OA_05</i>	<i>Config4_OA_05L</i>
Média do erro de distância (m)	0.0438	0.0425
Média do erro angular absoluto ($^{\circ}$)	0.7640	0.8721
Erro máximo de distância (m)	0.0854	0.0868
Erro angular absoluto máximo ($^{\circ}$)	3.215	3.348
Variância do erro da distância	$4.362 \cdot 10^{-4}$	$4.220 \cdot 10^{-4}$
Variância do erro angular	0.3584	0.4931

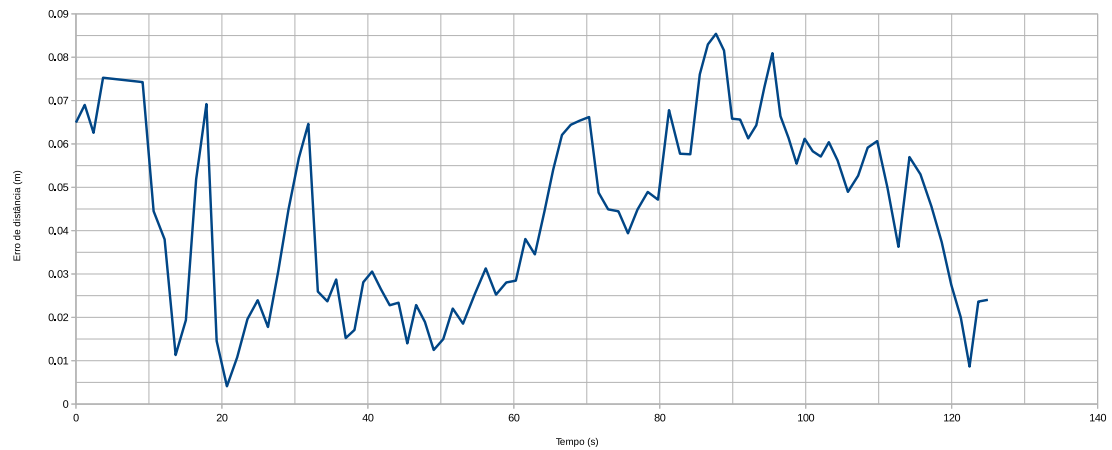


(a)

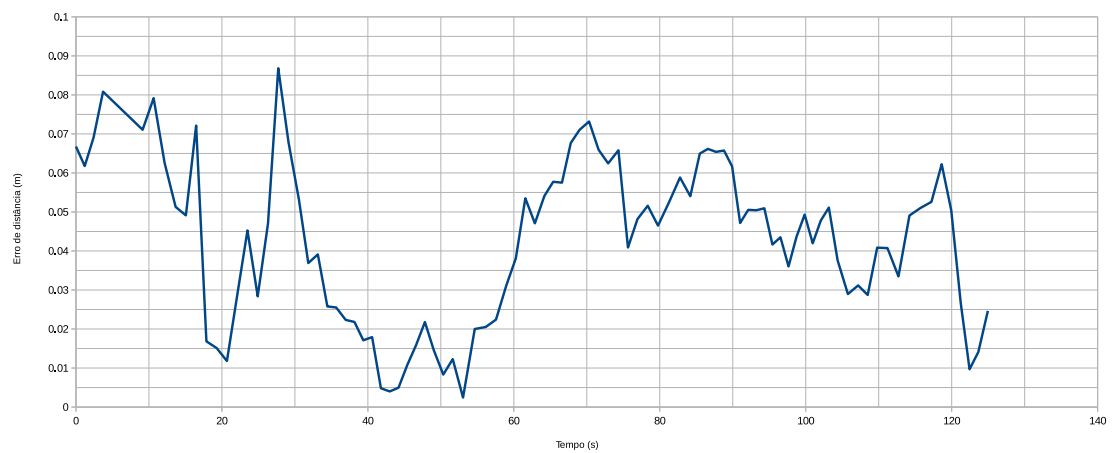


(b)

Figura 5.43: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config3_OA_05* e a (b) *Config4_OA_05L*.



(a)



(b)

Figura 5.44: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config3_OA_05* e a (b) *Config4_OA_05L*.

Na Figura 5.45 estão representados as três parametrizações escolhidas para o modelo da odometria, onde se pode verificar que, à medida que os valores atribuídos a esses parâmetros aumentam, a dispersão das partículas ao longo do movimento do robô também aumenta. Essa dispersão é realizada de forma aleatória a partir de uma distribuição Gaussiana com média nula e desvio padrão σ . O desvio padrão é controlado proporcionalmente através dos parâmetros da odometria (*odom_alpha's*), aumentando com o aumento destes valores.

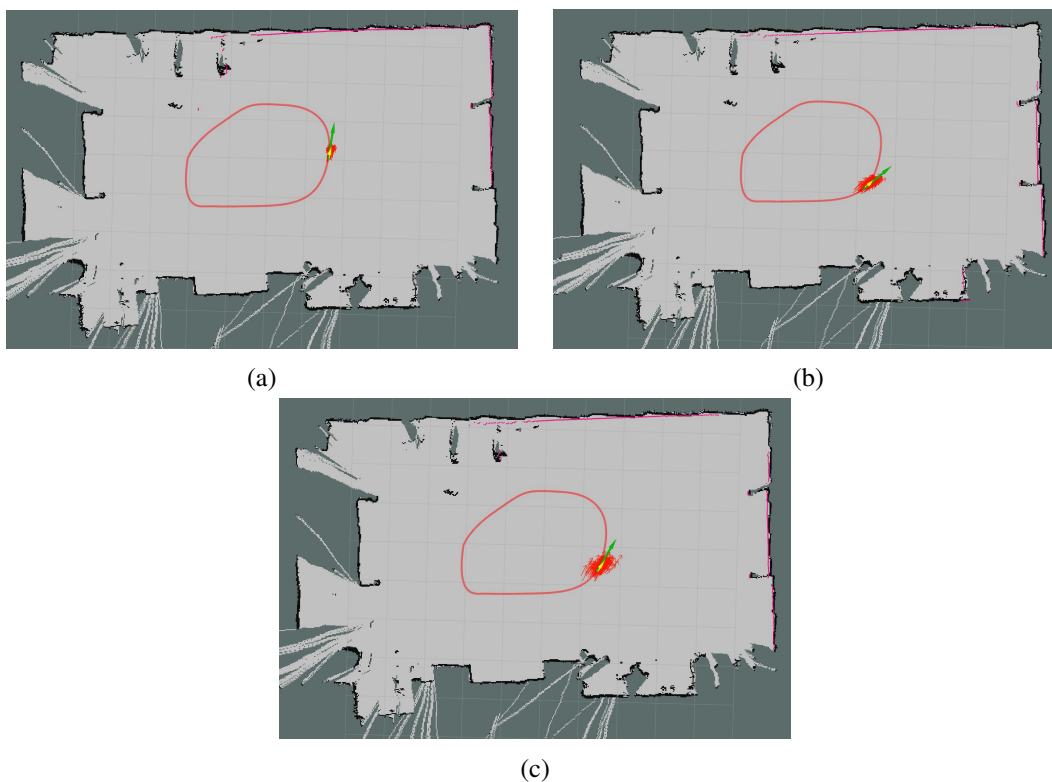


Figura 5.45: Dispersão das partículas em função dos parâmetros *odom_alpha's* com (a) 0.01 (b) 0.2 e (c) 0.5.

5.2.2.2 Erros forçados de odometria

Com vista a testar os efeitos dos erros de odometria no algoritmo AMCL, forçou-se a derrapagem das rodas do robô em dois momentos da trajetória.

Para este caso, considerou-se interessante analisar as [Config1_OA_001](#), [Config2_OA_02](#) e [Config3_OA_05](#). Visto que os erros induzidos nesta secção estão relacionados com a odometria do robô, excluiu-se desta análise a [Config4_OA_05L](#), cuja única diferença de parametrização está relacionada com o modelo do laser.

[Config1_OA_001](#), [Config2_OA_02](#) e [Config3_OA_05](#)

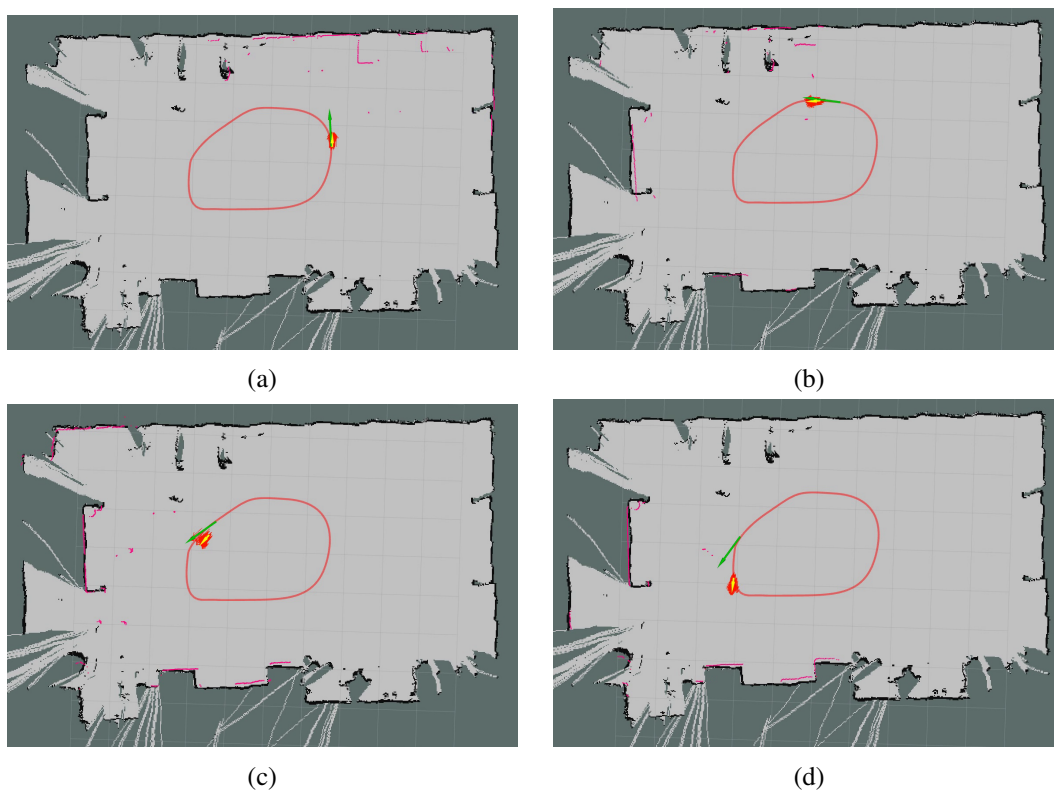


Figura 5.46: (a) Robô bem localizado, em relação ao *groundtruth* (seta verde) com uma reduzida dispersão de partículas (setas vermelhas). (b) No primeiro local onde se forçou o erro de odometria a estimativa de posição dada pelo AMCL (seta amarela) ganhou algum erro. (c) Na segunda zona onde houve um erro de odometria o robô já estava com um elevado erro de posição e em (d) continuou a divergir, perdendo-se.

Estes exemplos são ilustrativos da importância de possuir uma boa dispersão de partículas à volta da posição estimada do robô, ao longo da sua movimentação, controlada pelos fatores de ajuste da odometria (*odom_alpha*'s). O facto de se configurar estes parâmetros com valores superiores ao ruído esperado pela odometria traz a vantagem de aumentar a robustez do algoritmo a situações inesperadas em que há perturbação da movimentação do robô por fatores externos ao modelo calculado, permitindo a sua recuperação (Figura 5.47). Na Figura 5.46 é possível verificar-se

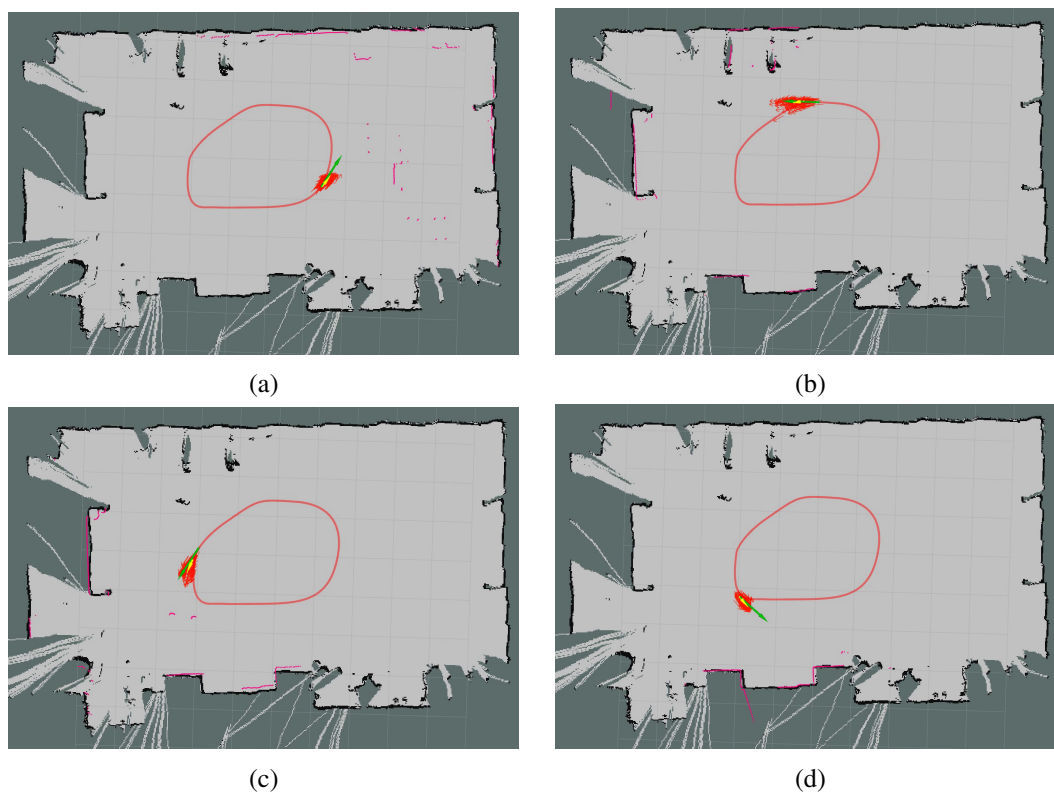


Figura 5.47: (a) Robô bem localizado, em relação ao *groundtruth* (seta verde) com alguma dispersão de partículas (setas vermelhas). (b) No primeiro local onde se forçou o erro de odometria a estimativa de posição dada pelo AMCL (seta amarela) ganhou algum erro. (c) Segunda zona onde houve um erro nos dados da odometria. (d) O robô conseguiu recuperar a localização correta.

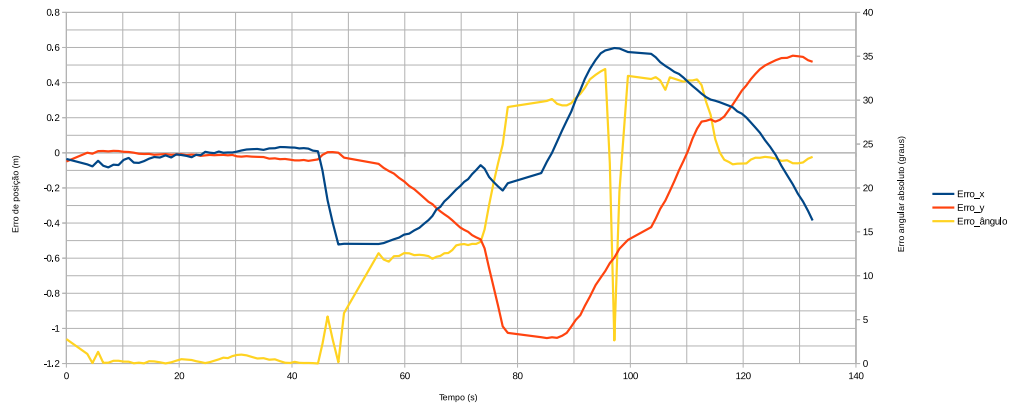
um fenómeno denominado de *particle deprivation* [2], que ocorre quando há uma maior concentração das partículas à volta de uma determinada posição errada e consequentemente deixam de haver partículas na localização correta do robô, impossibilitando a sua recuperação.

Tabela 5.21: Resultados estatísticos das *Config1_OA_001*, *Config2_OA_02* e *Config3_OA_05*, com erros forçados de odometria, em dois momentos.

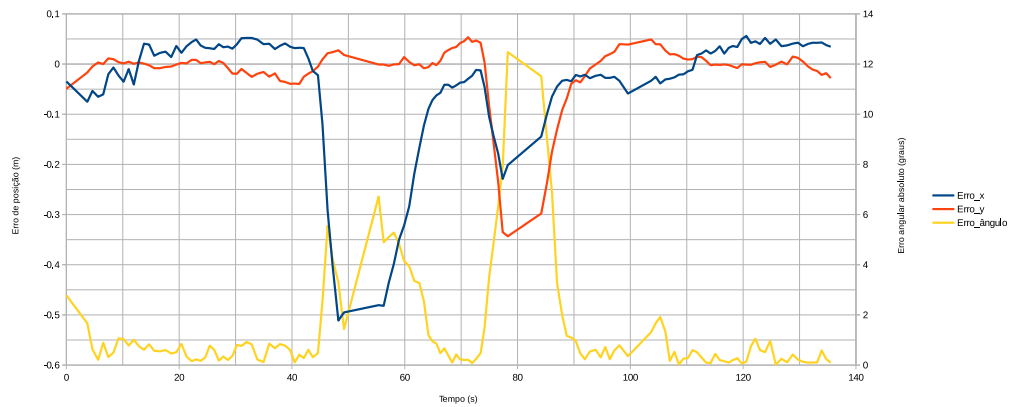
	<i>Config1_OA_001</i>	<i>Config2_OA_02</i>	<i>Config3_OA_05</i>
Média do erro de distância (m)	0.3984	0.0875	0.0758
Média do erro angular absoluto (°)	14.08	1.370	1.338
Erro máximo de distância (m)	1.056	0.5119	0.5317
Erro angular absoluto máximo (°)	33.55	12.48	8.967
Variância do erro da distância	0.1055	0.0132	0.0119
Variância do erro angular	154.7	4.609	2.601

A parametrização *Config3_OA_05* devolve valores similares aos da *Config2_OA_02*, tendo-se utilizado os coeficientes da odometria um pouco mais elevados. Assim, é possível verificar que o aumento da dispersão das partículas, para este exemplo, diminui o tempo de recuperação da falha

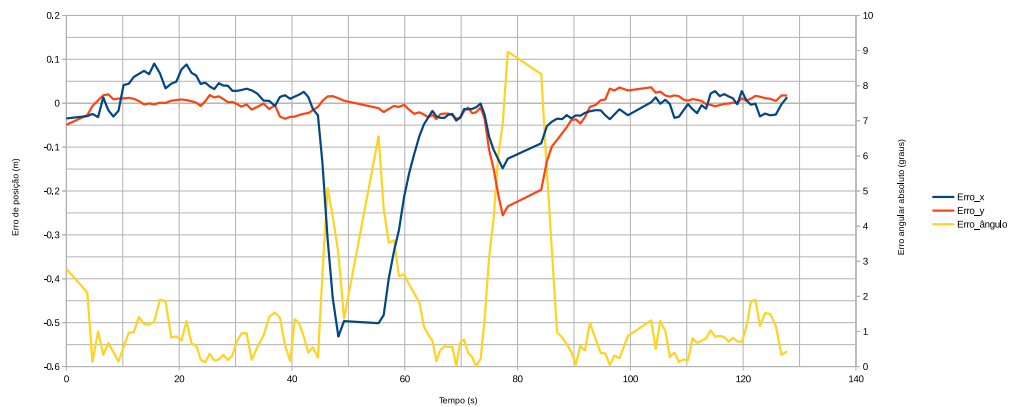
de odometria (Figuras 5.48 e 5.49) e diminui o erro geral ao longo do percurso, representado na Tabela 5.21. À medida que se aumenta a dispersão das partículas, a tolerância a erros de odometria também cresce, aumentando a área de erro a partir da qual é possível recuperar a posição correta.



(a)

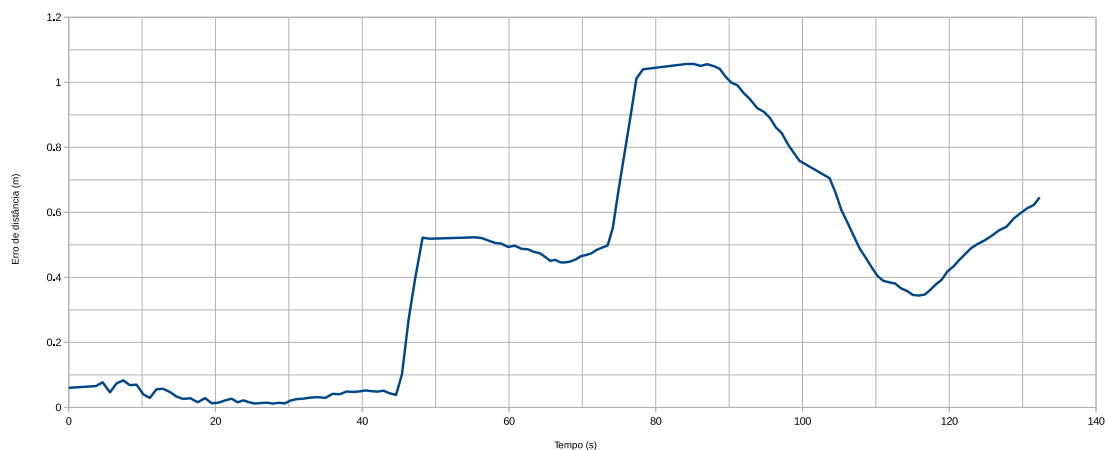


(b)

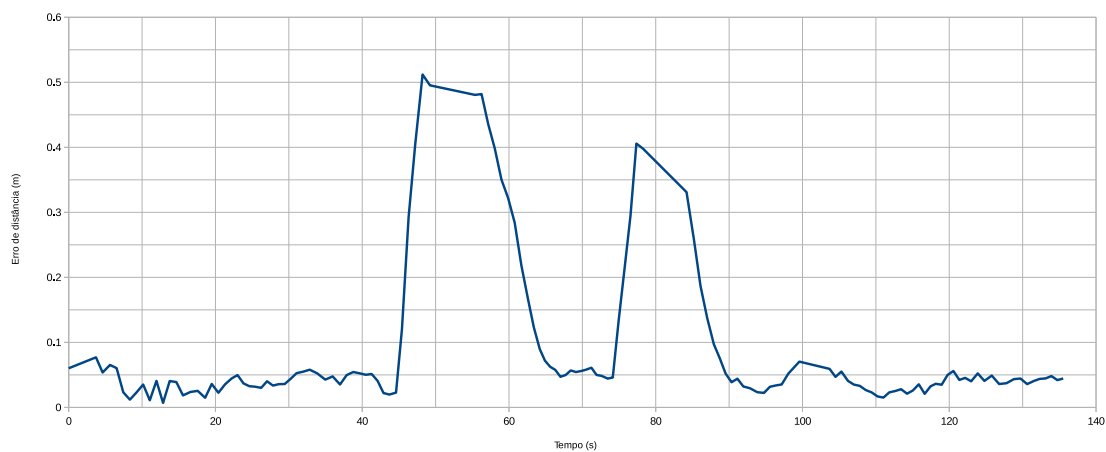


(c)

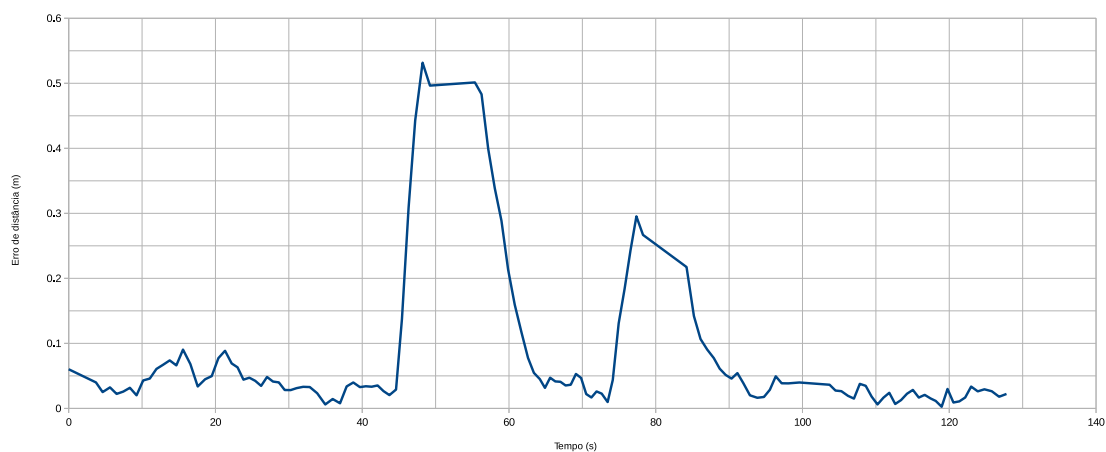
Figura 5.48: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config1_OA_001*, (b) *Config2_OA_02* e (c) *Config3_OA_05*.



(a)



(b)



(c)

Figura 5.49: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config1_OA_001*, (b) *Config2_OA_02* e (c) *Config3_OA_05*. É possível identificar os dois momentos em que se forçou um erro de odometria, correspondentes aos picos visíveis nas curvas do erro.

5.2.2.3 Ambiente com alguns outliers

Para testar a robustez do algoritmo a obstáculos externos ao mapa, alterou-se o ambiente de testes para as condições já descritas na Secção 5.1.3.2. A *Config1_OA_001* utiliza valores mais baixos para os coeficientes da odometria do que as Configurações *Config2_OA_02* e *Config4_OA_05L*, com esta última a possuir o valor máximo para o peso z_{hit} . A *Config3_OA_05* utiliza os mesmo parâmetros que a *Config4_OA_05L*, à exceção do modelo do laser, sendo mais interessante verificar o impacto deste parâmetro para a situação em que se colocam obstáculos que perturbam os dados recebidos pelo laser.

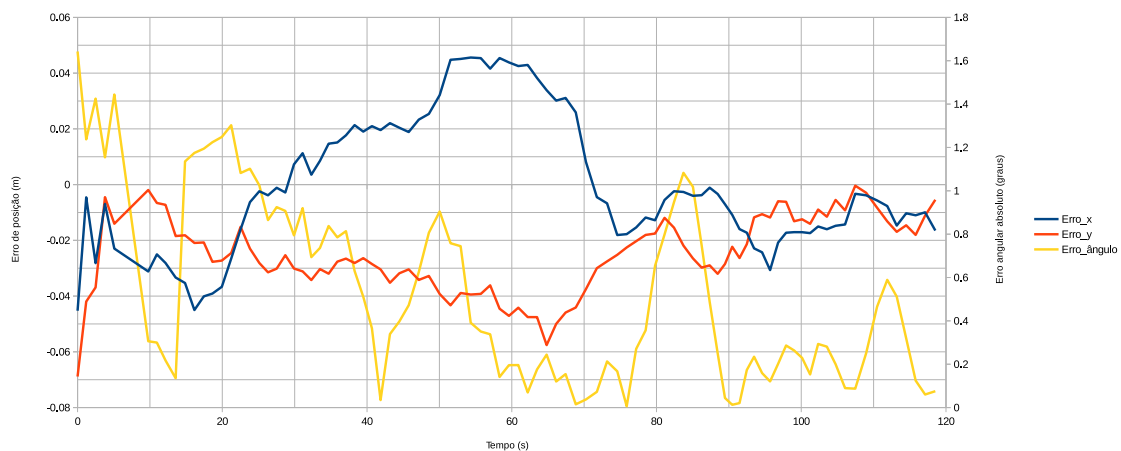
Config1_OA_001, *Config2_OA_02* e *Config4_OA_05L*

Tabela 5.22: Resultados estatísticos das *Config1_OA_001*, *Config2_OA_02* e *Config4_OA_05L*, num ambiente com alguns obstáculos.

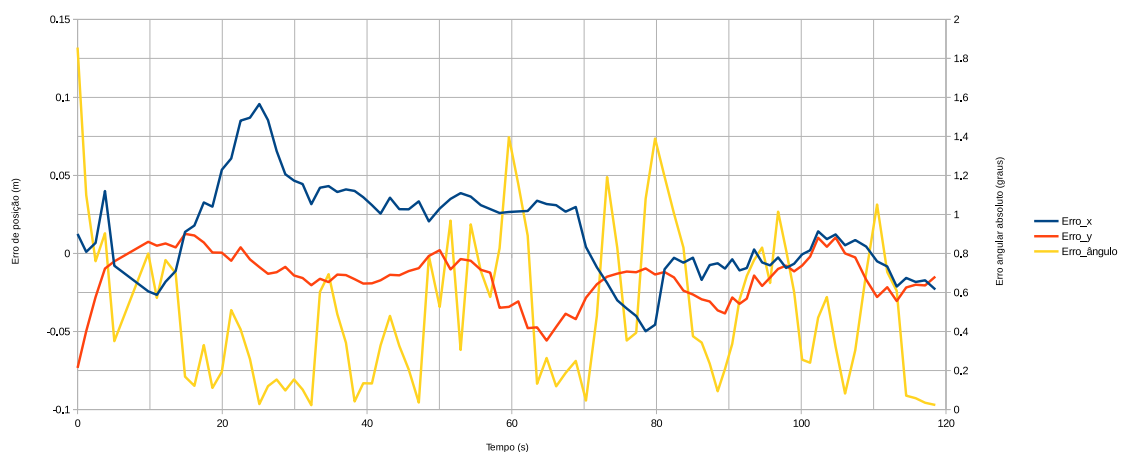
	<i>Config1_OA_001</i>	<i>Config2_OA_02</i>	<i>Config4_OA_05L</i>
Média do erro de distância (m)	0.0338	0.0351	0.0430
Média do erro angular absoluto (°)	0.5214	0.5029	0.9272
Erro máximo de distância (m)	0.0824	0.0962	0.1289
Erro angular absoluto máximo (°)	1.643	1.856	2.821
Variância do erro da distância	$2.474 \cdot 10^{-4}$	$3.472 \cdot 10^{-4}$	$7.446 \cdot 10^{-4}$
Variância do erro angular	0.1753	0.1461	0.4277

Para estas circunstâncias, num ambiente com alguns *outliers*, pode verificar-se um bom desempenho por parte das três configurações (Figuras 5.50 e 5.51), sendo que, para o caso em que se aumentou o valor do fator para a componente z_{hit} do modelo do laser (*Config4_OA_05L*), o erro foi superior aos outros dois casos. Isto porque os obstáculos presentes no mapa, ao serem detetados pelo laser, levaram a um aumento do erro na localização calculada pelo algoritmo.

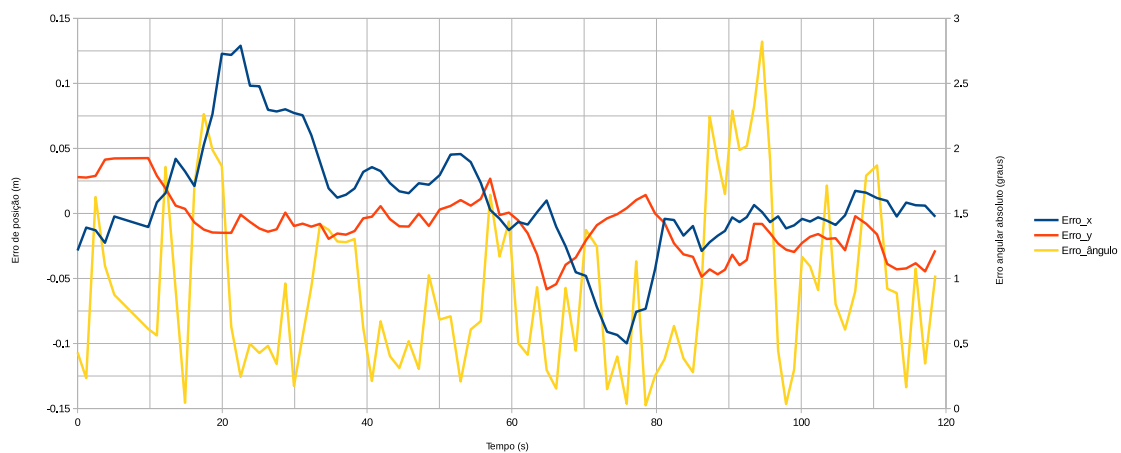
Para as duas primeiras configurações, como já se tinha verificado, o aumento da dispersão das partículas ao longo do movimento do robô faz aumentar um pouco o erro na estimativa do algoritmo (Tabela 5.22), mas aumenta a robustez a fenómenos inesperados relacionados com a movimentação do robô.



(a)

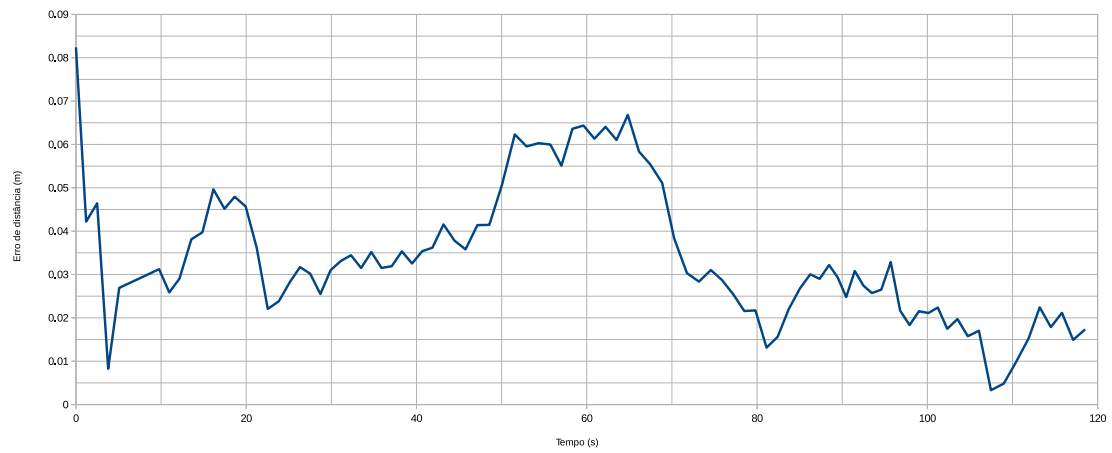


(b)

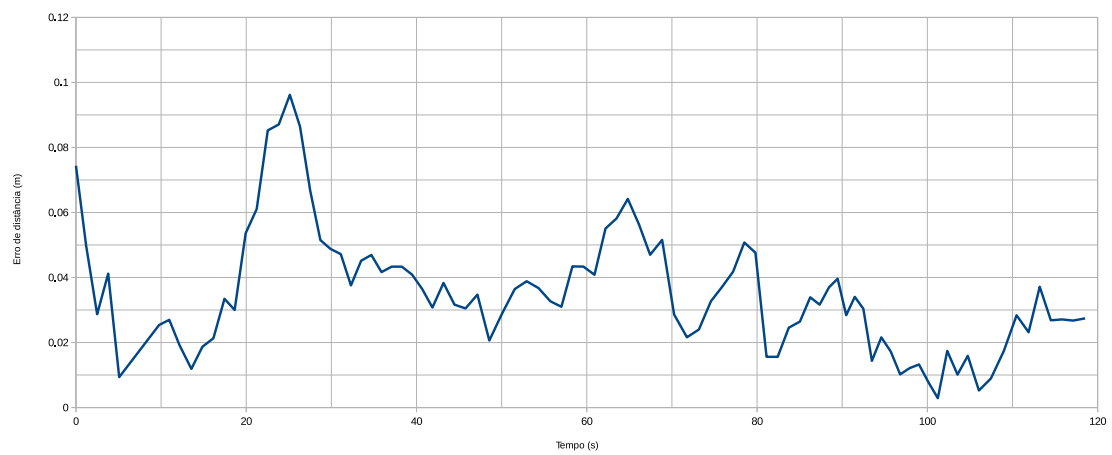


(c)

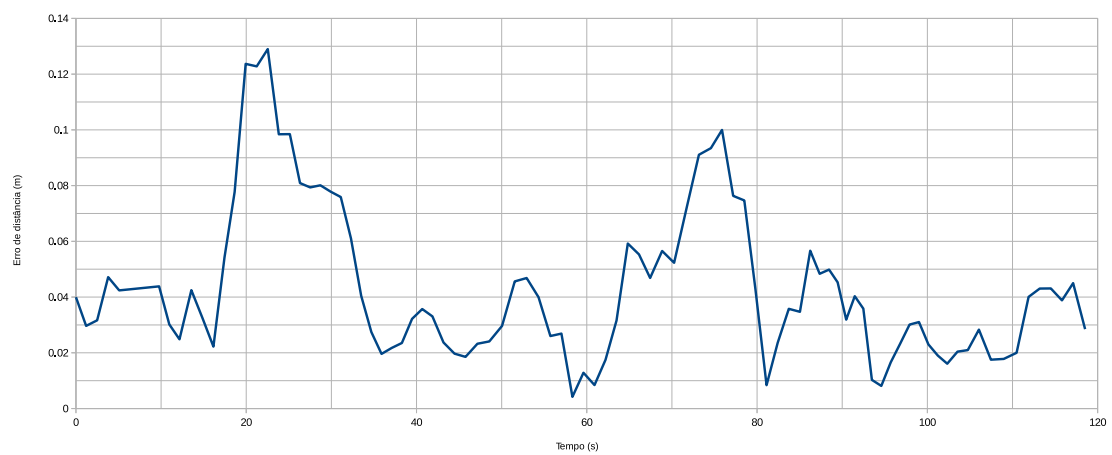
Figura 5.50: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config1_OA_001*, (b) *Config2_OA_02* e (c) *Config4_OA_05L*.



(a)



(b)



(c)

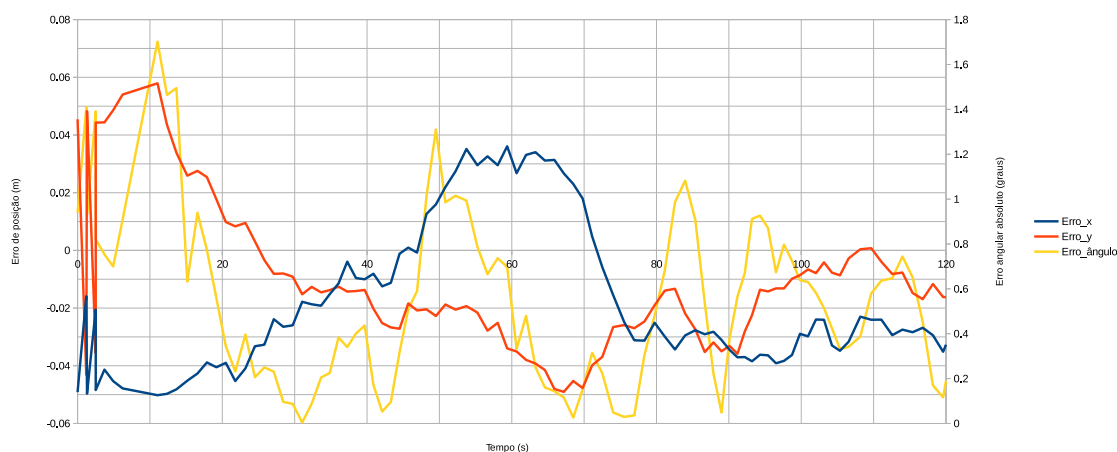
Figura 5.51: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config1_OA_001*, (b) *Config2_OA_02* e (c) *Config4_OA_05L*.

5.2.2.4 Ambiente com grande quantidade outliers

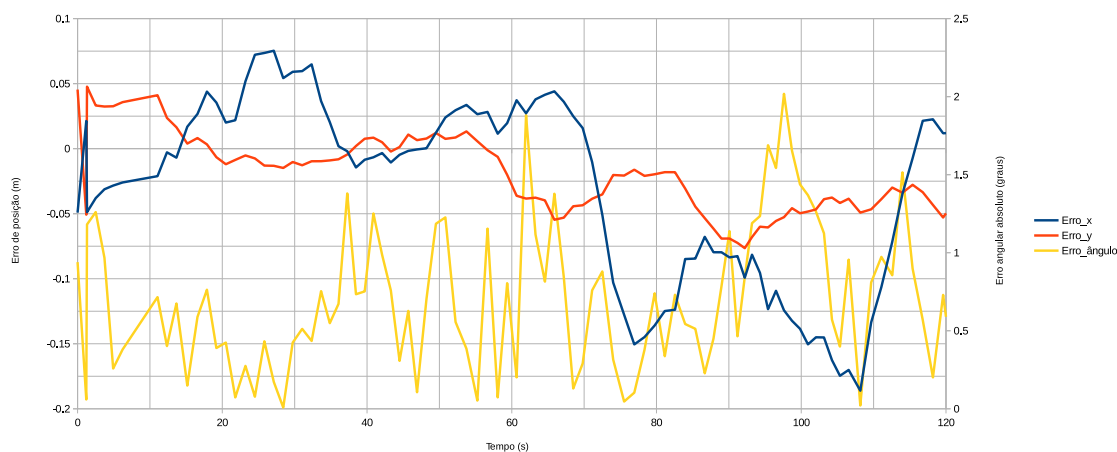
De forma semelhante ao anteriormente descrito na Secção 5.1.3.3, testou-se algumas configurações numa situação com elevada quantidade de objetos estranhos ao mapa.

Config1_OA_001 e Config2_OA_02

Para estes dois exemplos, é importante realçar a maior precisão na estimativa de localização do filtro (Tabela 5.23) quando se utilizam valores mais reduzidos nos parâmetros relacionados com a odometria (Figuras 5.52 e 5.53), concentrando mais as suas partículas ao longo da movimentação do robô e por isso ficando mais imune aos *outliers* detetados pelo laser. Por outro lado, como já foi analisado na Secção 5.2.2.2, uma demasiada concentração das partículas pode levar à perda do robô, caso haja uma perturbação física na sua movimentação. Deste modo, é necessário encontrar um conjunto de valores para os parâmetros da odometria que tenham em conta este *tradeoff*.

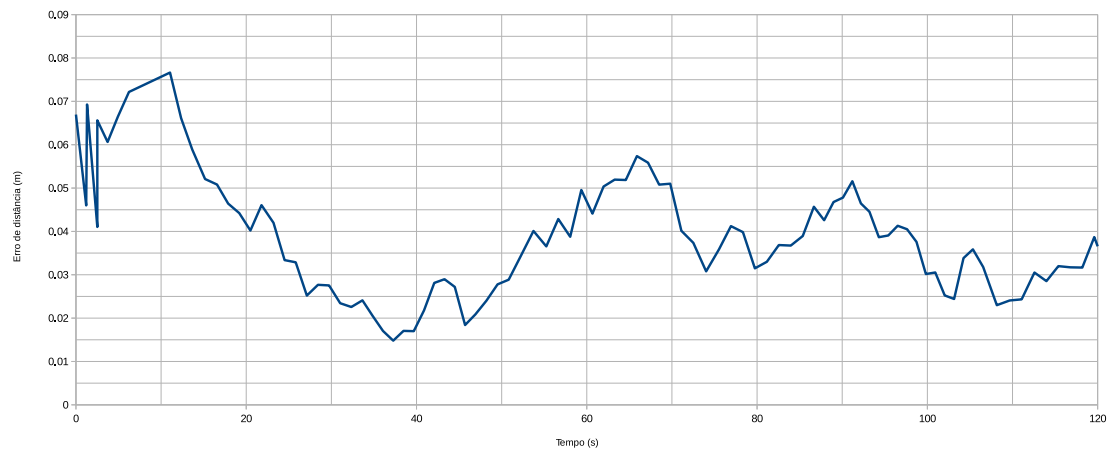


(a)

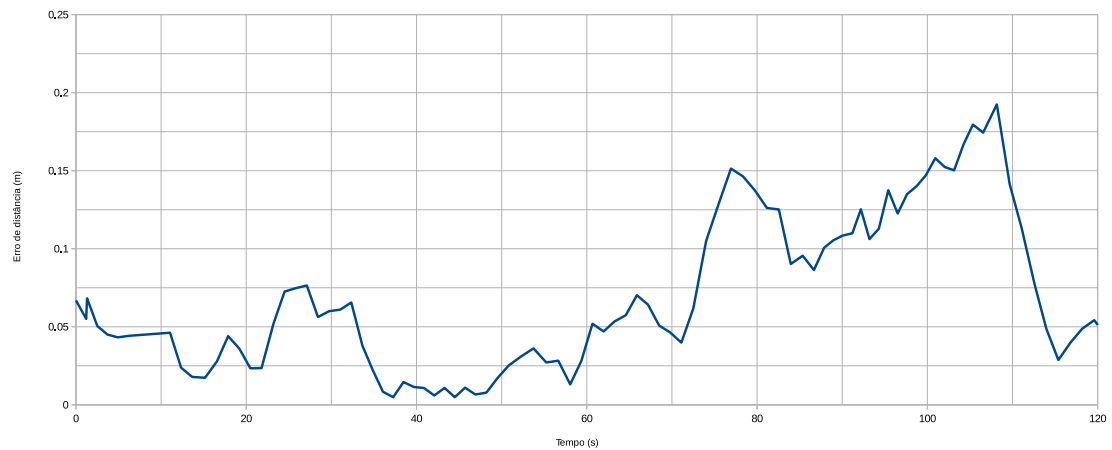


(b)

Figura 5.52: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config1_OA_001* e a (b) *Config2_OA_02*.



(a)



(b)

Figura 5.53: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config1_OA_001* e a (b) *Config2_OA_02*.

Tabela 5.23: Resultados estatísticos das *Config1_OA_001* e *Config2_OA_02*, num ambiente com grande número de obstáculos.

	<i>Config1_OA_001</i>	<i>Config2_OA_02</i>
Média do erro de distância (m)	0.0386	0.0696
Média do erro angular absoluto (°)	0.5488	0.7069
Erro máximo de distância (m)	0.0766	0.1925
Erro angular absoluto máximo (°)	1.702	2.0182
Variância do erro da distância	$1.848 \cdot 10^{-4}$	$2.460 \cdot 10^{-3}$
Variância do erro angular	0.1438	0.2164

Config3_OA_05 e Config4_OA_05L

Para estas duas configurações são utilizados valores elevados nos parâmetros do modelo do ruído da odometria, aumentando a dispersão ao longo do movimento do robô, tornando-se mais sensível aos dados recebidos do laser, como é possível verificar na Figura 5.54. Isto porque, havendo uma maior variedade no conjunto das estimativas à volta da posição do robô, e como os pesos atribuídos a essas estimativas estão dependentes dos dados do laser, a influência desses pontos resulta numa mais provável alteração das posições estimadas, do que se as partículas estivessem todas sensivelmente na mesma localização.

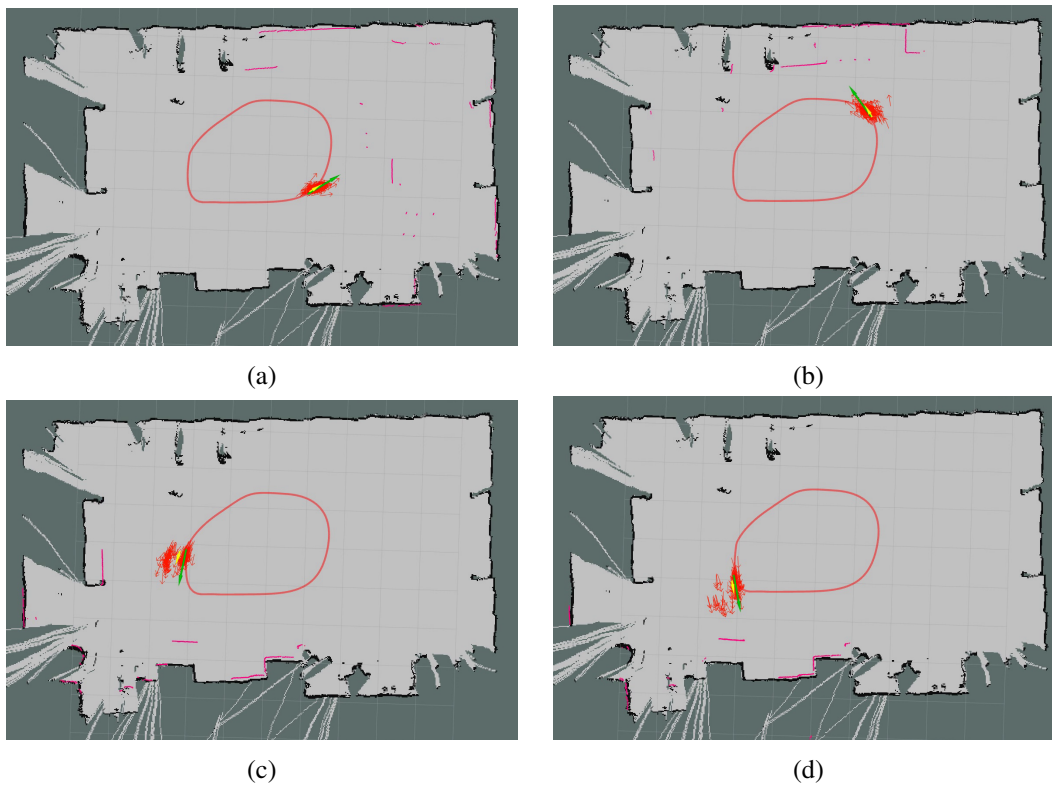
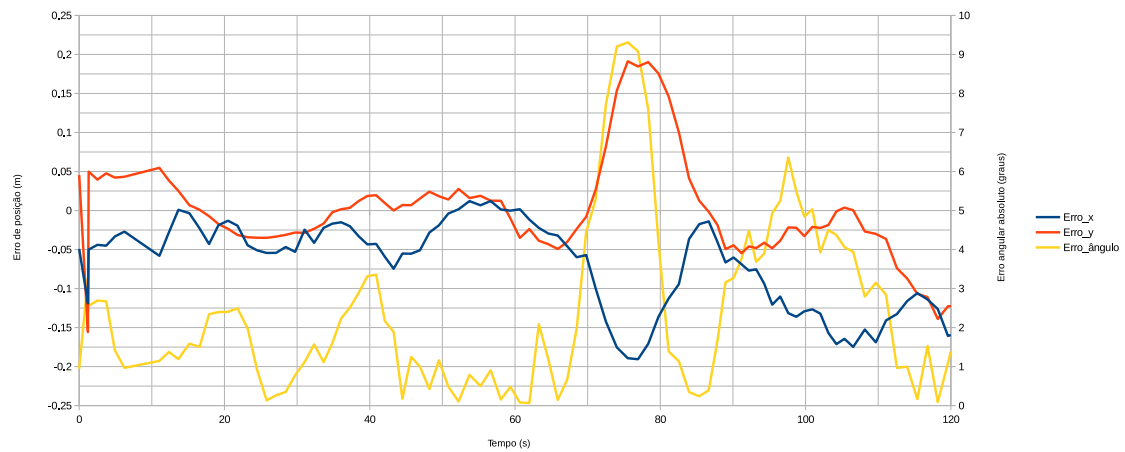


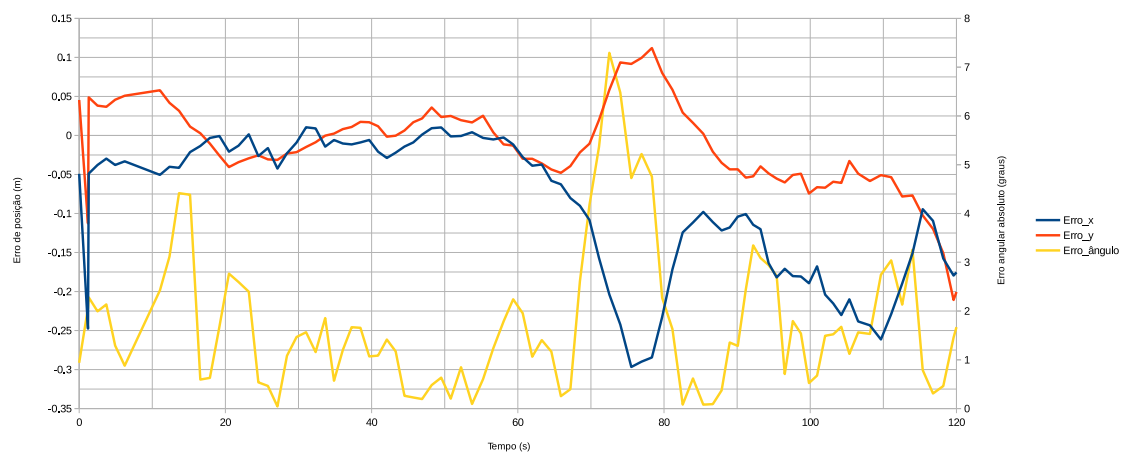
Figura 5.54: Sequência de imagens referentes à *Config4_OA_05L*: (a) e (b) Robô bem localizado, em relação ao *groundtruth* (seta verde) com alguma dispersão de partículas (setas vermelhas). (c) Devido aos *outliers* presentes no mapa, o robô ganha algum erro de posição (seta amarela) e formam-se duas concentrações principais de partículas, paralelas. (d) As partículas correspondentes à posição desajustada do robô começam a ser eliminadas e o robô volta a convergir para a posição correta.

Adicionalmente, ao atribuir-se o valor mais elevado ao parâmetro do modelo p_{hit} (*Config4_OA_05L*), os resultados obtidos têm um erro mais elevado (Tabela 5.24) comparativamente com as restantes configurações, devido à influência dos *outliers* nos dados fornecidos pelo laser. Nas Figuras 5.55 e 5.56 é possível identificar um momento, na fase final do percurso do robô, em que este é fortemente afetado pelos *outliers* presentes no mapa, levando a um aumento do erro nesse período. De forma complementar, na Figura 5.54 (c) e (d) verifica-se esse momento visualmente, em que as estimativas mantidas pelo filtro se dividem em dois grupos distintos, em que um deles

engloba a posição real do robô e outro está vários centímetros deslocado. Nesse momento a amostra com maior peso (probabilidade) ganhou algum erro em relação à posição do *groundtruth*, devido à influência dos *outliers* nos dados do laser.

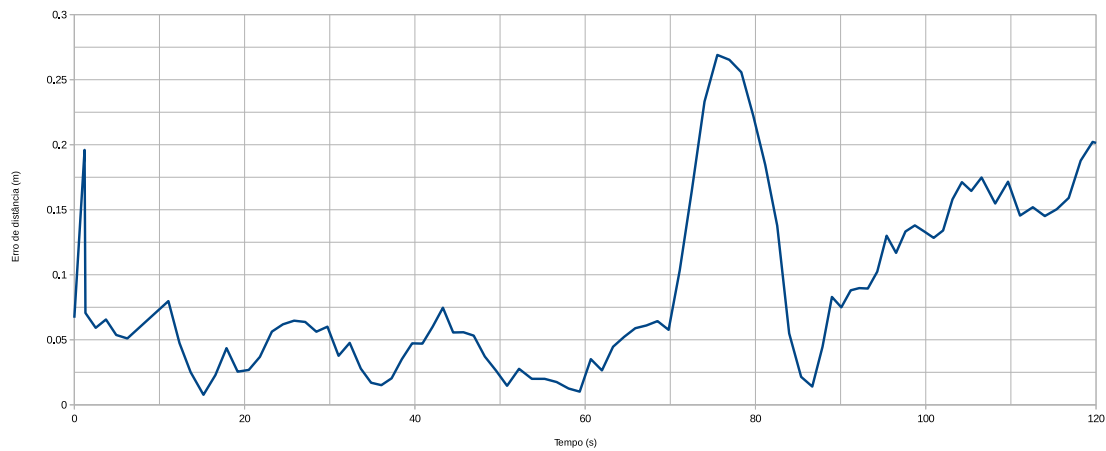


(a)

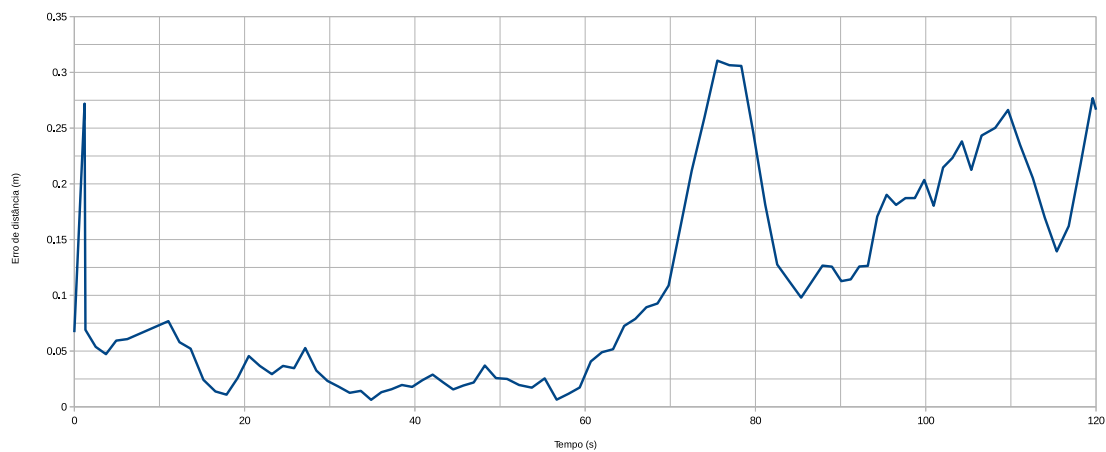


(b)

Figura 5.55: Evolução do erro em x, y e em ângulo, ao longo do percurso, utilizando a (a) *Config3_OA_05* e a (b) *Config4_OA_05L*.



(a)



(b)

Figura 5.56: Evolução do erro em distância, ao longo do percurso, utilizando a (a) *Config3_OA_05* e a (b) *Config4_OA_05L*.

Tabela 5.24: Resultados estatísticos das *Config3_OA_05* e *Config4_OA_05L*, num ambiente com grande quantidade de *outliers*.

	<i>Config3_OA_05</i>	<i>Config4_OA_05L</i>
Média do erro de distância (m)	0.0879	0.1071
Média do erro angular absoluto (°)	2.425	1.754
Erro máximo de distância (m)	0.2690	0.3105
Erro angular absoluto máximo (°)	9.309	7.294
Variância do erro da distância	$4.390 \cdot 10^{-3}$	$8.172 \cdot 10^{-3}$
Variância do erro angular	4.578	2.092

5.2.2.5 Resultados estatísticos em função do número de partículas

Com vista a estudar o efeito prático do número de partículas no algoritmo AMCL, em termos de precisão das suas estimativas e na robustez geral do filtro, utilizaram-se os mesmos ambientes de estudo e configurações anteriormente apresentadas, obtendo-se os resultados expostos nas Tabelas 5.25, 5.26, 5.27 e 5.28. Assim, utilizaram-se três quantidades diferentes para o número de partículas: 100, 200 e 300, tendo-se realizado uma análise a estes resultados na Secção 5.2.2.6.

Tabela 5.25: Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas para o ambiente sem *outliers*.

Configurações	Config1_OA_001			Config2_OA_02			Config3_OA_05		
Número de partículas	100	200	300	100	200	300	100	200	300
Média do erro de distância (m)	0.0429	0.0393	0.0363	0.0566	0.0507	0.0405	0.0523	0.0401	0.0438
Média do erro angular (°)	0.6592	0.5674	0.6556	0.8559	0.6607	0.6043	0.9767	0.8806	0.7640
Erro máximo de distância (m)	0.1003	0.1000	0.0995	0.1648	0.0938	0.1067	0.1856	0.1000	0.0854
Erro angular máximo (°)	3.668	3.661	3.708	3.673	2.729	2.955	3.783	3.551	3.215
Variância do erro da distância	0.0006	0.0003	0.0005	0.0014	0.0002	0.0004	0.0010	0.0004	0.0004
Variância do erro angular	0.6845	0.6317	0.6928	0.5274	0.3797	0.3711	0.6741	0.5944	0.3584

Tabela 5.26: Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas, com erros de odometria.

Configurações	Config2_OA_02			Config3_OA_05		
Número de partículas	100	200	300	100	200	300
Média do erro de distância	0.1061	0.1108	0.0875	0.1633	0.1021	0.0758
Média do erro angular	1.265	2.976	1.370	3.022	2.406	1.339
Erro máximo de distância	3.239	4.002	0.5119	7.657	4.334	0.5317
Erro angular máximo	13.66	170.1	12.48	166.1	113.6	8.967
Variância do erro da distância	0.0821	0.1204	0.0132	0.4160	0.1350	0.0119
Variância do erro angular	2.564	205.5	4.609	192.0	89.94	2.600

Tabela 5.27: Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas para o ambiente com poucos *outliers*.

Configurações	Config1_OA_001			Config2_OA_02		
Número de partículas	100	200	300	100	200	300
Média do erro de distância (m)	0.0369	0.0311	0.0338	0.0381	0.0334	0.0351
Média do erro angular (°)	0.4080	0.4996	0.5214	0.9169	0.7292	0.5029
Erro máximo de distância (m)	0.0607	0.0601	0.0824	0.0730	0.0645	0.0962
Erro angular máximo (°)	1.356	1.495	1.642	3.296	2.521	1.856
Variância do erro da distância	0.0001	0.0002	0.0002	0.0003	0.0002	0.0003
Variância do erro angular	0.1097	0.1470	0.1753	0.5329	0.2821	0.1461

Tabela 5.28: Comparação estatística do desempenho do AMCL em função do número de partículas utilizadas para o ambiente com muitos *outliers*.

Configurações	Config1_OA_001			Config2_OA_02			Config3_OA_05		
Número de partículas	100	200	300	100	200	300	100	200	300
Média do erro de distância (m)	0.0402	0.0365	0.0386	0.0602	0.0445	0.0696	0.1760	0.1109	0.0879
Média do erro angular (°)	0.5782	0.5019	0.5488	1.284	0.8197	0.7069	2.094	2.361	2.425
Erro máximo de distância (m)	0.0811	0.0778	0.0766	0.1290	0.1540	0.1925	0.5439	0.3094	0.2690
Erro angular máximo (°)	1.455	1.574	1.708	4.021	2.646	2.018	8.536	6.392	9.309
Variância do erro da distância	0.0002	0.0002	0.0002	0.0008	0.0008	0.0025	0.0139	0.0058	0.0044
Variância do erro angular	0.1162	0.1585	0.1438	0.7379	0.2988	0.2164	3.701	3.215	4.578

5.2.2.6 Conclusões

Após ter-se analisado várias configurações aplicadas em ambientes com diferentes características, o maior destaque passa pela boa parametrização dos coeficientes para o modelo da odometria, os quais revelam ter um grande impacto no desempenho geral do filtro em diversas circunstâncias. A utilização de coeficientes elevados para este modelo aumenta o erro de localização mas simultaneamente torna o algoritmo mais robusto a perturbações inesperadas, permitindo a sua recuperação.

Os parâmetros referentes ao laser, por si só, não têm muita influência na robustez ou precisão do filtro, devendo atribuir-se uma maior percentagem ao fator z_{hit} , em oposição ao parâmetro z_{rand} , que apenas distribui uniformemente os pesos das partículas.

Quanto à influência do número de partículas, é de notar que os resultados obtidos com um número de partículas superior são geralmente melhores do que com um número mais reduzido de partículas, sendo que as experiências realizadas com 100 partículas revelaram sempre piores resultados do que com 200 ou 300. Para as condições deste projeto, devido às limitações do computador presente no robô, o máximo número de partículas considerado viável é 300, para ser possível manter um ciclo de execução dos programas suficientemente baixo (ver 5.3).

A precisão de cada configuração em função do número de partículas é mais visível num ambiente onde não existem obstáculos que perturbem os dados obtidos pelo robô, como é o caso da situação onde não existem *outliers*. Nessas circunstâncias, é claro, pelos dados apresentados na Tabela 5.25, que à medida que há um aumento do número de partículas a estimativa de localização também melhora. Esta é uma situação ideal que na prática raramente acontece. Por outro lado, em alguns exemplos onde houveram perturbações externas ao sistema (aumento do número de *outliers*, erros de odometria forçados), as configurações utilizando 200 partículas revelaram resultados ligeiramente melhores em alguns campos de análise (*e.g.*, erro médio de posição) do que com 300, não querendo dizer que esta diminuição no número de partículas seja mais vantajosa nestas circunstâncias. O algoritmo AMCL é um algoritmo de localização probabilístico onde entram processos de geração de valores de forma aleatória (*e.g.*, na dispersão das partículas ao longo da movimentação do robô, de acordo com as covariâncias especificadas no modelo da odometria), sendo que, em alguns casos, a presença de partículas em posições específicas levam a uma mais rápida convergência e diminuição do erro na movimentação do robô. Quanto maior o número de partículas presentes no filtro, maior a probabilidade de existirem amostras nos locais "corretos", levando a um melhor seguimento de posição e a um menor erro. Por outro lado, na fase de seguimento de posição, depois do filtro convergir para uma posição correta, o número de partículas utilizado pode ser reduzido, não perdendo muita precisão na estimação da posição. Na verdade, uma das extensões do algoritmo AMCL, que adapta o número de partículas do filtro de acordo com a fase atual robô (*e.g.*, aumenta o número de partículas numa fase de localização global ou recuperação e diminui esse número durante o seguimento de posição), é baseada exatamente nesse princípio. Para o conjunto de testes realizados, é relevante que as condições de comparação de cada configuração fossem o mais constantes e previsíveis possível, com vista a colocar todas em

pé de igualdade, não se tendo utilizado um número variável de partículas, que no entanto pode ser vantajoso em aplicações práticas.

Deste modo, pode concluir-se, com base nas experiências efetuadas, que a utilização de um número próximo ou menor que 100 partículas torna o sistema menos preciso e menos robusto (*e.g.*, no caso da necessidade de recuperação de alguma situação inesperada, não havendo número suficiente de partículas para esse efeito), podendo definir-se este como o limite mínimo.

5.3 Comparação entre o AMCL e o Perfect Match

Nesta secção irá realizar-se uma análise comparativa entre os dois algoritmos de localização que têm sido explorados até ao momento: AMCL e Perfect Match. Depois das comparações efetuadas entre cada algoritmo, alterando alguns dos parâmetros que têm mais impacto no desempenho dos mesmos, selecionou-se a configuração que devolveu melhores resultados para cada um e comparou-se os dois algoritmos, utilizando essas parametrizações. Mantiveram-se as 4 situações já apresentadas na Secção 5.1.3 como ambientes de teste para esta análise.

Um dos dados iniciais relevantes para este estudo passa pelo tempo de processamento necessário para a execução dos programas, o que para computadores com processadores limitados, como acontece neste projeto, torna-se um factor crucial para a sua aplicação em tempo real. Assim, realizaram-se alguns testes no robô Clever, com vista a quantificar e analisar a evolução dos tempos de ciclo dos dois algoritmos de localização.

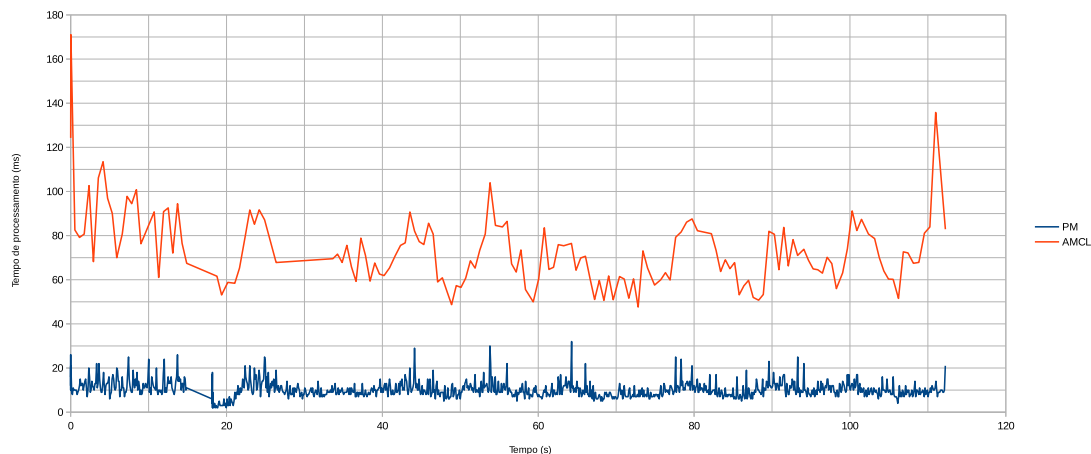


Figura 5.57: Evolução do tempo de ciclo para a execução do programa AMCL, com 300 partículas em comparação com o programa Perfect Match.

Tabela 5.29: Tempos de ciclo médio, máximo e mínimo para o AMCL (com 300 partículas) e para o Perfect Match.

	AMCL	PM
Número de partículas	300	-
Média do tempo de ciclo (ms)	73.0790	10.2747
Tempo máximo (ms)	171	32
Tempo mínimo (ms)	48	2

Tabela 5.30: Tempos de ciclo médio, máximo e mínimo para o AMCL (com 200 partículas) e para o Perfect Match.

	AMCL	PM
Número de partículas	200	-
Média do tempo de ciclo (ms)	50.25	11.97
Tempo máximo (ms)	93	61
Tempo mínimo (ms)	26	5

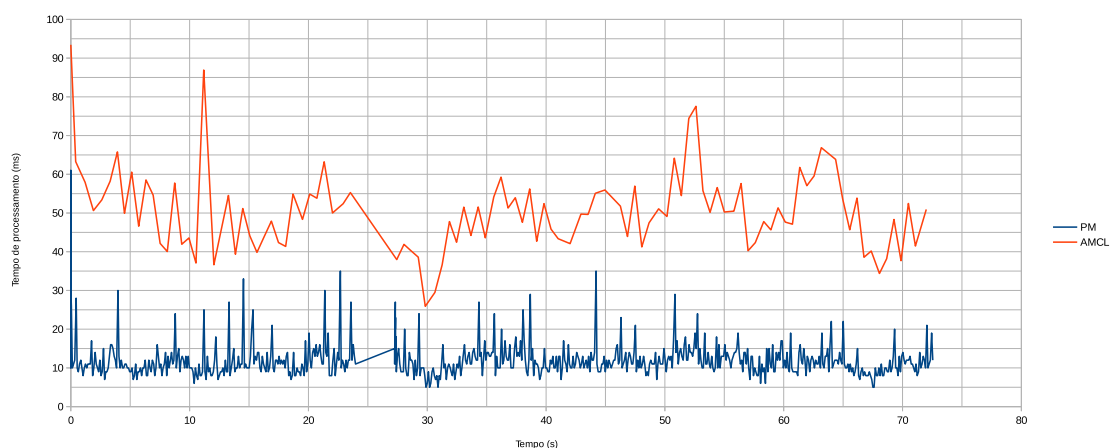


Figura 5.58: Evolução do tempo de ciclo para a execução do programa AMCL, com 200 partículas em comparação com o programa Perfect Match.

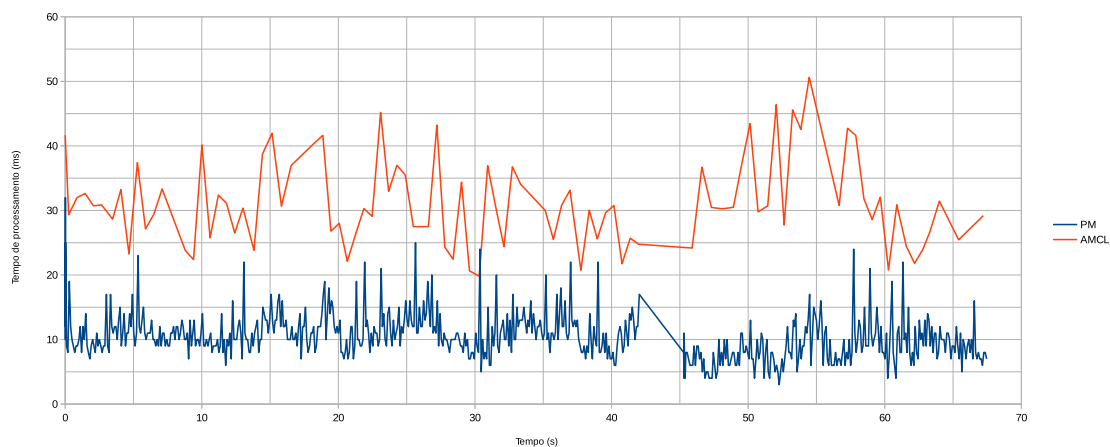


Figura 5.59: Evolução do tempo de ciclo para a execução do programa AMCL, com 100 partículas em comparação com o programa Perfect Match.

Tabela 5.31: Tempos de ciclo médio, máximo e mínimo para o AMCL (com 100 partículas) e para o Perfect Match.

	AMCL	PM
Número de partículas	100	-
Média do tempo de ciclo (ms)	31.18	10.45
Tempo máximo (ms)	51	32
Tempo mínimo (ms)	20	3

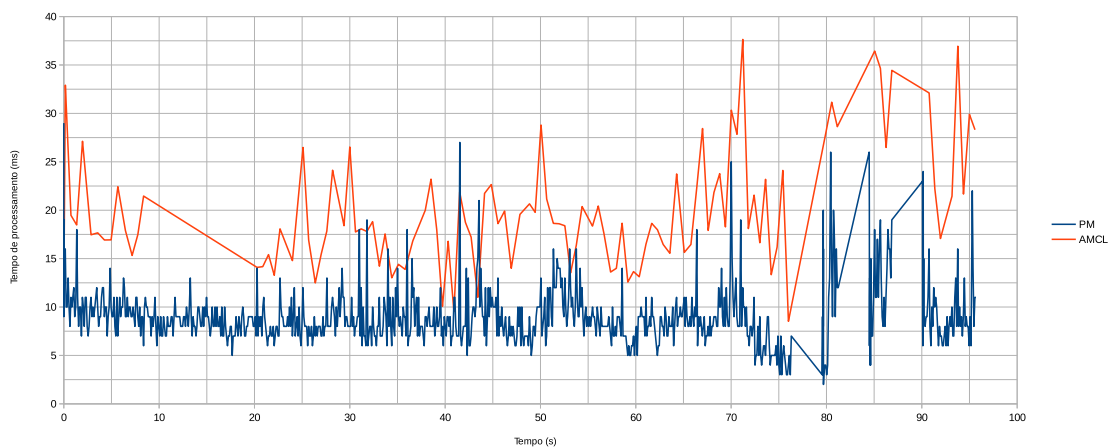


Figura 5.60: Evolução do tempo de ciclo para a execução do programa AMCL, com 50 partículas em comparação com o programa Perfect Match.

Os dados apresentados nas figuras e tabelas anteriores foram obtidos exactamente nas mesmas condições tanto para o AMCL como para o Perfect Match, ao serem executados em paralelo durante a movimentação do robô. De acordo com estes valores, para que o AMCL atinja a leveza computacional do Perfect Match, teriam de se utilizar menos de 50 partículas, o que seria um valor demasiado baixo para este algoritmo, comprometendo muito o seu desempenho. Por outro lado, a operação do filtro utilizando 300 partículas tem tempos de ciclo bastante elevados (com uma média de 73 ms e chegando aos 173 ms - Tabela 5.29), tornando-se no limite máximo viável para as aplicações pensadas para este projeto, em que o robô pode atingir velocidades na ordem dos 0.5 m/s.

Tabela 5.32: Tempos de ciclo médio, máximo e mínimo para o AMCL (com 50 partículas) e para o Perfect Match.

	AMCL	PM
Número de partículas	50	-
Média do tempo de ciclo (ms)	19.84 ms	8.895 ms
Tempo máximo (ms)	38 ms	29 ms
Tempo mínimo (ms)	9 ms	2 ms

Quanto à análise da precisão e robustez dos dois modelos de localização, para as mesmas circunstâncias e ambientes das anteriores comparações, obtiveram-se os resultados que serão apresentados e discutidos de seguida. É importante referir que ambos os algoritmos foram executados em simultâneo, ao longo do percurso do robô, para que os resultados fossem os mais justos possível, sem beneficiar ou prejudicar algum dos dois.

Assim, para a sua comparação utilizaram-se as melhores parametrizações de cada um dos algoritmos:

- Para o AMCL utilizou-se a [Config2_OA_02](#), com 300 partículas.
- Para o Perfect Match utilizou-se a [Config5_OF_F1F2WD](#).

Ambiente sem outliers

Para as circunstâncias em que não haviam *outliers* ao longo do percurso do robô, obtiveram-se os resultados apresentados nas Figuras 5.61 e 5.62. É possível verificar que tanto os erros de distância como os erros angulares, em relação ao *groundtruth*, tomam valores próximos ao longo do percurso, sendo que, em média, o AMCL apresenta erros menores do que o Perfect Match, para um ambiente sem *outliers* (Tabela 5.33).

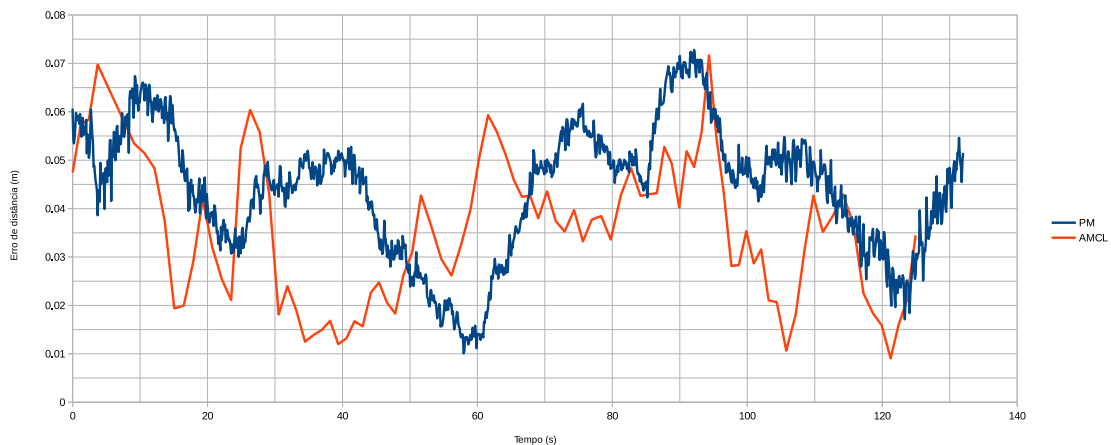


Figura 5.61: Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente sem obstáculos.

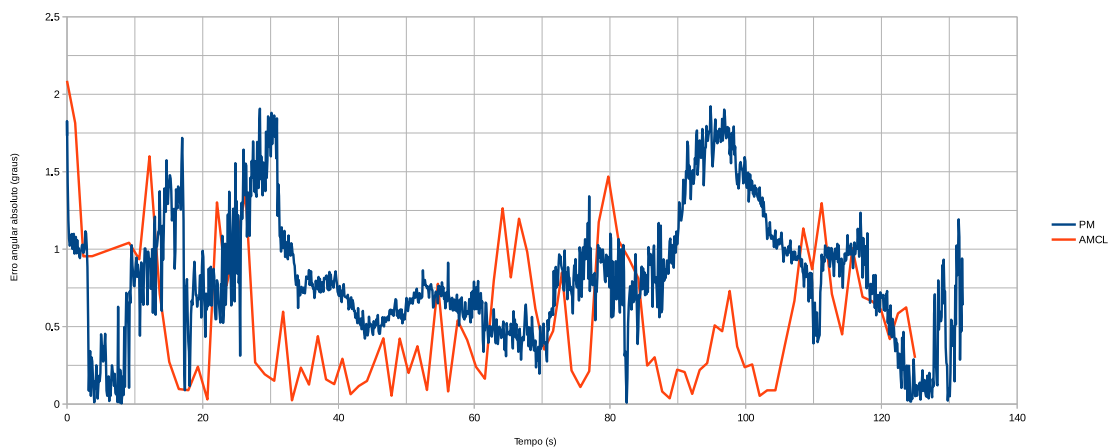


Figura 5.62: Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente sem obstáculos.

Tabela 5.33: Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), num ambiente livre de obstáculos.

	AMCL	PM
Média do erro de distância (m)	0.0352	0.0441
Média do erro angular absoluto (°)	0.5372	0.8221
Erro máximo de distância (m)	0.0716	0.0727
Erro angular absoluto máximo (°)	2.087	1.922
Variância do erro da distância	$2.123 \cdot 10^{-4}$	$1.749 \cdot 10^{-4}$
Variância do erro angular	0.1985	0.1662

Ambiente com alguns *outliers*

Nas Figuras 5.63 e 5.64 estão representados os erros ao longo do tempo, num ambiente onde existiam obstáculos que iam sendo detetados pelo laser em algumas zonas do percurso. Ao observar a Figura 5.63, é possível identificar um momento em que o AMCL ganhou algum erro de posição, numa porção do percurso onde se encontravam alguns objetos externos ao mapa. Pelo contrário, o Perfect Match não foi afetado por estes obstáculos, devido ao filtros de rejeição de *outliers* em que esse pontos foram descartados e não entraram na execução do algoritmo. Mesmo assim, nos locais em que ambos os modelos estavam bem localizados em relação ao *groundtruth*, com menor influência dos *outliers*, o AMCL revelou uma maior precisão, mantendo um erro ligeiramente inferior ao do Perfect Match. Para além disso, as médias do erro de distância e de ângulo também são inferiores aos do Perfect Match, apesar de ter atingido um erro máximo superior (Tabela 5.34).

Tabela 5.34: Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), num ambiente com alguns *outliers*.

	AMCL	PM
Média do erro de distância (m)	0.0345	0.0349
Média do erro angular absoluto (°)	0.5874	0.8160
Erro máximo de distância (m)	0.0809	0.0579
Erro angular absoluto máximo (°)	1.897	3.468
Variância do erro da distância	$2.095 \cdot 10^{-4}$	$9.4668 \cdot 10^{-5}$
Variância do erro angular	0.2048	0.3791

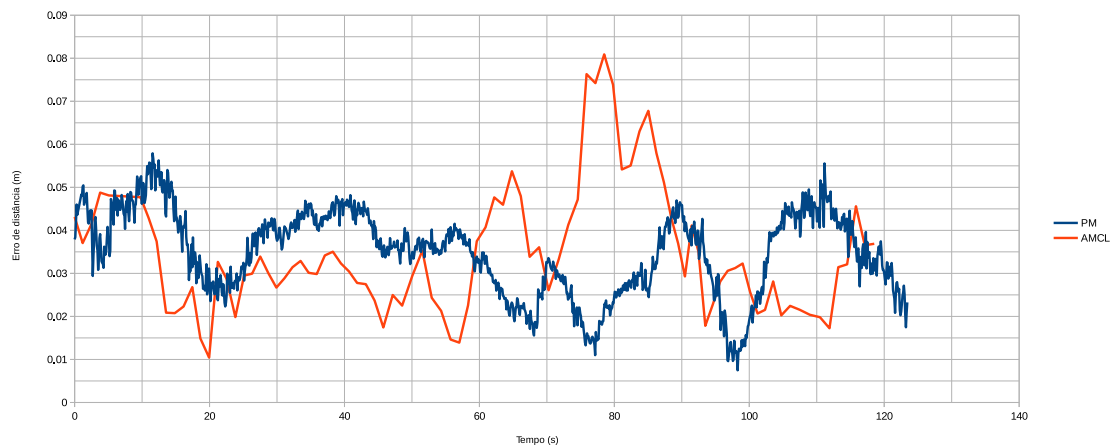


Figura 5.63: Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com alguns objetos externos ao mapa.

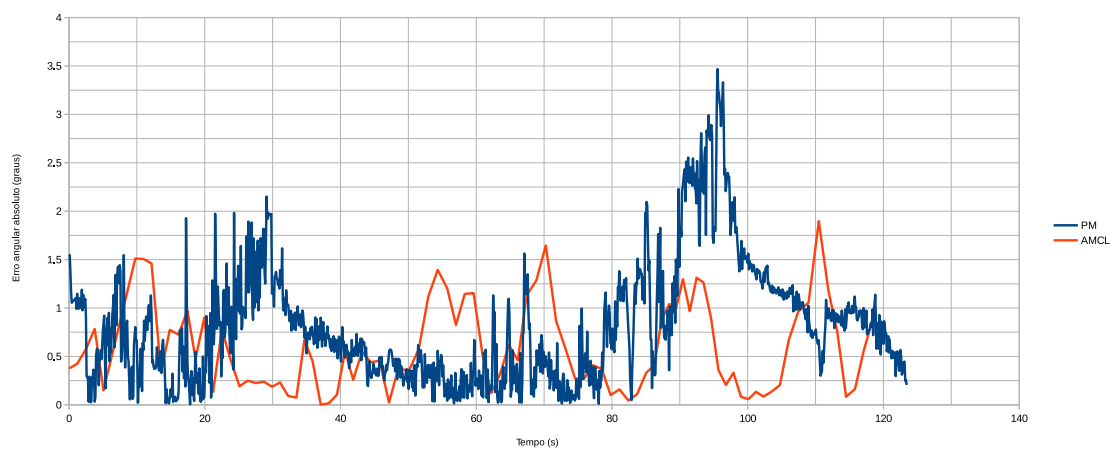


Figura 5.64: Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com alguns objetos externos ao mapa.

Ambiente com grande quantidade de *outliers*

No caso do ambiente onde havia um elevado número de objetos colocados especificamente para interferir com as medidas do laser do robô, obtiveram-se os resultados das Figuras 5.66, 5.67 e da Tabela 5.35. Para este exemplo, é notório que ambos os algoritmos foram afetados pela grande quantidade de *outliers* mantendo erros superiores aos casos anteriores e atingindo valores elevados na zona final do percurso do robô. Mais uma vez, nos locais onde as estimativas mantinham um erro baixo em relação ao *groundtruth* o AMCL possuía erros inferiores ao do Perfect Match. No campo dos erros angulares, o Perfect Match demonstrou maior inconsistência e grandes oscilações em algumas partes do percurso, com o AMCL a manter erros inferiores a 2 graus.

Na Figura 5.65 pode visualizar-se, nos momentos finais da trajetória do robô, um aumento do erro na estimação de ambos os algoritmos de localização, em que é perceptível a existência de dois "blocos" de partículas por parte do AMCL, causadas pelos *outliers* (pontos cor de rosa). Neste caso, o AMCL tem duas zonas em que considera haver maior probabilidade do robô se encontrar, em que a partícula com maior peso (seta amarela) ainda está na concentração de partículas referentes à zona da localização real do robô. Apesar de não se ter verificado para este exemplo, em situações deste tipo, é comum que a estimativa de localização (partícula com maior peso) oscile de uma concentração para a outra, dependendo dos dados sensoriais recebidos pelo robô. Ao haver um "salto" na estimativa de localização dada pelo AMCL, se se utilizar um modelo de supervisão, é possível detetar esta oscilação a partir do cálculo das velocidades linear e angular do robô inferidas a partir das informações devolvidas pelo AMCL. Deste modo, consegue-se detetar situações anómalas no sistema de localização e perceber que o robô pode estar na eminência de perda, agindo de forma a contrariar essa tendência.

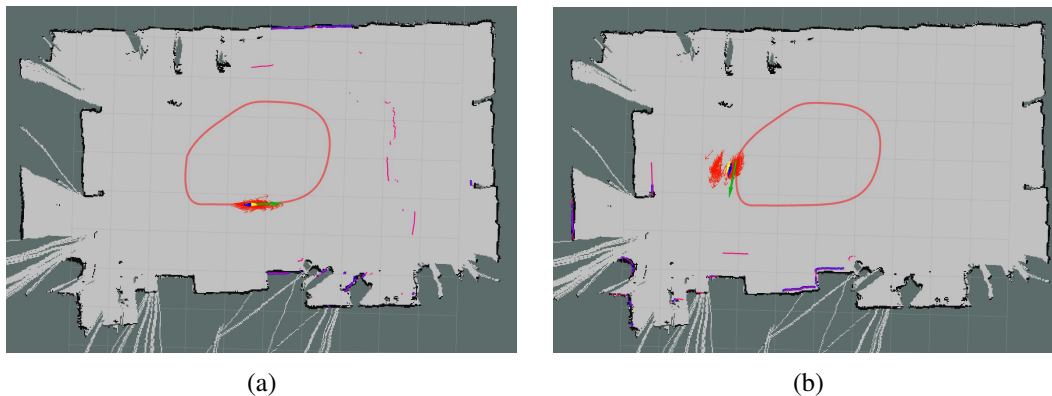


Figura 5.65: (a) Boas estimativas de posição, dadas pelo algoritmo AMCL (a seta amarela corresponde à partícula com maior peso e as restantes estão representadas com setas vermelhas) e pelo Perfect Match (seta azul), em relação ao *groundtruth* (seta verde) (b) Aumento do erro das estimativas de localização, na fase final do percurso do robô.

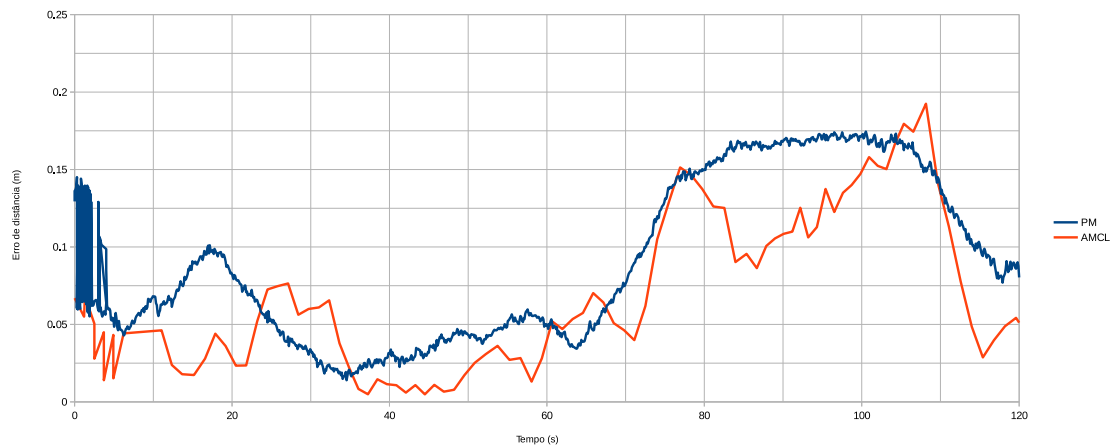


Figura 5.66: Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com muitos obstáculos.

Tabela 5.35: Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), num ambiente com grande quantidade de *outliers*.

	AMCL	PM
Média do erro de distância (m)	0.0680	0.0905
Média do erro angular absoluto (°)	0.6944	2.264
Erro máximo de distância (m)	0.1925	0.1745
Erro angular absoluto máximo (°)	2.018	15.96
Variância do erro da distância	$2.461 \cdot 10^{-3}$	$2.669 \cdot 10^{-3}$
Variância do erro angular	0.2160	5.829

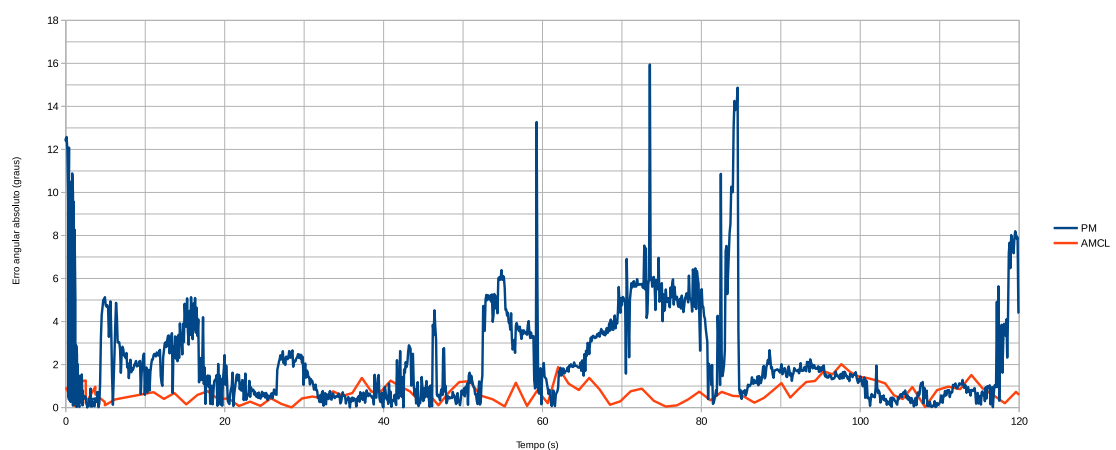


Figura 5.67: Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô num ambiente com muitos obstáculos.

Erros forçados de odometria

Em resposta a perturbações externas, causando erros nos valores de odometria, ambos os algoritmos revelaram a capacidade para recuperar desta situação, convergindo de novo para a solução correta. Na Figura 5.69 é possível verificar que o Perfect Match recuperou, das duas vezes, mais rapidamente que o AMCL, tendo também atingido valores inferiores no erro máximo de distância e ângulo em relação à posição correta (Tabela 5.36). Para este exemplo prático, o AMCL demonstrou ser mais sensível a perturbações na odometria do robô, atingindo valores muito elevados no erro angular (Figura 5.70), para além do que já foi referido.

Na Figura 5.68 estão representadas sequências de imagens que descrevem a evolução nos comportamentos dos algoritmos de localização, havendo dois momentos em que há um aumento do erro (5.68b e 5.68c), devido a perturbações externas, conseguindo ambos recuperar e manter uma boa estimativa de localização.

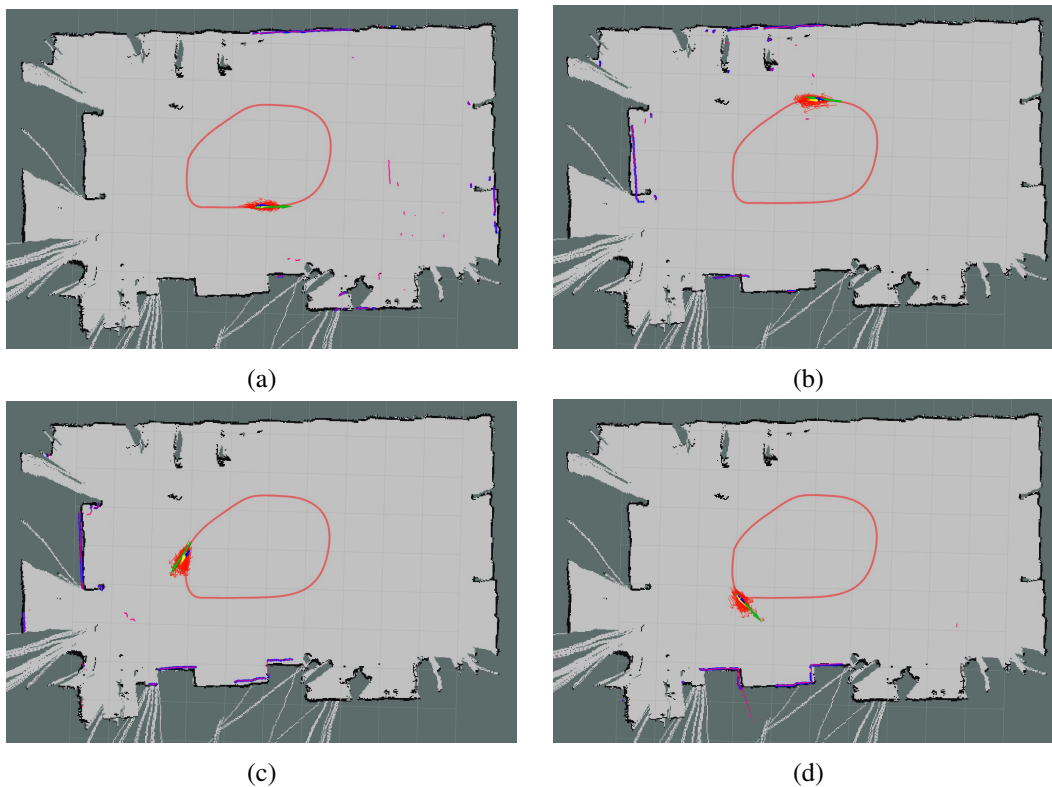


Figura 5.68: (a) Robô inicialmente bem localizado, tanto pela estimativa do AMCL (seta amarela), como pela estimativa do Perfect Match (seta azul), em relação ao *groundtruth* (seta verde). (b) Primeiro local onde se forçou o erro de odometria, com o conseqüente aumento do erro por parte dos dois algoritmos. (c) Segunda zona onde a odometria devolveu valores errados, aumentando de novo os erros da estimativas, após primeira recuperação. (d) Ambos os algoritmos de localização conseguiram voltar a recuperar a posição correta.

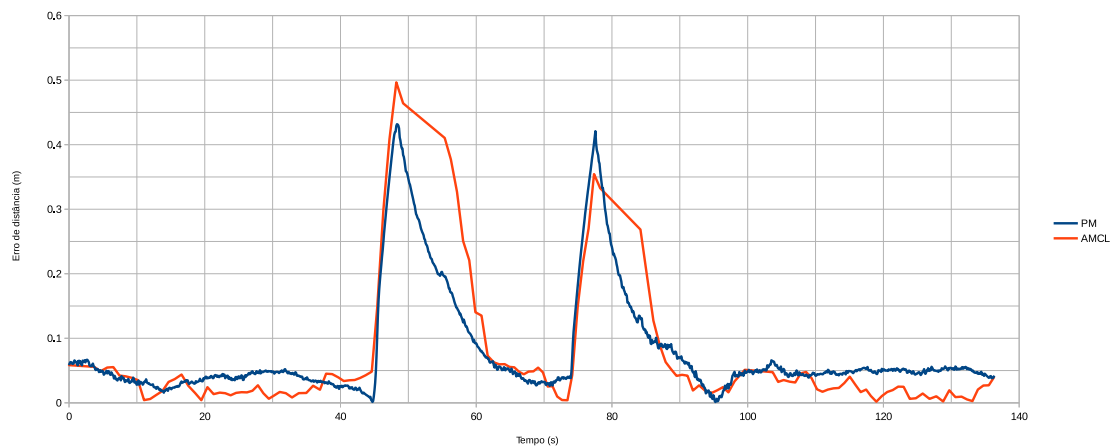


Figura 5.69: Evolução do erro de distância das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô com dois momentos de erros forçados de odometria.

Tabela 5.36: Resultados estatísticos das estimativas do AMCL e do Perfect Match (PM), perante erros forçados de odometria.

	AMCL	PM
Média do erro de distância (m)	0.0645	0.0800
Média do erro angular absoluto (°)	1.221	1.061
Erro máximo de distância (m)	0.4966	0.4318
Erro angular absoluto máximo (°)	12.06	4.509
Variância do erro da distância	$9.679 \cdot 10^{-3}$	$7.479 \cdot 10^{-3}$
Variância do erro angular	2.961	0.5822

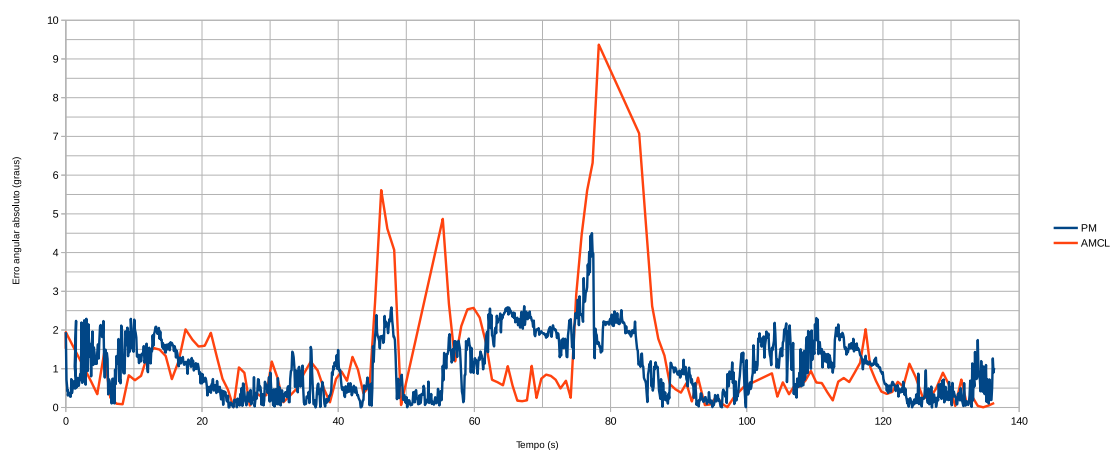


Figura 5.70: Evolução do erro angular das estimativas calculadas pelo AMCL (vermelho) e pelo Perfect Match (azul), ao longo do percurso do robô com dois momentos de erros forçados de odometria.

5.3.1 Conclusões

Nesta secção apresentaram-se os resultados da comparação prática entre o AMCL e o Perfect Match, para um caso particular e exemplos concretos (utilizando um determinado laser, a uma baixa altura e tendo em mente as restrições de processamento do computador onde estes algoritmos vão ser executados), não havendo intenção de generalizar as conclusões para outro tipo de situações ou aplicações.

Inicialmente é importante referir que cada um destes algoritmos é baseado em princípios matemáticos diferentes e que a sua comparação nem sempre é um processo linear. O AMCL é um modelo multi-hipóteses, calculando um conjunto de estimativas (partículas), atribuindo-lhes pesos (probabilidades) e destacando a que possuiu maior peso como sendo a posição atual mais provável do robô. Este facto, apesar das vantagens conhecidas em termos de robustez e eficácia na localização, tem a desvantagem de tornar o algoritmo mais pesado e mais lento computacionalmente, à medida que o número de estimativas calculadas aumenta. A odometria tem um papel muito importante no filtro, sendo responsável pela propagação da posição do robô, numa primeira fase, antes da atribuição dos pesos às partículas na fase sensorial do modelo. Deste modo, havendo falhas de odometria não modeladas (*e.g.*, ao utilizar valores baixos nos coeficientes dados ao modelo da odometria) pode levar à perda de localização, mesmo utilizando um número elevado de partículas. Isto porque, apesar do filtro utilizar os dados sensoriais do laser, estes são utilizados apenas para atribuir um peso a cada uma das partículas, sendo que, no caso de não haver amostras perto da posição correta do robô, estes dados não contribuem para a recuperação do robô. É importante referir que num filtro de partículas é possível modelar qualquer distribuição de probabilidade referente às condições e características esperadas, tanto para o ambiente de operação do robô, como para as suas próprias características intrínsecas. Ainda assim, na prática é muito complicado modelar todo o tipo de perturbações e características, tanto pelo crescente aumento na complexidade matemática que isso implica, como pela impossibilidade de prever todos os tipos de situações que possam ocorrer.

O Perfect Match é um algoritmo que apenas assume e estima uma posição do robô, baseando-se num método de ajuste de mapa e consequentemente dependendo muito dos dados fornecidos pelo laser (ou da sua privação) e das características (forma) do mapa. Assim, há uma maior sensibilidade à existência de objetos inesperados no mapa, o que, mesmo utilizando os filtros de rejeição de *outliers* (que provaram melhorar muito o algoritmo, aumentando a sua robustez), têm limitações quanto ao número de pontos que restam para executar o ajuste de mapa. Por outro lado, a integração da odometria no Perfect Match tem essencialmente o objetivo de impedir que as estimativas de posição devolvidas pelo algoritmo oscilem ou "saltem" de uns locais para os outros (não sendo modelada, como acontece com o AMCL), permitindo um seguimento eficaz de posição.

Um facto já conhecido é a leveza e rapidez computacional do Perfect Match, que consegue atingir tempos de ciclo várias vezes inferiores aos do AMCL, mesmo utilizando um número não muito elevado de partículas. Para um processador com pouco poder de cálculo, como é o caso

do computador presente no robô Clever (ver Secção 3.1.8), o Perfect Match manteve tempos de ciclo na ordem dos 10 ms, permitindo colocar o robô a movimentar-se a velocidades elevadas sem colocar em risco o seu controlo em tempo real. Em situações ainda mais extremas, como é exemplo um projeto para desenvolvimento de uma cadeira de rodas inteligente [49, 50, 51, 52, 53], onde o autor deste trabalho está a iniciar a colaboração, em que o centro de processamento do sistema será realizado por um Raspberry Pi, o poder de cálculo é ainda mais limitado e a utilização do algoritmo AMCL não será viável por esse motivo.

Assim, com base nos resultados experimentais obtidos, os dois modelos revelaram-se capazes de operar nos diversos tipos de ambientes adversos nos quais foram testados. Em geral, ambos os algoritmos, para estas configurações, mantiveram erros de estimativa de posição muito similares e da mesma ordem de grandeza (de apenas alguns centímetros) nos momentos em que estavam bem localizados (próximos da posição dada pelo *groundtruth*, excluindo as fases em que houve um aumento do erro devido a perturbações externas).

Por outro lado, o Perfect Match, no caso de falhas na odometria revelou resultados superiores e uma maior robustez em comparação com o AMCL, recuperando mais rapidamente e não sendo tão susceptível às mesmas.

Como já foi comentado, e foi verificado nos testes, nas situações em que foram colocados alguns *outliers* no mapa de operação do robô, o AMCL demonstrou uma maior imunidade (e consequente robustez) aos dados perturbados do laser, principalmente devido à sua característica de multi-hipóteses.

Concluindo, ambos os algoritmos são opções viáveis para o processo de auto-localização de robôs móveis, cada um com os seus pontos fortes e fracos, baseando-se em diferentes paradigmas.

5.4 Supervisão dos algoritmos de localização

Como já foi referido anteriormente, as características da estrutura e do *hardware* do robô impõem um conjunto de limitações e dificuldades ao processo de auto-localização. Sendo esta a base e uma das partes mais fundamentais na movimentação do robô, é de extrema importância que este seja capaz de o fazer de forma robusta e fiável.

Indo mais além, com vista a evitar possíveis perdas de localização ou situações anómalas durante a navegação do robô, desenvolveu-se um programa de supervisão e integração de dois ou mais algoritmos de localização. De uma forma geral, a ideia passa por haver dois métodos de auto-localização distintos a serem executados, em paralelo, no robô. Cada um destes algoritmos devolve a posição estimada num determinado instante de tempo. O programa de supervisão recebe cada uma dessas estimativas, calcula as velocidades linear e angulares através da diferença de posição num determinado período de tempo e compara-as entre si, juntamente com outros métodos de localização relativa (*e.g.*, odometria, IMUs). Ao dispor de mais do que um processo de localização e dos dados fornecidos pela odometria/IMU, é possível comparar e validar as velocidades (linear e angular) calculadas periodicamente entre eles, detectando possíveis erros de localização ou "saltos" na posição estimada por algum algoritmo. Por exemplo, se um dos métodos, devido a perturbações no ambiente (*e.g.*, passagem de um grupo de pessoas à frente do laser) devolveu uma posição errada uns cms mais deslocados do que seria de esperar, é possível detectar essa variação anormal da posição momentânea, através da comparação com outros processos de localização (*e.g.*, odometria, IMU, outro algoritmo de localização – Perfect Match, AMCL) e corrigi-la, evitando o aumento do erro ou o início da sua perda de localização.

Nos testes realizados utilizou-se o AMCL e o Perfect Match, dois algoritmos de localização global em paralelo com os dados da odometria, que fornece uma posição relativa e incremental. Assim, pode realizar-se um "desempate" quanto à concordância dos dados devolvidos por cada processo, comparando-os dois a dois e assim detectar qual dos três algoritmos falhou. Neste caso, como o Perfect Match é um algoritmo de localização muito leve computacionalmente e o AMCL é quanto mais pesado quanto maior o número de partículas definidas para o algoritmo, é possível manter este sistema com bons tempos de ciclo e com pouco *overhead*, compensando o menor número de partículas do AMCL com a integração de outro algoritmo de localização leve.

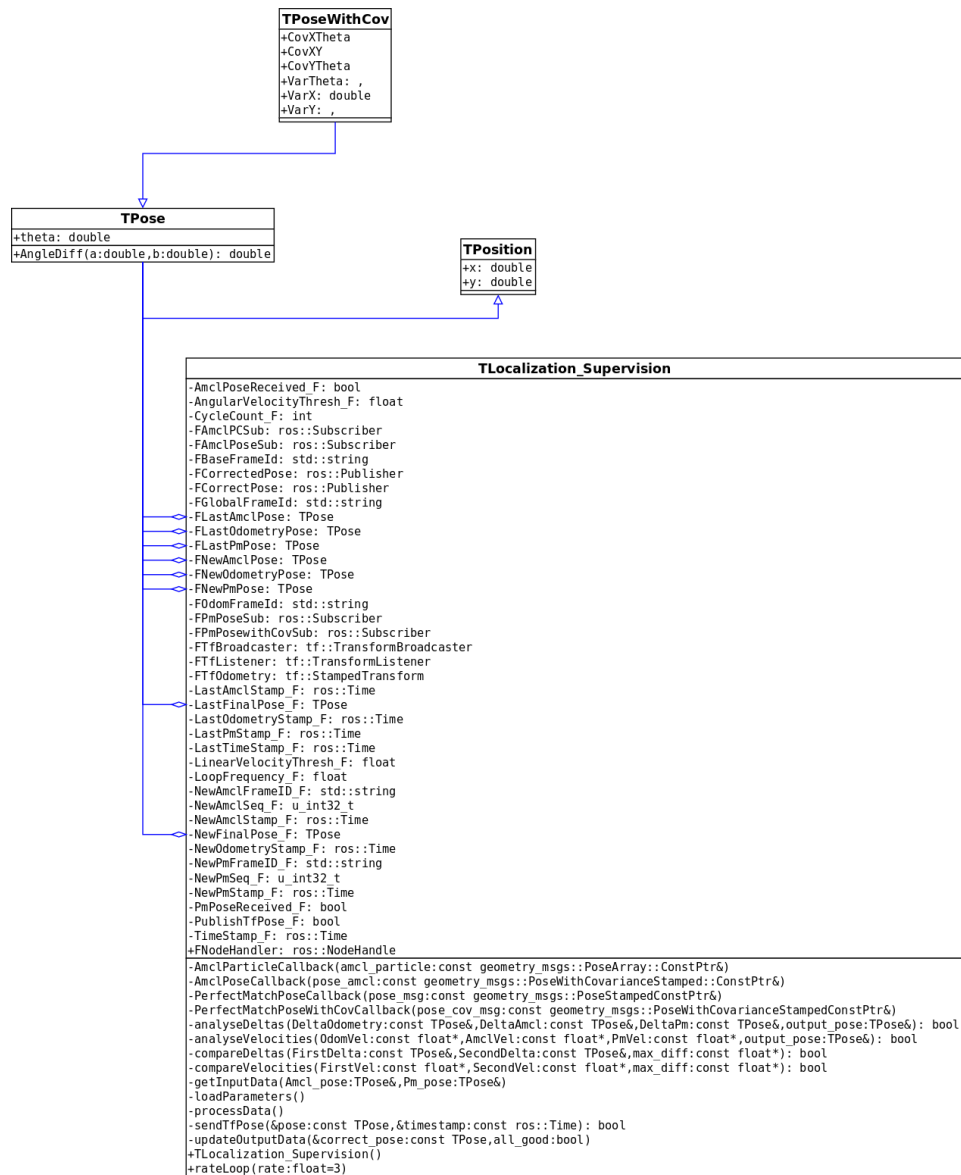


Figura 5.71: Diagrama de classes UML do programa supervisor dos algoritmos de localização, desenvolvido em C++ e aplicado em ROS.

Capítulo 6

Conclusão

O projeto da presente dissertação focou-se essencialmente na integração de um sistema que permitisse uma operação autónoma, robusta e fiável para o CleverRobot ou qualquer outro robô cujos dados de localização tenham perturbações frequentes (robôs baixos, como por exemplo uma cadeira de rodas autónoma) e com baixo poder de cálculo, sendo que, o maior desafio e o ponto mais crucial deste sistema passa por conseguir uma auto-localização consistente e precisa. Assim, realizaram-se análises comparativas de forma a perceber e otimizar os algoritmos de localização mais promissores para este tipo de aplicações: AMCL e Perfect Match, cujos resultados foram apresentados no Capítulo 5.

As análises aos algoritmos de localização, para além do que já foi referido, tinham também o objetivo de estudar os comportamentos e reações do AMCL e do Perfect Match, em função de um número de variáveis (*e.g.*, diferentes parâmetros, diferentes ambientes, resposta a falhas, etc), com vista à melhor exploração das suas características, para uma futura e eficaz integração conjunta. A ideia da utilização destes dois algoritmos, em paralelo, durante a operação do robô, promete um aumento da robustez do sistema a ambientes adversos ou perturbações externas inesperadas. Por isso, explorou-se a ideia de um sistema de supervisão destes métodos de localização em conjunto com odometria e IMUs, com o objetivo de integrá-los e validá-los entre si, aproveitando os seus pontos fortes e fracos, detetando falhas ou perdas de localização.

Numa outra vertente, a criação de um sistema de troca automática de baterias foi outro dos focos deste projeto, em que se desenvolveram algumas ferramentas como formas de aproximação a esse objetivo (programa de controlo dos servomotores, em ROS, e um mecanismo de encaixe e elevação das caixas de baterias). A estação de troca de baterias, devido à necessidade de ter alguma complexidade (ver Capítulo 4), não ficou pronta a tempo de ser utilizada ou de se realizarem testes à mesma. Mesmo assim, o projeto e as ideias para as suas características já estão definidas.

Por fim, o sistema robótico ficou integrado, tanto ao nível de *hardware* como ao nível de *software* e a funcionar de forma autónoma, tendo sido testado para uma das aplicações projetadas para o robô: como é o caso do transporte de prateleiras móveis.

Capítulo 7

Trabalho futuro

Depois da integração e desenvolvimento de *software* para as várias camadas do robô: ao nível do controlo, localização, navegação e tomada de decisões, é possível melhorar o desempenho de alguns destes modelos. Nesta fase inicial, a camada de tomada de decisões apenas define um conjunto de pontos sequenciais de destino, introduzidos inicialmente pelo utilizador, para o robô se movimentar, detetando quando este os atinge e fornecendo o ponto seguinte. Assim, há melhoramentos e extensões a fazer a esta camada de *software*, aumentando as suas funcionalidade e complexidade, de acordo com o tipo de aplicações futuras.

Para além disto, seria também importante desenvolver uma interface gráfica para o utilizador interagir mais facilmente com o robô, executando algumas tarefas ou visualizar o estado atual do mesmo.

Ao nível da auto-localização, depois do trabalho realizado neste projeto, é ainda necessário aperfeiçoar e realizar mais testes ao desempenho do modelo de supervisão dos algoritmos de localização, com o intuito de analisar a sua viabilidade e possível aplicação prática futura.

Quanto ao mecanismo de troca automática de baterias, apesar da estação não ter ficado pronta a tempo de se integrar no restante sistema, o conjunto de características necessárias estão pensadas e projetadas, apresentadas no Capítulo 4, cuja execução será trabalho futuro.

Em adição, sendo esta uma plataforma robótica genérica é também importante testar e adaptá-la para outras aplicações para além da logística, como vigilância e publicidade, já referidas anteriormente.

Por fim, pensa-se aplicar os algoritmos desenvolvidos a outros robôs com características similares (baixos, com pouco poder de processamento), como por exemplo a uma cadeira de rodas inteligente.

Referências

- [1] Huosheng Hu e Dongbing Gu. Landmark-based navigation of mobile robots in manufacturing. Em *Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA'99. 1999 7th IEEE International Conference on*, volume 1, páginas 121–128. IEEE.
- [2] S. Thrun, W. Burgard, e D. Fox. *Probabilistic Robotics*. MIT Press, 2005. URL: https://books.google.pt/books?id=k_y0QgAACAAJ.
- [3] Martin Lauer, Sascha Lange, e Martin Riedmiller. *Calculating the perfect match: an efficient and accurate approach for robot self-localization*, páginas 142–153. Springer, 2006.
- [4] Manuel Gouveia, Antonio Paulo Moreira, Paulo Costa, Luis Paulo Reis, e Marcos Ferreira. Robustness and precision analysis in map-matching based mobile robot self-localization. Em *EPIA, 14th Portuguese Conference on Artificial Intelligent*, páginas 243–253.
- [5] Heber Sobreira, Miguel Pinto, Antonio Paulo Moreira, Paulo Gomes Costa, e Jose Lima. Robust robot localization based on the perfect match algorithm. Em *CONTROLO2014 - Proceedings of the 11th Portuguese Conference on Automatic Control*, páginas 607–616. Springer.
- [6] Milo C Silverman, Boyoon Jung, Dan Nies, e Gaurav S Sukhatme. Staying alive longer: autonomous robot recharging put to the test. *Center for Robotics and Embedded Systems (CRES) Technical Report CRES*, 3:015, 2003.
- [7] Ren C Luo, Chung T Liao, Kuo L Su, e Kuei C Lin. Automatic docking and recharging system for autonomous security robot. Em *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, páginas 2953–2958. IEEE.
- [8] Yi-Cheng Wu, Ming-Chang Teng, e Yi-Jeng Tsai. Robot docking station for automatic battery exchanging and charging. Em *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, páginas 1043–1046. IEEE.
- [9] Johann Borenstein, HR Everett, e Liqiang Feng. Where am i? sensors and methods for mobile robot positioning. *University of Michigan*, 119(120):27, 1996.
- [10] Joao Sena Esteves. Metodologia de autolocalizacao absoluta em ambientes quase-estruturados. 2005.
- [11] Sebastian Thrun, Dieter Fox, Wolfram Burgard, e Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1):99–141, 2001.
- [12] Dieter Fox. Kld-sampling: Adaptive particle filters and mobile robot localization. *Advances in Neural Information Processing Systems (NIPS)*, 2002.

- [13] Frank Dellaert, Dieter Fox, Wolfram Burgard, e Sebastian Thrun. Monte carlo localization for mobile robots. Em *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, páginas 1322–1328. IEEE.
- [14] Joao Messias, Renato Ventura, Pedro Lima, Junieta Sequeira, Paulo Alvito, Carlos Marques, e Paulo Carriço. A robotic platform for edutainment activities in a pediatric hospital. Em *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on*, páginas 193–198. IEEE, 2014.
- [15] Kok Seng Chong e Lindsay Kleeman. Accurate odometry and error modelling for a mobile robot. Em *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, páginas 2783–2788. IEEE.
- [16] R. Siegwart e I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Book, 2004. URL: https://books.google.pt/books?id=gUbQ9_weg88C.
- [17] Chih-Fu Chang, Ching-Chih Tsai, Jui-Cheng Hsu, Shoei-Chuen Lin, e Chu-Chen Lin. Laser self-localization for a mobile robot using retro-reflector landmarks. Em *Proceedings of American Control Conference, Denver, Colorado USA*, páginas 2471–2476.
- [18] Margrit Betke e Leonid Gurvits. Mobile robot localization using landmarks. *Robotics and Automation, IEEE Transactions on*, 13(2):251–263, 1997.
- [19] Sooyong Lee e Jae-Bok Song. Mobile robot localization using infrared light reflecting landmarks. Em *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, páginas 674–677. IEEE.
- [20] Erik D Demaine, Alejandro Lopez-Ortiz, e J Ian Munro. *Robot localization without depth perception*, páginas 249–259. Springer, 2002.
- [21] R Gonzalez, F Rodriguez, JL Guzman, e M Berenguel. Comparative study of localization techniques for mobile robots based on indirect kalman filter. Em *Proceedings of IFR Int. Symposium on Robotics*, páginas 253–258.
- [22] Johann Borenstein, Hobart R Everett, Liqiang Feng, e David Wehe. Mobile robot positioning-sensors and techniques. Report, DTIC Document, 1997.
- [23] Dieter Fox, Wolfram Burgard, e Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, páginas 391–427, 1999.
- [24] Carlos Miguel Correia da Costa. Robot self-localization in dynamic environments. 2015.
- [25] Dieter Fox, Wolfram Burgard, Frank Dellaert, e Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999:343–349, 1999.
- [26] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.
- [27] Jason Zhou e Loulin Huang. Experimental study on sensor fusion to improve real time indoor localization of a mobile robot. Em *Robotics, Automation and Mechatronics (RAM), 2011 IEEE Conference on*, páginas 258–263. IEEE, 2011.
- [28] Rui Lin, Maohai Li, e Lining Sun. Real-time objects recognition and obstacles avoidance for mobile robot. Em *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, páginas 1157–1162. IEEE, 2013.

- [29] Arif Mahmood, Adeel Baig, e Qaisar Ahsan. Real time localization of mobile robotic platform via fusion of inertial and visual navigation system. Em *Robotics and Artificial Intelligence (ICRAI), 2012 International Conference on*, páginas 40–44. IEEE, 2012.
- [30] Mohinder S Grewal. *Kalman filtering*. Springer, 2011.
- [31] Vishwanath Sangale e Abhishek Shendre. Localization of a mobile autonomous robot using extended kalman filter. Em *Advances in Computing and Communications (ICACC), 2013 Third International Conference on*, páginas 274–277. IEEE, 2013.
- [32] Laura Perea, Jonathan How, Louis Breger, e Pedro Elosegui. Nonlinearity in sensor fusion: Divergence issues in ekf, modified truncated sof, and ukf. Em *AIAA Guidance, Navigation and Control Conference and Exhibit*, volume 6514, 2007.
- [33] Simon J Julier e Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. Em *AeroSense'97*, páginas 182–193. International Society for Optics and Photonics, 1997.
- [34] Fredrik Gustafsson e Gustaf Hendeby. Some relations between extended and unscented kalman filters. *Signal Processing, IEEE Transactions on*, 60(2):545–555, 2012.
- [35] Giuseppe Cotugno, Luigi D'Alfonso, Walter Lucia, Pietro Muraca, e Paolo Pugliese. Extended and unscented kalman filters for mobile robot localization and environment reconstruction. Em *Control & Automation (MED), 2013 21st Mediterranean Conference on*, páginas 19–26. IEEE, 2013.
- [36] Joseph J LaViola Jr. A comparison of unscented and extended kalman filtering for estimating quaternion motion. Em *American Control Conference, 2003. Proceedings of the 2003*, volume 3, páginas 2435–2440. IEEE, 2003.
- [37] Mathieu St-Pierre e Denis Gingras. Comparison between the unscented kalman filter and the extended kalman filter for the position estimation module of an integrated navigation information system. Em *Intelligent Vehicles Symposium, 2004 IEEE*, páginas 831–835. IEEE, 2004.
- [38] Tetsuya Wada, Zhidong Wang, Tomokazu Matsuo, Yuji Ogawa, Yasuo Hayashibara, Yasuhisa Hirata, e Kazuhiro Kosuge. Building human motion map for mobile robot in the indoor dynamic environment. Em *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, páginas 543–548. IEEE.
- [39] Kuo-Lan Su, Yi-Lin Liao, Shih-Ping Lin, e Sian-Fu Lin. An interactive auto-recharging system for mobile robots. *International Journal of Automation and Smart Technology*, 4(1):43–53, 2014.
- [40] KH Kim, HD Choi, S Yoon, KW Lee, HS Ryu, CK Woo, e YK Kwak. Development of docking system for mobile robots using cheap infrared sensors. Em *Proceedings of the 1st International Conference on Sensing Technology*, páginas 287–291.
- [41] Riccardo Gasparini, Neha Chawla, e Sabri Tosunoglu. Cybernetic helper: A wireless self-charging robot.
- [42] Riccardo Cassinis, Fabio Tampalini, Paolo Bartolini, e Roberto Fedrigotti. Docking and charging system for autonomous mobile robots. *Department of Electronics for Automation, University of Brescia, Italy*, 2005.

- [43] David Austin, Luke Fletcher, e Alexander Zelinsky. Mobile robotics in the long term-exploring the fourth dimension. Em *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, páginas 613–618. IEEE.
- [44] Youn-Ho Choi e Dong-Ha Lee. Battery swap station for mobile robot. Em *Electrical Machines and Systems (ICEMS), 2013 International Conference on*, páginas 1445–1447. IEEE.
- [45] Morgan Quigley, Brian Gerkey, e William D Smart. Programming robots with ros. 2015.
- [46] Thomas Hellström. Robot navigation with potential fields. *Department of Computing Science, Umea University, Tech. Rep*, 2011.
- [47] Johann Borenstein e Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, 7(3):278–288, 1991.
- [48] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, e Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. Em *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, páginas 155–160. IEEE, 2011.
- [49] Rodrigo AM Braga, Marcelo Petry, Antonio Paulo Moreira, e Luis Paulo Reis. Concept and design of the intellwheels platform for developing intelligent wheelchairs. Em *Informatics in control, automation and robotics*, páginas 191–203. Springer, 2009.
- [50] Brígida Mónica Faria, Luís Paulo Reis, Nuno Lau, António Paulo Moreira, Marcelo Petry, e Luís Miguel Ferreira. Intelligent wheelchair driving: Bridging the gap between virtual and real intelligent wheelchairs. Em *Portuguese Conference on Artificial Intelligence*, páginas 445–456. Springer, 2015.
- [51] Luis Paulo Reis, Rodrigo AM Braga, Márcio Sousa, e Antonio Paulo Moreira. Intellwheels mmi: A flexible interface for an intelligent wheelchair. Em *Robot Soccer World Cup*, páginas 296–307. Springer, 2009.
- [52] Marcelo R Petry, Antonio Paulo Moreira, Brigida Monica Faria, e Luis Paulo Reis. Intellwheels: intelligent wheelchair with user-centered design. Em *e-Health Networking, Applications & Services (Healthcom), 2013 IEEE 15th International Conference on*, páginas 414–418. IEEE, 2013.
- [53] Rodrigo Antonio Marques Braga, Marcelo Petry, Luís Paulo Reis, e Antonio Paulo Moreira. Intellwheels: Modular development platform for intelligent wheelchairs. *Journal of Rehabilitation Research & Development*, 48(9):1061–1077, 2011.