[S2] - Implementação do FastSLAM utilizando Visual Markers e o robô Pioneer P3-DX

[1]Daniel Fortunato, [2]Daniel Nunes , [3]David Ribeiro, [4]Pedro Fareleira

[1]Instituto Superior Técnico

3 de Maio de 2019

Abstrato—Neste artigo iremos descrever como resolvemos o problema de localização e mapeamento simultaneamente, utilizando o FastSLAM. O projeto foi desenvolvido na unidade curricular de Sistemas Autónomos, no Instituto Superior Técnico. Através do ROS (Robot Operating System), obtivemos vários packages para controlo do robô, reconhecimento de visual markers e aquisição de dados, tanto pela odometria do veículo como pelos landmarks. O robô usado foi o Pioneer P3-DX, para a identificação de Visual Markers utilizou-se a câmara Kinect e implementámos o algoritmo FastSLAM na linguagem Python. Este algoritmo utiliza o Rao-Blackwellized Particle Filter, para o sampling das localizações do robô e dos landmarks, e o Extended Kalman Filter (EKF) para estimar a posição de cada landmark observado. Iremos apresentar os resultados finais que obtivemos com dados reais e se estes foram satisfatórios face ao pretendido, para a resolução do problema de localização e mapeamento em simultâneo. Conjuntamente, os resultados experimentais irão auxiliar na nossa interpretação das vantagens e limitações do FastSLAM.

1 Introdução e Motivação

O desenvolvimento exponencial da robótica nos últimos anos expôs um nível de complexidade do qual já é esperado que robôs modernos realizem um leque de diferentes funções. Deste modo, torna-se imprescindível um robô ser autónomo ao ponto de não só localizar-se num espaço real, mas também mapear o ambiente em que se insere. Resolver o problema da localização e do mapeamento, em simultâneo, abre as portas para múltiplas outras funções que, sem estes conhecimentos, não seriam possíveis. A complexidade deste problema, tal como as implicações no mundo real foi o que nos levou a escolher este projeto.

O problema do SLAM (Simultaneous Localization and Mapping) refere-se à construção de um mapa do meio físico em que o robô se insere através da leitura de landmarks, obtidos à medida que este se movimenta no espaço. Devido ao facto de a deslocação do robô estar sujeita a erros, o que origina um problema de localização, que pode ser contornado com recurso a um mapa. Mas nem sempre há a possibilidade de aceder a um, consequentemente surge a necessidade de fazer o mapeamento.

A proposta inicial de resolução para o problema SLAM passa pela utilização do Extended Kalman Filter (EKF) para estimar incrementalmente a posição do robô e dos landmarks. No entanto, esta abordagem apresenta um problema de natureza computacional. Isto acontece pois este metodo requerer tempo quadrático ao número de landmarks no mapa, ou seja, a matriz de covariância dada pelo Kalman filter tem $O(K^2)$ elementos, em que todos têm que ser atualizados mesmo que o robô observe apenas um dos landmarks. Esta complexidade limita o número de landmarks que podem ser utilizados com esta solução.

Dada esta limitação, é necessário formar uma solução alternativa ao problema do SLAM – o FastSLAM. Este algoritmo utiliza um Rao-Balckwellized Particle Filter para estimar a localização do robô. Cada partícula possui K Kalman Filters independentes, que estimam a localização dos K landmarks com base no percurso da partícula. Representando as partículas como árvores binárias de Kalman Filters, as observações podem ser incorporadas no FastSLAM em tempo $O(M \log K)$, onde M é o número de partículas e K o número de landmarks, sendo assim mais eficaz a trabalhar com um elevado numero de landmarks, em relação ao primeiro algoritmo.

Neste artigo, iremos demonstrar, através de resultados experimentais extensos, a eficácia do algoritmo FastSLAM implementado, usando dados reais para resolver o problema de localização e mapeamento simultâneos.

2 Métodos e Algoritmos

Nesta secção vamos descrever os métodos utilizados para resolver o nosso problema. Vão ser referidos os algoritmos utilizados, assim como as *packages* do *ROS* e bibliotecas do *Python*.

No nosso projeto, o algoritmo FastSLAM foi utilizado para resolver o problema de localização e mapeamento simultâneos. Esta solução implementa uma abordagem estocástica usando o algoritmo de um *particle filter*.

Para cada partícula, a distribuição conjunta da *pose* e do mapa, dadas as observações e os movimentos, é fatorizada, baseado no Rao-Blackwellization, e reescrito como o produto da probabilidade da *pose* dadas as observações e os movimentos (*path posterior*) e a probabilidade do mapa dada a posição e as observações (*map posterior*).

$$p(s^t, L|n^t, z^t, u^t) = p(s^t|n^t, z^t, u^t) \prod_{n=1}^{N} p(l_n|s^t, n^t, z^t) \quad (1)$$

Na equação (1), s^t é a *pose*, L é o conjunto de todos os *landmarks*, n^t está associado ao índice (ID) de cada landmark identificado, z^t são os *measurements*, u^t diz respeito ao controlo do robô. Todos estes parâmetros correspondem a cada tempo t registado no percurso do robô.

Esta factorização pode ser feita pois as posições dos *landmarks* são condicionalmente independentes, assim a probabilidade do mapa é o produto de todas as probabilidades dos *landmarks*. O EKF (Extended Kalman Filter) é então usado para a representação individual de cada um dos *landmarks*, tendo assim para cada partícula um EKF por *landmark*.

Cada partícula tem uma representação de uma possível *pose* do robô. No nosso caso, consideramos apenas duas dimensões, uma vez que o robô se move apenas no chão e não há variação de altura. Assim a sua *pose* pode ser representada como:

$$X_P = [x_P, y_P, \theta_P] \tag{2}$$

Onde, para uma partícula p, x_p e y_p refletem a posição do robô, em relação a um ponto inicial (0,0), e θ_p representa a orientação do robô, em relação à sua orientação inicial.

Cada partícula também tem um mapa com a posição dos *landmarks*. Cada *landmark* é posicionado num plano bidimensional e e representado como:

$$X_M = [x_M, y_M] \tag{3}$$

A inicialização correta do algoritmo FastSLAM é muito importante. Num problema apenas de localização, cada partícula do filtro é tipicamente inicializada com uma *pose* amostrada através de uma distribuição uniforme sobre todo o espaço do ambiente. No entanto, no problema do FastSLAM cada mapa produzido é desenhado em relação à sua *pose* da partícula. Como o mapeamento é feito em simultâneo com a localização, a aleatóriedade introduzida inicialmente nunca iria ser resolvida. Por essa razão, todas as partículas são inicializadas na mesma posição, correspondendo à origem do referencial, e com o mesma orientação. Esta inicialização é valida pois no início temos a certeza absoluta de que a *pose* do robô corresponde à origem do mapa, isto é (0,0,0).

Sempre que um novo *landmark* é observado, uma estimativa da sua posição deve ser inicializada e adicionada ao mapa. Uma vez que a câmara fornece, para cada *landmark*, as coordenadas (x, y) em relação ao *frame* do marker, é necessário transformar estas coordenadas (x, y) em coordenadas (r, ϕ), com o intuito de se retirar a distância e o ângulo em relação ao *frame* da câmara. Depois de se obter estes valores tem de se passar do frame da câmara para o do robô. Para isso recorre-se as seguintes equações:

$$\begin{cases} a = d_{pc} + d_{lm} * \cos(\phi_{lm}) \\ b = -d_{lc} + d_{lm} * \sin(\phi_{lm}) \\ d_{pM} = \sqrt{a^2 + b^2} \\ \phi_{pM} = arctg(b/a) \end{cases} \tag{4}$$

Onde p corresponde ao centro robô, c ao centro câmara, m ao centro de um *landmarker* e l à lente da camara, e d representa distancias e ϕ ângulos. À distância e ao ângulo determinados (d_{pM} e ϕ_{pM}) aplicámos, por fim, as equações descritas em (5) para que as coordenadas dos *landmarks* fiquem então fixas no *frame* inicial do mapa.

$$\begin{cases} x_M = x_P + d_{pM} \cos(\theta + \phi_{pM}) \\ y_M = y_P + d_{pM} \sin(\theta + \phi_{pM}) \end{cases} \tag{5}$$

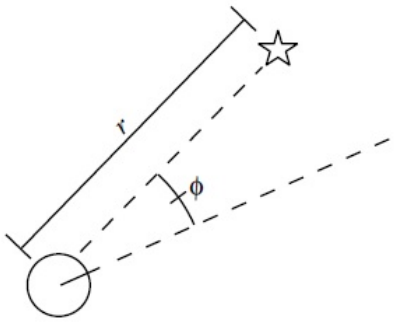

Figura 1: Measurement

Na figura 1 esta esboçado o modelo de medição, onde d_{pM} corresponde ao r e ϕ_{pM} ao ϕ.

A restante explicação da inicialização será efetuada, com maior detalhe, na secção da Implementação quando for explicado o *Extended Kalman Filter*.

Para implementar todos os métodos referidos, algumas bibliotecas ROS e python foram importadas e usadas. O ROSPY foi usado como uma interface entre ROS e *Python*, de modo a que fosse possível a passagem de dados do ROS para o ficheiro *Python*, mediante o uso de subscrições de tópicos do ROS. Através do *node* ROSARIA efetuámos o controlo móvel do robô e ao mesmo tempo este retorna a *pose* do robô com base na odometria. O *package* ROS ARUCO, em conjugação com a câmara USB Kinect, identificou os *landmarks*, sob a forma de *Aruco markers*, e apresentou informações da distância do *landmark* em relação ao robô.

3 Implementação

3.1 *Particle Filter*

Nesta secção vamos mostrar como o *Particle Filter* foi implementado.

Primeiramente, começámos por gerar um conjunto inicial de N partículas. Decidimos inicializar a posição de todas as partículas na *odometry frame*, que é a origem, e a orientação igual a zero.

Agora que temos o conjunto de partículas inicializado iremos explicar em detalhe como foi implementado o *update*. Em cada *update step*, o *particle filter* recebe um *input* da odometria que contém um movimento, o qual é

determinado subtraindo a *pose* atual do fram do *odom* e a *pose* antrior da mesma frame, obtendo assim o movimento do robô neste *step*. Depois de obter esta diferença, adicionamo-la a cada partícula em conjunto com um ruído (proporcional ao movimento feito em cada uma das três dimensões x_p, y_p e θ_p), para que cada partícula receba um *input* diferente. No fim disto, obtemos uma nova *pose* para cada partícula.

Depois procedemos ao cálculo do peso da partícula. Para obter o peso de cada partícula usámos a estimativa da posição dos *landmarks* e a *pose* da partícula para obter a distância que deveríamos medir e, de seguida, comparamos com as medidas que obtidas. Este peso resulta do produto de M densidades de probabilidade, tal como pode ser visto na equação 6, sendo M o números de *landmarks* observados.

$$w_i = \prod_{k=1}^{M} p(Z = \hat{z}_k) \tag{6}$$

Na expressão, p representa o valor da densidade de probabilidade da variável aleatória Z, que foi escolhida para ter uma distribuição Gaussiana de média z_k, que é o valor da distância e do ângulo medidos do *landmark* observado k, e variância σ_z^2, que pode ser ajustada de modo a obter melhores resultados.

Depois de obter o nosso *belief* representada por um conjunto de partículas, normalizamos o peso de todas as partículas, que são calculados a partir da equação 7:

$$w_i(\text{normalized}) = \frac{w_i}{\sum_{i=1}^{N} w_i} \tag{7}$$

No fim de termos os pesos das partículas todos normalizados decidimos então se é necessário fazer o *resampling*. Este tópico merece um pouco mais de atenção por isso iremos explicar o que implementámos um pouco mais adiante.

3.2 *Extended Kalman Filter*

Como foi explicado anteriormente, o algoritmo do FastSLAM usa um *Extended Kalman Filter* para cada *landmark*. Este EKF usa a distância e os ângulos medidos do *landmark* até à câmara para fazer a estimativa da posição dos *landmarks*.

Em primeiro lugar é feita a predição (assume-se a inicialização feita na equação (5)), onde baseado na nova posição dada do robô, se faz uma estimativa da medida que se espera obter bem como da covariância dessa medição. Este valor esperado é obtido com base no seguinte *measurement model*:

$$\hat{Z} = [\sqrt{(x_M - x_p)^2 + (y_M - y_p)^2}, arctg(\frac{y_M - y_p}{x_M - x_p}) - \theta_p] \tag{8}$$

Como se trata de um sistema não linear é necessário fazer um gradiente em ordem as coordenadas dos *landmarkers* para se fazer a aproximação de Taylor, para a determinação da matriz de covariancias (esta é inicializada como uma matriz diagonal com valores muito altos, visto não se saber onde estão os *landmarkers*).

Em seguida é feita a observação, onde é determinado o valor da diferença entre as medidas feitas e previstas. Este valor vai servir para atualizar o valor da posição do *landmarker*.

Por ultimo temos a atualização. Neste passo é determinado o valor do *Kalman gain*, com base na covariância, no gradiente do *measurement model* e na matriz inovação (esta é inicializada com base no erro da câmara e do gradiente do *measurement model*). Com base neste ganho e na diferença obtida entre a medida efetuada e esperada atualiza-se a posição do *landmarker*. Por fim atualiza-se a matriz de inovação.

Os valores da diferença entre medida obtida e esperada e da covariancia do marker determinados no processo servem para calcular a probabilidade do *landmarker* estar na posição determinada, probabilidade essa que irá ser multiplicada ao peso da partícula.

3.3 *Resampling*

O *resampling step* serve para descartar as partículas de menor peso e manter as partículas de maior peso, uma vez que estas fazem uma melhor aproximação da *pose* do robô. Contudo, sabemos que a aleatoriedade induzida neste *resampling step* é, na realidade, uma das fontes de erro do *particle filter*. Em particular, o processo de *resampling* induz uma perda de diversidade na população das partículas, que se manifesta como um erro. Este erro é chamado de variância do estimador. E embora a variância do conjunto de partículas diminua, esta variância do estimador aumenta. Controlar esta variância, ou erro, do *particle filter* é essencial para qualquer implementação prática.

A estratégia que utilizámos para reduzir este problema da variância foi recorrer a um método conhecido por *low variance resampling*, onde a ideia base consiste em, ao invés de se selecionarem *samples* independentes no processo de *resampling*, a seleção envolve um processo estocástico sequencial. Ou seja, em vez de escolher M números aleatórios e selecionar as partículas que correspondem a esses números aleatórios, este algoritmo calcula um único número aleatório e seleciona *samples* de acordo com este número. Isto é alcançado através da utilização de um número aleatório r no intervalo $[0, M^{-1}[$, onde M é o número de *samples* a ser utilizado no tempo t. O algoritmo seleciona depois as partículas, através do processo de adição de M^{-1} a r e escolhendo a partícula que corresponde ao resultado da soma. Uma explicação mais geométrica deste algoritmo encontra-se na figura 2:

As vantagens de usar *low variance resampling* são: o algoritmo cobre o espaço das *samples* de uma forma mais sistemática, uma vez que o algoritmo faz o ciclo ao longo de todas as partículas sistematicamente, em vez de as escolher aleatoriamente. Outra vantagem relaciona-se com o caso em que todas as *samples* têm o mesmo peso. O

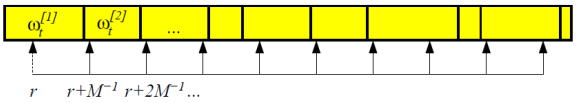

Figura 2: Princípio do procedimento da *low variance resampling*.

resultado do conjunto de partículas é equivalente ao do conjunto que foi gerado antes do *resampling* e, assim, nenhuma partícula é perdida nesse processo. Por fim, a última vantagem é a menor complexidade computacional que, neste caso, é $O(Nlog(M))$.

3.4 *ROS*

Para esta implementação é usado o ROSPY, que é a biblioteca que permite aos utilizadores terem a vantagem de utilizar a linguagem de programação PYTHON para trabalhar na estrutura do ROS, estando ambos a fornecer altos níveis de abstração para fins de implementação.

Os dados que foram utilizados para a implementação do algoritmo deste projeto vem de dois *subscribers*: o tópico que retorna a *pose*, publicada pelo ROSARIA e o tópico que retorna as distâncias dos ARUCOS à câmera, publicadas pelo ROSARUCO. A cada subscrição destas foi associada uma função de *callback* que é chamada quando os dados são recebidos.

4 Resultados Experimentais

Foram efetuadas várias experiências com diferentes variáveis de forma a testar as capacidades e os limites do algoritmo para diversas situações, tais como localizações distintas, com números totais de *landmarks* e partículas diferentes.

Operámos em três espaços físicos distintos do piso 5 da Torre Norte do Instituto Superior Técnico: os corredores do piso (que constituem um quadrado quando percorridos), a zona de elevadores e o laboratório de Sistemas Autónomos. Os dois últimos espaços com um número de *landmarks* reduzido relativamente ao primeiro.

Na primeira experiência, visitamos todos os locais com este seguimento, utilizando o FastSlam 1.0.

Na figura 3, os pontos pequenos a preto correspondem ao percurso do robô (cada ponto representa uma iteração do algoritmo, o qual só é feito quando o robô faz um movimento e pelo menos uma leitura), o ponto preto representa a posição do robô e os círculos vermelhos são estimações da localização de cada *landmark*, onde o raio do circulo é determinado pelo desvio padrão associado à posição do *landmark*. As coordenadas dos eixos x e y estão em metros. Os quadrados representam os três ambientes físicos que serão estudados isoladamente: os corredores com a cor verde; a zona dos elevadores a azul; e o laboratório em laranja. Nesta situação não foi medida a localização real dos

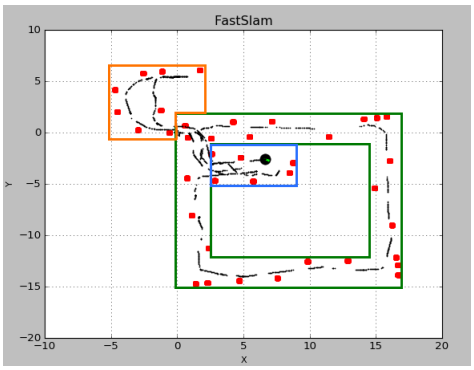

Figura 3: FastSLAM 1.0 percorrido em todos os locais

landmarks, mas obtivemos resultados qualitativos, nomeadamente na posição (0,0), onde foram colocados dois *landmarks*, um em cada lado do robô quando este se encontra na sua posição inicial. Ao terminar a primeira volta aos corredores, estes dois *landmarks* foram identificados perto da localização esperada, ou seja, uma ligeiramente acima e outra ligeiramente abaixo da origem do referencial. Sendo estes os últimos *landmarks* a ser vistos, antes do robô entrar na sala (ultimo local do teste feito), acredita-se que os outros estão próximos as suas posições reais.

A segunda experiência foi feita exclusivamente no corredor com o mesmo ponto inicial, mas agora com uma diferença: o caminho será feito em sentido horário em vez de anti-horário, de forma a comparar se a direção do percurso afeta o algoritmo.

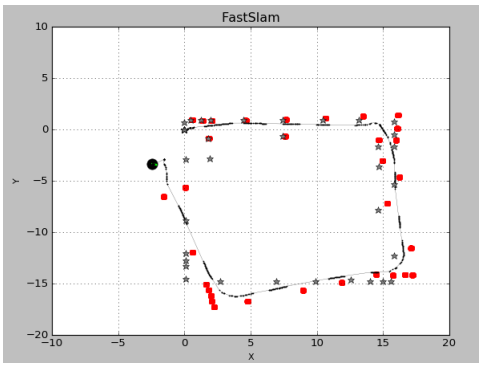

Figura 4: FastSLAM 1.0 no corredor em sentido horário

Como podemos verificar nas figuras 4 e 5, deparámo-nos com uma divergência enorme, quer na posição dos *landmarks* identificados em relação à sua posição real (nesta figura representada por estrelas cinzentas), como no percurso do robô.

Tomando atenção à primeira reta do corredor feita, do ponto (0,0) ao ponto (15,0), verificamos que os *landmarks* foram reconhecidos corretamente, mas ao virar para sul, os *landmarks* observados já não correspondem ao seu posicionamento verdadeiro. Este erro é cada vez maior à medida que o robô executa curvas para a sua direita, em cada esquina. Este facto leva-nos leva a concluir que existe um

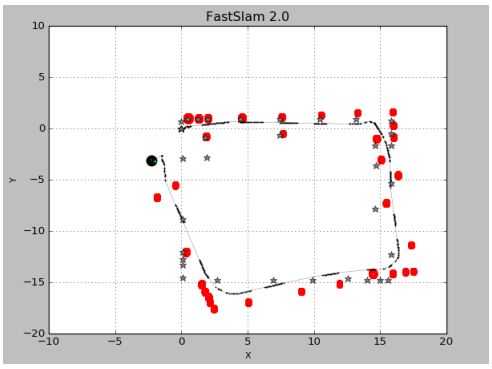

Figura 5: FastSLAM 2.0 no corredor em sentido horário

erro com o nosso projeto. Após uma análise extensiva dos nossos algoritmos, não foi possível determinar qual o erro que podera levar a este problema. Para tentar resolver este problema desenvolveu-se a versão 2.0 do FastSlam. Esta versão tem uma maior capacidade de fechar *loops* e é mais indicada para situações onde o erro da odometria é elevado, dando maior importancia as medidas feitas, atualizando a posição do robô com base na melhor medição efetuada, para cada partícula, em cada iteração. Infelizmente os resultados obtidos não melhoraram.

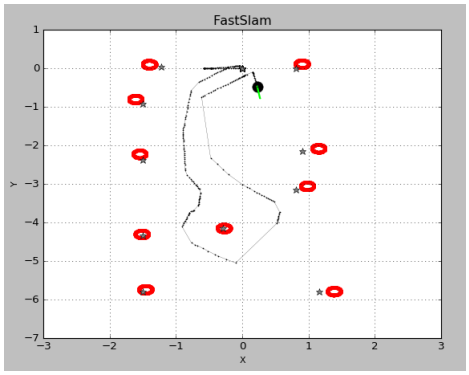

Figura 6: FastSLAM 1.0 na zona dos elevadores em sentido anti-horário

A mesma situação volta a repetir-se num ambiente diferente com menos *landmarks*. No sentido anti-horário, ou seja, efetuando as curvas para o lado esquerdo do robô, os *landmarks* apresentam erros consideravelmente menores em comparação com a experiência no sentido horário.

	Anti-horário	Horário
Média (m)	0.1463	0.3603
Desvio Padrão (m)	0.0744	0.2408
Mínimo (m)	0.0152	0.0859
Máximo (m)	0.2491	0.9774

Tabela 1: Comparação de erros na zona dos elevadores

Analisando a tabela 1, é evidente que no sentido horário existem erros com maior magnitude, onde se obteve um

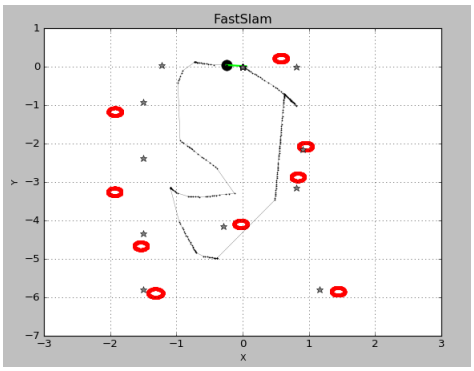

Figura 7: FastSLAM 1.0 na zona dos elevadores em sentido horário

erro máximo de quase 1 metro.

A próxima experiência foi efetuada no laboratório de Sistemas Autónomos, com um percurso mais dinâmico no qual efetuamos curvas tanto pelo lado direito como pelo esquerdo do Pioneer.

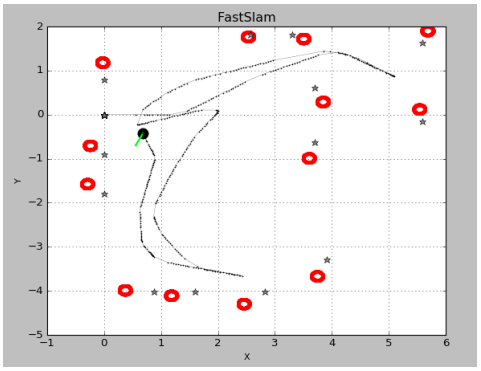

Figura 8: FastSLAM 1.0 no laboratório com percurso dinâmico A

Neste caso em particular, onde usamos um percurso com maior complexidade, podemos observar que em ambos os testes obtivemos posições para os *landmarks* que se encontram próximas da posição real, enquanto existem outros que foram identificados a uma distância considerável do local que se esperava.

	Percurso A	Percurso B
Média (m)	0.2361	0.3425
Desvio Padrão (m)	0.1516	0.1111
Mínimo (m)	0.0439	0.0582
Máximo (m)	0.5083	0.5068

Tabela 2: Comparação de erros no laboratório

Na última situação o algoritmo foi testado novamente na zona dos elevadores, mas com números de partículas distintos. É de salientar que até este ponto, todos os testes foram exercidos com um número de partículas $N = 100$.

Após examinar o algoritmo com 10, 100 e 1000

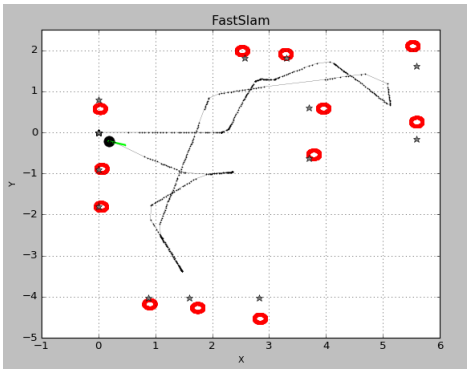

Figura 9: FastSLAM 1.0 no laboratório com percurso dinâmico B

Número de partículas	10	100	1000
Média (m)	0.3217	0.2338	0.4140
Desvio Padrão (m)	0.1113	0.0954	0.1203
Mínimo (m)	0.1454	0.0659	0.2194
Máximo (m)	0.4750	0.4042	0.6557

Tabela 3: Comparação de erros com números totais de partículas diferentes

partículas, verificou-se que o erro sofreu algumas alterações inesperadas. O pressuposto seria que o erro iria diminuir quando maior fosse o número total de partículas, o que aconteceu quando subimos de 10 para 100 partículas, mas ao aumentar para 1000, o erro subiu. Embora não possamos fornecer uma explicação concreta para este fenómeno, pode-se prever que o numero de partículas não é fator determinante nesta situação, ou seja, os bons resultados não são de todo proporcionais ao numero de partículas.

Em seguida abalizamos o tempo de execução do algoritmo:

	Tempo
Média (ms)	23.8
Desvio Padrão (ms)	8.7
Mínimo (ms)	15.3
Máximo (ms)	62.1

Tabela 4: Tempo das iterações

Os resultados obtidos para os tempos de cada iteração são bastante satisfatórios, encontrando-se sempre abaixo do valor de atualização da odometria (componente mais lenta) que atualiza o seu valor num intervalo de tempo de 100 ms, não tendo assim qualquer tipo de influencia em possíveis resultados obtidos.

Importa, ainda, realçar que nestas experiências, e sobretudo nas experiências nos corredores, poderíamos ter obtido melhores resultados se os corredores tivessem melhor iluminação. Em situações de menor luminosidade o reconhecimento dos *markers* fica bastante condicionado. Com vista a melhorar esse reconhecimento, efetuámos a calibração da câmara, contudo, não registámos grandes melhorias.

5 Conclusões

Em suma, o algoritmo de FastSLAM que desenvolvemos aparenta ser pouco consistente. O algoritmo mostra-se robusto em certas situações, apresentando erros mínimos, contudo, em boa parte dos casos apresenta algum erro e não se mostrou capaz de se localizar em situações de maior movimentação independente, ou seja foi incapaz de fechar o *loop* na situação exposta na figura 4. Ponderam-se que possíveis causas para este problema sejam erros de odometria elevados entre iterações relacionados com os sensores do Pioneer, possíveis erros na leitura da posição dos markers ou uma má estimativa da evolução do erro ao longo do percurso. Mesmo falando no algoritmo este apenas age "ativamente"perante as posições dos *markers*, sendo que a posição do robô fica dependente da existência de partículas na posição real ou próximo a ela. Notámos, também, que, ao utilizar um *Particle Filter*, é necessário ter algumas cautelas em relação ao número de partículas a utilizar, devido à relação linear que este apresenta com o tempo de execução do algoritmo. A realização do FastSLAM implicou uma constante aprendizagem do ROS, por parte do grupo de trabalho. Por outro lado, o FastSLAM é um algoritmo com um nível de complexidade elevado, dado que comprime muitos conhecimentos paralelos, tais como, por exemplo, o *Extended Kalman Filter* e o *Particle Filter*. Por esse motivo, nem sempre conseguimos obter o progresso que pretendíamos no projeto.

A nível aplicacional, o projeto desenvolvido mostra-se muito interessante, pois, apesar de termos considerado algumas simplificações na nossa abordagem (por exemplo, só considerámos duas dimensões,assumindo que a altura do robô nunca varia), acabámos por desenvolver a base/essência do FastSLAM, que pode ser aplicado a projetos mais ambiciosos, como, por exemplo, uma localização e um mapeamento simultâneo para um robô da NASA, em Marte.

6 Referências Bibliográficas

[1] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, J. Nieto, E. Nebot. "FastSLAM: An Efficient Solution to the Simultaneous Localization And Mapping Problem with Unknown Data Association"

[2] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit. "FastSLAM: A factored solution to the simultaneous localization and mapping problem", in Proc. AAA1 Nat. Conf. Artif. Intell. , 2002, pp. 593–598

[3] P. Lima. SAut, Class Lecture, Topic: "SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)" Instituto Superior Técnico, 2017

[4] S. Thrun, W. Burgard and D. Fox. Probabilistic Robotics. Cambridge, Massachusetts: MIT Press, 2006