Relatório de Sistemas Autónomos [LMP]

Murillo Teixeira - 105038
murillo.teixeira@tecnico.ulisboa.pt

Pedro Bernardes - 100057
pedro.bernardes@tecnico.ulisboa.pt

Afonso Nobre - 99883
afonso.nobre@tecnico.ulisboa.pt

Mauro Neves - 96284
mauro.neves@tecnico.ulisboa.pt

***Abstract*—Este trabalho apresenta a implementação do método de localização de Monte Carlo, algoritmo amplamente utilizado para resolver o problema de localização global de sistemas autónomos. Foram recolhidos dados para teste com o robô Pioneer3DX e realizada a análise dos resultados obtidos em diferentes ambientes.**

I. Introdução e Motivação

A localização de robôs móveis é um problema que consiste em estimar a sua posição num determinado ambiente. Muitas vezes designado por *pose estimation*, o problema da localização tem duas variantes muito importantes: ***global localization*** e ***position tracking***. A principal diferença entre as duas é que na *global localization* o robô não tem conhecimento da sua posição inicial; enquanto que na segunda o robô parte de uma pose (posição + orientação) previamente conhecida. Nos dias de hoje, diversas tarefas no contexto da robótica exigem informação sobre a localização dos objectos que estão a ser manipulados. Por conseguinte, a capacidade de um robô se localizar desempenha um papel importante nas recentes aplicações de robôs móveis. Apesar de ser um método relativamente recente, a **M**onte **C**arlo **L**ocalization (MCL) já se tornou um algoritmo bastante popular no campo da robótica, uma vez que: é fácil de implementar; tende a funcionar bem numa vasta gama de problemas de localização; reduz drasticamente a quantidade de memória necessária, em comparação com a localização baseada em *grid* e pode integrar medições a uma frequência consideravelmente mais elevada.

II. Algoritmos e Métodos

A. Monte Carlo Localization

Também conhecido por algoritmo de filtros de partículas, a ideia chave consiste em determinar a pose x de um robô num determinado ambiente. Para tal, é gerado um conjunto de N partículas aleatórias $\mathcal{X} = \{x^{[1]}, x^{[2]}, ..., x^{[N]}\}$, que representam a crença da pose. Para o problema de localização de um robô num ambiente 2D, cada partícula representa um possível estado do robô (x, y, θ), e para cada partícula é atribuído um fator de importância (peso de cada partícula, w). Um conjunto de partículas constitui uma aproximação discreta de uma distribuição de probabilidade. O MCL processa-se em três partes principais: ***motion model*** - quando o robô se move, são geradas N novas partículas que aproximam a posição do robô após o comando de movimento - ***measurement model*** - quando o robô deteta o seu ambiente, atribui um peso w a cada partícula - e ***resampling*** - em que se elegem N novas partículas da crença anterior, de acordo com a distribuição dos pesos. [2] A seguir são especificadas em detalhe cada uma dessas 3 partes.

B. Motion model: Odometry motion model

O modelo de movimento escolhido foi o **odometry motion model**, que utiliza informação relativa da odometria interna do robô. Supondo que, num intervalo de tempo $[t-1, t]$, o robô avance da pose x_{t-1} para x_t, os dados de odometria que são reportados indicam um avanço relativo desde $\overline{x}_{t-1} = (\overline{x}\ \overline{y}\ \overline{\theta})$ até $\overline{x}'_t = (\overline{x}'\ \overline{y}'\ \overline{\theta}')$. Importa salientar que estas medidas de odometria estão incorporadas dentro do referencial de coordenadas intrínseco ao robô, cuja relação com as coordenadas do mundo real é desconhecida, daí a utilização das barras por cima das variáveis. Deste modo, a diferença entre as duas poses serve como um bom indicador no que diz respeito à informação de movimento, isto é, odometria relativa do robô (u_t): $u_t = \overline{x}_t - \overline{x}_{t-1}$. A movimentação do robô surge como a sequência de três etapas: uma rotação, seguida de uma translação e, no fim, mais uma rotação. Para efeitos de ilustração, pode observar-se a figura abaixo (Fig. 1) [1].

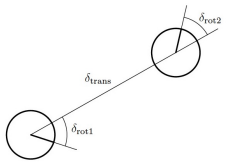

Fig. 1: Odometry Motion Model

Algorithm 1 Odometry Motion Model

$\delta_{rot1} = atan2(\overline{y}' - \overline{y}, \overline{x}' - \overline{x})$
$\delta_{trans} = \sqrt{(\overline{x} - \overline{x}')^2 + (\overline{y} - \overline{y}')^s}$
$\delta_{rot2} = \overline{\theta}' - \overline{\theta} - \delta_{rot1}$

$\hat{\delta}_{rot1} = \delta_{rot1} - \textbf{sample}(0, \alpha_1\delta_{rot1} + \alpha_2\delta_{trans})$
$\hat{\delta}_{trans} = \delta_{trans} - \textbf{sample}(0, \alpha_3\delta_{trans} + \alpha_4(\delta_{rot1} + \delta_{rot2}))$
$\hat{\delta}_{rot2} = \delta_{rot2} - \textbf{sample}(0, \alpha_1\delta_{rot2} + \alpha_2\delta_{trans}$

$x' = x + \hat{\delta}_{trans}\cos(\theta + \hat{\delta}_{rot1})$
$y' = y + \hat{\delta}_{trans}\sin(\theta + \hat{\delta}_{rot1})$
$\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

return $x_t = (x', y', \theta')^T$

No Algoritmo 1 [1] pode verificar-se o pseudocódigo referente ao método utilizado para calcular x_t. Este algoritmo recebe como dados de entrada os dados de odometria relativa do robô u_t e a pose anterior, x_{t-1}, gerando uma amostra de partículas distribuídas de acordo com $p(x_t|u_t, x_{t-1})$. Para tal, faz-se uso da função **sample**(μ, σ^2). As variáveis α dizem respeito aos parâmetros de ruído especificados pela implementação.

C. *Measurement model: Likelihood field model*

Os modelos de medição descrevem o processo de formação através do qual as medições sensoriais são produzidas no mundo físico. De modo formal, o *measurement model* é definido como uma distribuição de probabilidade condicional $p(z_t|x_t, m)$, em que x_t diz respeito à pose do robô, z_t é a medição no instante t e m é o mapa do ambiente estudado. O modelo escolhido para o desenvolvimento deste trabalho foi o *Likelihood field model*. A ideia principal deste modelo passa por projetar os pontos finais de um sensor z_t no espaço de coordenadas globais do mapa m. Para isto acontecer, é preciso saber em que coordenadas do mapa se encontra o sistema de coordenadas do robô, em que ponto do robô se origina o feixe do sensor z_k e para onde aponta. Mantendo uma visão bidimensional do ambiente, assume-se que a localização relativa do sensor no sistema de coordenadas local fixo do robô por $(x_{k;sens}\ y_{k;sens})^T$, e a orientação angular do feixe do sensor em relação à direção do robô por $\theta_{k;sens}$.

O ponto final de cada medição, z_k^t, é agora mapeado no sistema de coordenadas global através de uma transformação trigonométrica que leva em conta a posição da partícula, a posição relativa do sensor, o ângulo da medição feita pelo laser e o *range* medido.

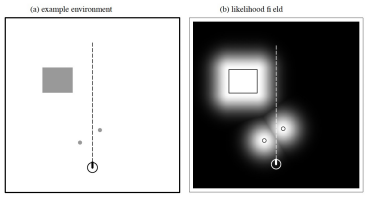

Fig. 2: Exemplo do *Likelihood field* (b)

O ruído resultante do processo de medição é modelado utilizando Gaussianas. No espaço x-y, isto implica encontrar o obstáculo mais próximo no mapa. Seja $dist$ a distância euclidiana entre as coordenadas de medição $(x_{z_t}^k\ y_{z_t}^k)^T$ e o objeto mais próximo no mapa m. Então, a probabilidade de uma medição do sensor é dada por uma distribuição gaussiana centrada em zero que modela o ruído do sensor: $p_{hit}(z_k^t|x_t, m) = \varepsilon_{\sigma_{hit}^2}(dist^2)$. A Figura 2a mostra um mapa e a Figura 2b mostra a probabilidade gaussiana correspondente para os pontos de medição $(x_{z_t}^k\ y_{z_t}^k)^T$ no espaço 2-D. Quanto mais clara for uma região, maior é a probabilidade de se detetar um objeto com um laser.

Após o cálculo de p_{hit}^k para cada uma das k leituras, os pesos podem ser definidos como a produtória de $z_{hit}p_{hit}^k + z_{rand}$ para cada valor de k. Os parâmetros z_{hit} e z_{rand} são as importâncias dadas à leitura e a uma componente aleatória que provém de possíveis erros de medição, definidos pelo implementador. [1]

D. *Resampler: Low Variance Resampler*

O processo de reamostragem num filtro de partículas remove as partículas com um peso de importância baixo e substitui-as por partículas com um peso elevado. Sem este passo, a maioria das partículas representaria estados com baixa probabilidade após algum tempo. O método de amostragem escolhido foi o *Low Variance Resampler*, que é utilizado para reduzir a variância da distribuição discreta das partículas. Inicialmente, é cálculado um número aleatório com valor máximo equivalente ao inverso do número de partículas e estabelece-se um *threshold* que varia de acordo com a indexação da partícula na estrutura de dados. O critério de escolha de uma partícula recaí então sobre se a soma cumulativa de todos os pesos das partículas contabilizadas pelo algortimo ultrapassa esse threshold ou não. Isto é ilustrado pelo Algoritmo 2. [1]

Algorithm 2 Low Variance Resampler.

$\overline{\mathcal{X}}_t = \emptyset$
$r = \text{rand}(0; M^{-1})$
$c = w_t^{[1]}$
$i = 1$
for $m = 1$ **to** M **do**
  $u = r + (m-1) \cdot M^{-1}$
  while $u > c$ **do**
    $i = i + 1$
    $c = c + w_t^{[i]}$
  end while
  add $x_t^{[i]}$ to $\overline{\mathcal{X}}_t$
end for
return $\overline{\mathcal{X}}_t$

III. Implementação

A. *Rosnode* `monte_carlo_localization`

Toda a implementação da solução foi realizada seguindo a estrutura de um pacote para o ROS (*Robotic Operating System*). O pacote conta com um `rosnode` (implementado por meio da biblioteca `rospy`) que possui `Subscribers` para ler os dados provenientes da odometria (`rostopic \pose`) e do laser (`rostopic \scan`).

Na inicialização do nó, são instanciados objetos das classes *Map*, onde é guardado o mapa e calculado o seu likelihood field e *ParticleFilter*, que lida com as partículas e implementa de facto o filtro usado para o cálculo das posições. Dentro do *ParticleFilter* são inicializadas as partículas (local ou globalmente, dependendo do problema a ser resolvido), instanciadas pela classe *ParticleSet* que extende a `np.ndarray`.

O nó conta ainda com um timer para a execução iterativa do algoritmo, com a frequência definida no arquivo `config.yaml`. No callback desse timer são chamadas as

funções principais do Monte Carlo Localization. Inicialmente, é chamado o método que realiza a movimentação das partículas a partir dos dados da odometria. Seguidamente é chamado o método do *likelihood field algorithm*, que calcula os pesos das partículas a partir dos dados de medição do laser. Com os pesos calculados, as partículas que estão fora do mapa têm o seu peso atualizado para zero e em seguida esses pesos são normalizados (uma exigência do *Low Variance Resampler*). Por fim, caso o número efetivo de partículas seja menor que o *threshold* estabelecido, é realizado o *resampling*.

A visualização dos resultados é feita através da utilização da classe `Visualizer`, que atualiza os plots das partículas, odometria e da visualização das leituras do laser em tempo real usando a biblioteca *matplotlib*.

B. *Odometry Motion Model*

A versão do Odometry Motion Model implementada pelo grupo de trabalho seguiu passo por passo a algoritmia matemática descrita no respetivo pseudo-código apresentado. Para o efeito, recorreu-se às funções disponibilizadas pela biblioteca `numpy` do *Python*. O bom funcionamento deste algoritmo foi testado num ambiente de microsimulação com dados fabricados pelo grupo. Neste teste, procurou-se garantir que as partículas eram capazes de descrever um caminho de forma coerente com os dados a que o algoritmo foi sujeito, com e sem ruído introduzido nos dados.

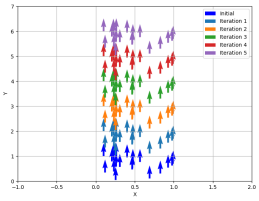

Fig. 3: Simulação sem ruído

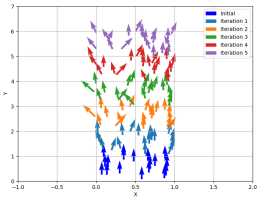

Fig. 4: Simulação com ruído

Em relação à implementação real do *motion model*, os dados de odometria foram adquiridos através da função `rospy.Subscriber()` da biblioteca `rospy`, que comunica com o `rostopic \pose`, onde o robô guarda os dados da odometria. Os dados disponibilizados descrevem um deslocamento relativo a uma referência inicial do robô, ou seja, os dados obtidos em cada período de deslocação são relativos a um único ponto fictício, não sendo possível obter diretamente o valor da deslocação entre *callbacks* (que corresponde ao tipo de dados relevantes para a nossa implementação). Assim, foi necessário recorrer à salvaguarda de dados de odometria anteriores, para se proceder ao cálculo das deslocações relativas entre *callbacks*. Para além disto, foi necessário fazer ainda a conversão de alguns dados passados pelo `\pose` para unidades tratáveis pela implementação, nomeadamente os dados relativos à orientação do robô, que se encontravam em forma quarteniana. Assim, procedeu-se ao desenvolvimento de uma função que realizasse a conversão dos dados para coordenadas eulerianas.

Para além disto, na execução de testes com dados retirados de *rosbags* levantadas pelo grupo, verificou-se que os parâmetros α que melhor descrevem potenciais erros nos dados adquiridos pela odometria do robô são [0.01, 0.01, 0.4, 0.008]. O método para obtenção destes parâmetros baseou-se essencialmente na análise qualitativa dos resultados obtidos.

C. *Likelihood Field Algorithm*

De forma a simplificar a implementação deste algoritmo e acelerar a sua execução, o likelihood field foi calculado com antecedência no processo de inicialização do mapa, por meio da função `distance_transform_edt` da biblioteca `scipy.ndimage`.

Assim, para o cálculo dos pesos torna-se necessário somente calcular as posições das leituras do laser para cada uma das partículas, como descrito na subseção II-C e realizar um posterior cruzamento de dados com o likelihood field, que fornece o valor de p_{hit}. Após isso, o cálculo do peso de cada partícula se dá como o produto entre os diversos $z_{hit}p_{hit} + z_{rand}$ de cada leitura.

Antes de implementar no ambiente do nó com as demais funções, foi feita a microsimulação que pode ser vista na Fig. 5. Nela, é possível perceber, no canto superior esquerdo, a simulação da partícula original, de onde são tiradas os ranges do laser, e 8 partículas em posições ligeiramente diferentes da original, com os respectivos pesos calculados acima de cada plot. Foi possível validar, com essa simulação, que de fato o quão mais próximas da posição real as partículas estiverem, maior o peso calculado.

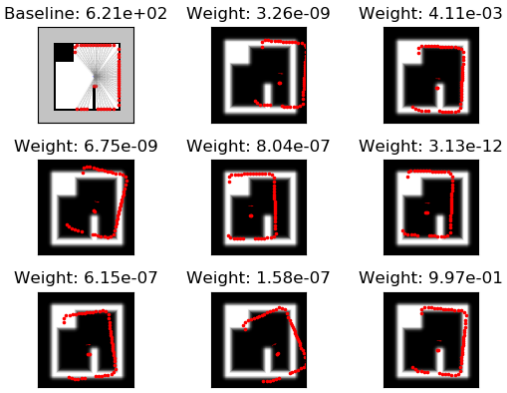

Fig. 5: Microsimulação do algoritmo de likelihood field

D. *Low Variance Resampler*

Mais uma vez, e à semelhança do que foi feito na implementação do *Odometry Motion Model*, o grupo desenvolveu código para esta parte do projeto seguindo a algoritmia matemática apresentada no pseudo-código, com o auxílio da biblioteca `numpy`. Executaram-se testes num ambiente de simulação com dados fabricados, procurando que o número total de partículas não convergisse para um ponto único do mapa, preservando assim a hipótese de correção de possíveis

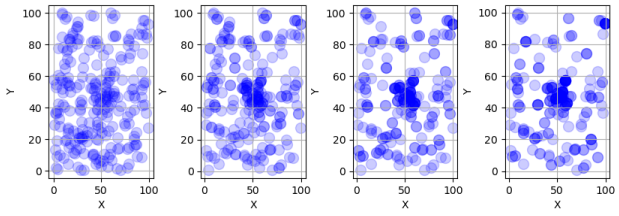

Fig. 6: Exemplo de uma microsimulação do algoritmo de Reamostragem

erros na atribuição dos pesos ou nalgum processo de *resampling* anterior.

Foram ainda desenvolvidos alguns métodos para uma utilização mais criteriosa do *resampler*, com o objetivo de combater a convergência exagerada de partículas e para desprezar partículas que, depois da aplicação do *motion model*, saíssem do mapa. No combate da convergência indesejada, calculou-se um parâmetro relativo aos pesos das partículas que indica se a distribuição das partículas apresenta uma variância demasiado grande e, portanto, se precisa de ser atualizada . Este parâmetro é o $n_{eff} = \frac{1}{\sum_{m=1}^{M}(w_t^{(m)})^2}$. O n_{eff} funciona como uma estimativa da variância da distribuição dos pesos das partículas que é comparável com o número de partículas existente no conjunto. Assim, este parâmetro permite estabelecer um *threshold* fixo relacionado com o número de partículas, que também é um parâmetro fixo no nosso algortimo. Depois de alguns testes, o grupo chegou à conclusão que se devia fazer *resampling* assim que $n_{eff} < 0.9M$, com $M =$ número de partículas. Ao longo dos testes, verficou-se também que o algoritmo perdia alguma exatidão e precisão na localização final devido a partículas que saíam do mapa, mas às quais era possível atribuir algum peso. Assim, desenvolveu-se código que eliminasse as partículas que se encontravam fora do mapa. Este código converte a posição em metros do robô para a posição em pixéis dentro do mapa. De seguida, verifica se a posição em questão corresponde à uma zona branca do mapa e depois elimina os pesos das partículas que não foram sinalizadas.

E. *Estimador da posição*

Para estimar a posição do robô, foi utilizado o algoritmo K-Means de *clustering* do *sklearn.cluster*, para separar os resultados em *clusters* de partículas. A partir da média da posição e orientação das partículas do *cluster* mais populoso é estimada a posição e orientação do robô. O número de clusters é definido de acordo com a quantidade de regiões distintas em que se concentram partículas.

IV. Resultados

Para avaliar a performance do algoritmo implementado, o grupo optou por realizar testes em duas vertentes: *position tracking* e *global localization*. No caso de *position tracking*, a posição inicial do robô é conhecida e as partículas são inicializadas ao seu redor de acordo com uma distribuição gaussiana de variância conhecida ($50cm^2$). Por outro lado, no caso de *global localization*, nada se sabe sobre a posição inicial do robô e as partículas são inicializadas e distribuídas uniformemente pelo mapa.

Como *baseline* comparativa para os resultados obtidos, foram utilizados os *outputs* do pacote *amcl* do ROS [3], que implementa o algoritmo *Adaptative Monte Carlo Localization*. Vale realçar que o próprio *amcl* nem sempre alcança o *ground truth*, mas constitui uma boa base comparativa, dada a sua eficiência.

Tanto nos testes de *position tracking* quanto nos de *global localization* foram utilizadas 5000 partículas. Na secção IV-E a performance do algoritmo é avaliada face ao número de partículas utilizado.

Vale ainda realçar que o algoritmo de MCL (tal como o *amcl*) possui uma importante componente estocástica, que influencia os resultados. Assim, parte dos resultados obtidos provém da inicialização aleatória das partículas e do resampling.

A. *Descrição do ambiente de testes*

Os testes ao programa foram executados através da utilização de *bags* e mapas de três ambientes diferentes (Fig. 7): o 5º piso da Torre Norte, o elevador do mesmo piso e a sala LSDC4. O critério usado para a escolha destes ambientes seguiu, essencialmente, duas propriedades topológicas do espaço: simetria e área percorrida pelo robô. Assim, os resultados obtidos para o mapa do elevador correspondem a uma topologia onde a área percorrida é pequena e o espaço é assimétrico; os resultados obtidos para a sala LSDC4 correspondem a uma topologia com uma maior área percorrida e com alguma simetria e os resultados obtidos para o corredor do 5º piso da Torre Norte correspondem a uma topologia com uma grande área percorrida e com alta simetria.

Fig. 7: Mapas utilizados para a localização

B. *Análise qualitativa do Position tracking*

Na Fig. 8 encontram-se representados os trajetos calculados pelo programa, onde a posição inicial do robô é conhecida para cada um dos mapas. A trajetória obtida pela implementação do grupo assemelha-se, qualitativamente, à fornecida pelo *amcl*.

Um ponto interessante neste tópico e que merece ser realçado está relacionado com a dispersão das partículas, representadas pelos pontos vermelhos: no *amcl*, as partículas encontram-se menos dispersas quando comparadas com a implementação do grupo. Assim, por vezes, este acontecimento pode resultar em ”pulos” no estimador de posição, como é possível observar nas inflexões da *bag* referente à sala LSDC4.

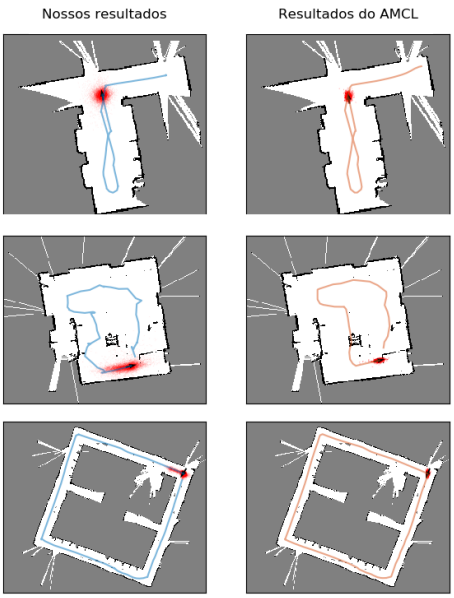

Fig. 8: Trajetórias obtidas para *position tracking* pela implementação do grupo (esquerda) e pelo *amcl* (direita) para bags das 3 localizações: elevador (acima), LSDC4 (ao centro) e do 5º piso (abaixo);

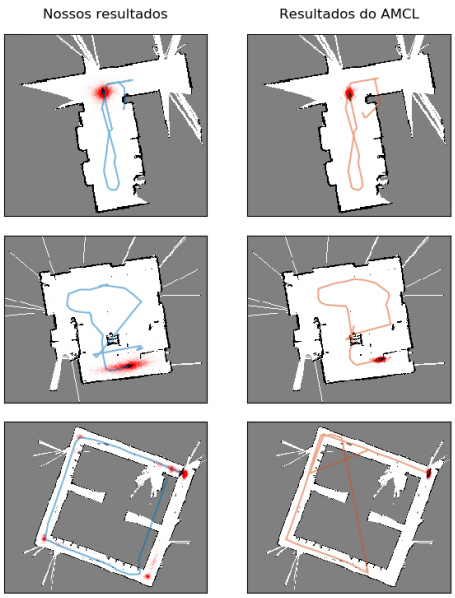

Fig. 9: Trajetórias para *global localization* obtidas pela nossa implementação (esquerda) e pelo *amcl* (direita) para bags das 3 localizações: elevador (acima), LSDC4 (ao centro) e do 5º piso (abaixo);

C. *Análise qualitativa do Global localization*

Na Fig. 9, encontram-se representados os trajetos resultantes do algoritmo de localização global. É possível constatar que, em todos os mapas, o algoritmo passa por um período de acomodação inicial, em que a trajetória do estimadores é espúria. Contudo, em ambos os casos, essa situação é posteriormente normalizada, pelo que a restante trajetória já se assemelha bastanta às obtidas com o *position tracking*.

É igualmente interessante realçar que no teste realizado no 5º piso é possível observar a formação de "nuvens" de partículas em posições que acompanham a simetria do mapa. Tal acontecimento deve-se ao facto de o conjunto de *motion model* e *measurement model* não serem suficientes para diferenciar entre posições simétricas.

D. *Análises quantitativas e comparativas dos métodos*

Para analisar quantitativamente os resultados, o grupo recorreu à análise do erro translacional (distância entre a estimativa de posição da implementação do MCL feita pelo grupo e a retornada pelo *amcl*) e angular, tanto do *position tracking* quanto do *global localization*.

Os resultados encontram-se representados na Fig. 10. É possível observar pela análise da curva que, de início, há um erro elevado para o *global localization*, mas que rapidamente é corrigido pelo measurement model, dadas as assimetrias do mapa do Elevador, levando o programa a convergir com o *amcl* após aproximadamente 30s. Para essa *bag*, a *position tracking* também apresenta bons resultados e uma convergência rápida, com um erro menor que 0.25m após 50s de execução. Os erros angulares de forma geral são pequenos e devem-se maioritariamente ao atraso na resposta de um algoritmo em relação ao outro, mas não ultrapassam os 20° após a convergência.

É possível observar uma situação semelhante nas Figuras 11 e 12, em que o position tracking apresenta erros significativamente menores de início. Com o passar do tempo e à medida que o algoritmo de *global localization* converge, esses erros acabam por se igualar.

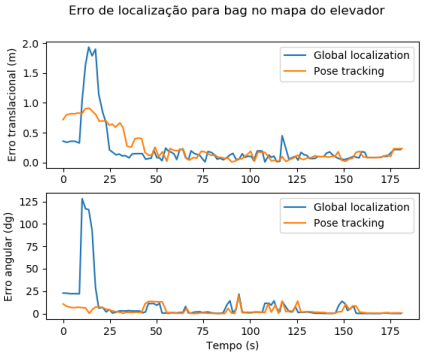

Fig. 10: Erro para os testes realizados na rosbag do Elevador

E. *Evolução dos resultados obtidos com o número de partículas*

Sendo o nosso algoritmo uma especificação de um filtro de partículas, a sua eficácia está altamente dependente do número de partículas que são utilizadas para criar a distribuição discreta que carateriza a crença da posição do robô. Assim, procurou-se fazer uma análise comparativa entre os resultados

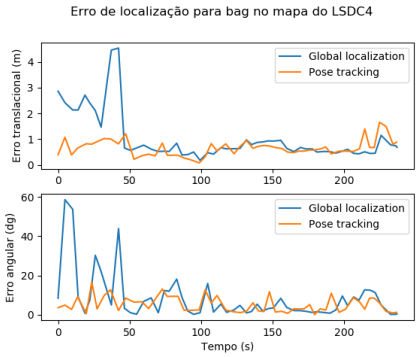

Fig. 11: Erro para os testes realizados na rosbag do LSDC4

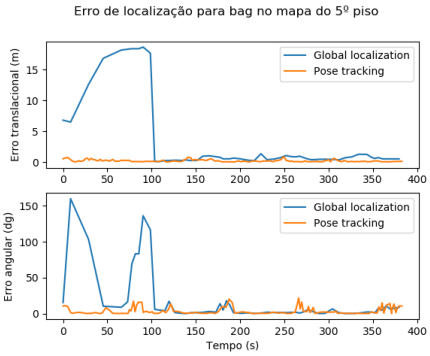

Fig. 12: Erro para os testes realizados na rosbag do 5º piso

obtidos pelo nosso algoritmo e pelo *amcl*, variando o número de partículas. Esta análise tem como base os resultados obtidos no ambiente da sala do elevador e do corredor do 5º piso, de forma a ser possível avaliar a necessidade de um maior número de partículas para ambientes com uma maior área e simetria.

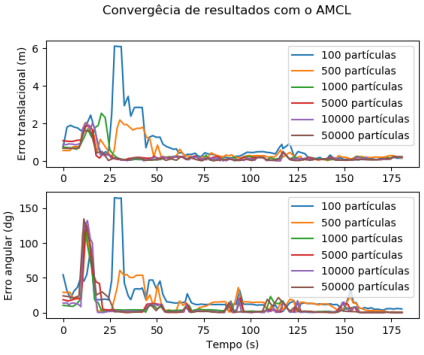

Fig. 13: Resultados para a sala do elevador

Na fig.13 encontram-se representados os resultados obtidos para o ambiente do elevador. É possível concluir através da sua análise que se obtêm melhores resultados (mais convergentes com o *amcl*) para a execução do programa com um conjunto de partículas entre 1000 e 10000 partículas. Por outro lado, no ambiente do corredor (Fig.14), verifica-se que os resultados mais convergentes com o *amcl* são obtidos com um conjunto de 5000 a 20000 partículas.

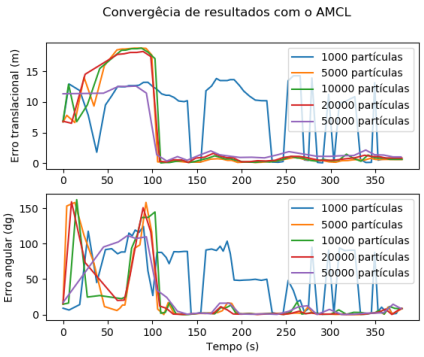

Fig. 14: Resultados para o corredor

V. Conclusão

Consideramos a execução do projeto bem sucedida, uma vez que foram obtidos resultados satisfatórios, como é possível concluir pelas análises anteriores. Verifica-se que existe uma convergência muito próxima entre os nossos resultados e os do *amcl*, em todos os ambientes testados.

Apesar disso, a escolha dos parâmetros do *Likelihood Field Algorithm* e do *threshold* do *resampler* foi pouco aprofundada, pelo que os resultados obtidos poderiam ter sido mais exatos e precisos.

Futuramente, seria interessante corrigir estes aspetos e robustecer a nossa implementação para casos de *kidnapping*, reservando um conjunto de partículas com poses aleatórias ao longo de toda a execução do algoritmo.

De forma geral, conclui-se que os objetivos deste projeto foram alcançados com sucesso e que o processo contribuiu para uma maior familiarização com o *ROS* e com as ferramentas de *Python* para desenvolvimento de projetos neste ambiente.

References

[1] Thrun S., Burgard W., Fox D. (2005), *Probabilistic robotics*, MIT Press, Cambridge, Massachusetts, London, England .

[2] Thrun S., Burgard W., Fox D., Dellaert F. (2001), *Monte Carlo Localization: Efficient Position Estimation forMobile robôs*

[3] Gerkey B. P., "amcl" ROS Wiki. http://wiki.ros.org/amcl (acessado em 06 de junho, 2023).