Localização Indoor baseada no Laser Range Finder (LiDAR) usando o Extended Kalman Filter

João Lopes (83486) André Meneses (84005) Gonçalo Carvalho (84065) Sophie Taboada (84187)

***Resumo*—A localização de robôs é um conhecido problema na área da Robótica, sendo bastante comum a escolha de uma soluções que utilizam ambos métodos de localização absoluta e relativa. Neste artigo, o nosso foco é o desenvolvimento de um algoritmo de fusão entre a Odometria e o Laser Range Finder (LiDAR) para a localização precisa de um robô de serviço (MBOT). Utiliza-se para isso o Extended Kalman Filter (EKF) para solucionar este problema de estimação estocástica de sistemas dinâmicos não lineares, e o conhecido algoritmo de "model matching", Iterative Closest Point (ICP) que estimativa da posição do robô no mapa com o dados do laser. Desta maneira é possível corrigir os erros sistemáticos relacionados nos sensores do robô, e erros não sistemáticos relacionados com imperfeições no piso e certas manobras do robô. Para avaliar o método proposto, comparamos o desempenho do nosso algoritmo de localização com o Adaptative Monte Carlo Localization (AMCL) do MBOT como ground-truth. O projecto é testado com dados reais efectuado no oitavo piso da Torre Norte do IST num ambiente indoor. Os resultados são positivos e consistentes. É testada a localização para velocidades maiores que permite confirmar a robustez do programa desenvolvido.**

Palavras-Chave: Localização, robô, Extended Kalman Filter, Iteractive Closest Points, Laser Range Finder, feature matching

I. Introdução

Nos dias de hoje, o problema de auto-localização, na estimativa da pose de um determinado sistema (posição + orientação) em ambientes internos tem-se tornado cada vez mais um problema comum. Este projecto surge no âmbito da cadeira de Sistemas Autonomos cujo objectivo é o de resolver o problema de auto-localização num plano 2D com base no LiDAR e no algoritmo Extended Kalman Filter (EKF) que tem uma performance melhor do que a estimação da pose baseada apenas na Odometria. Neste artigo abordaremos uma implementação do EKF, usando o MOnarCH [1], mais conhecido como MBOT, e um LiDAR cuja a ideia principal é combinar os dados com os sinais desse sensor com a odometria, a fim de obter informações que permitam uma localização precisa do nosso robot. O MBot possui uma medição Unidade Inercial Incorporada (IMU) que permite medir os deslocamentos e orientações nos eixos x,y,z do corpo do robot ao longo do tempo. A odometria, em oposição a métodos de localização baseados em dados de laser, permite apenas localização relativa, i.e, calcula os movimentos com base na última pose. A consequência disto é que os erros de estimativa são acumulados ao longo do tempo e, portanto, a localização da odometria torna-se imprecisa ao longo do tempo. O restante deste artigo possui os métodos e algoritmos utilizados, implementação, resultados experimentais e conclusões retiradas.

II. Métodos e Algoritmos

A. Modelo Dinâmico do Sistema

Neste projecto foi estudado um sistema não linear através dos seus modelos de acção e de medida cujas equações estão apresentadas da forma seguinte.

$$\left\{ \; X_{k+1} = f(X_k, U_K) + v_k \quad k = 0, 1, ..., . \right. \tag{1}$$

Onde $X = (x, y, \theta)$

O X_k e $X_{k+1} \in R^n$ correspondem ao vector de estado actual e seguinte, respectivamente. A acção do sistema é representada por $U_k \in R^n$ e função f pode ser escolhida de modo a conseguir estimar o estado seguinte X_{k+1}, através do estado actual X_{k+1}, e a sua acção U_k. A este resultados acresce-se um ruído branco w_k com distribuição de média nula e covariância Q_t. O modelo dinâmico f é descrito pelas equações de 2. Tendo:

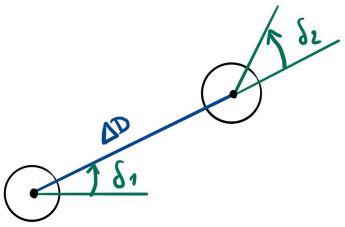


Figura 1: Modelo do movimento

, então o modelo dinâmico é representado por:

$$\begin{cases} x_{k+1} = x_k + \Delta D cos(\theta_k) + \delta_1 \\ y_{k+1} = y_k + \Delta D sin(\theta_k) + \delta_1 \quad k = 0, 1, ... \\ \theta_{k+1} = \theta_k + \delta_1 + \delta_2 \end{cases} \tag{2}$$

sendo assim f uma matriz.

B. Modelo De Observação

As medições do projecto são efectuadas através do laser e com a utilização do algoritmo ICP.

$$\left\{ \; Z_k = h(X_k) + w_k \quad k = 0, 1, ..., . \right. \tag{3}$$

Onde $Z = (\bar{x}, \bar{y}, \bar{\theta})$
O modelo de medida é representado por $z_k \in R^m$, dependendo da pose do estado actual e ao qual se acresce um ruído branco

v_k, também de média nula e covariância R_k. A função h é então escolhida de modo a calcular a medida prevista Z_k segundo o estado previsto na equação anterior, X_k. Mais para a frente veremos que a função $h(X_k) = X_k$ e que consequentemente a matrix jacobiana de h, H é a matriz identidade.

C. Extended Kalman Filter

O *Extended Kalman Filter* é um algoritmo de estimação que permite estimar a pose de sistemas dinâmicos não lineares, avaliando os seus modelos dinâmicos e de observação. Este algoritmo recorrendo às equações:

$$\begin{cases} \mu_t = \bar{\mu}_t + \sum_i K_t^i (Z_t{}^{i\prime} - \hat{Z}_t^i) \\ \Sigma_t = (I - \sum_i K_t^i H_t^i)\bar{\Sigma}_t \end{cases} \quad t = 0, 1, ..., . \tag{4}$$

Sendo μ a media da distribuição da estima da pose do sistema, ie, a estimativa da posição. Σ a variância da distribuição, ie, a incerteza da estima da pose do sistema. H consiste na matriz jacobiana do modelo de observação e K corresponde ao *Kalman Gain*. $\bar{\mu}$ e $\bar{\Sigma}$ correspondem, respectivamente, à média estimada no instante anterior e à sua variância. $\hat{z}_t^i$ corresponde À observação obtida através da ultima posição feita pelo *Prediction*.

Este algoritmo é consistido por 2 passos: o *Prediction* e o *Update*.

1) Predicton: A *Prediction* no algoritmo do EKF pretende estimar a pose do robô consoante a pose que foi estimada no instante anterior, os dados recebidos pela odometria e o modelo de observação, isto é, o mapa. Trata-se de uma localização relativa: conhecendo a posição inicial do robô no referencial do mapa é possível estimar a sua posição ao longo do tempo no mapa, utilizando o modelo dinâmico expresso em 2.

Esta estimativa é sujeita a erros sistemáticos relacionados com erros nos sensores utilizado, e erros não sistemáticos relacionados com imperfeições do piso e certas manobras do robô que envolvam velocidade ou alterações bruscas de trajectória.

É importante referir também que odometria é expressada no referencial do robô, considerando a sua pose inicial como (0,0,0). É portanto necessário expressar a odometria no referencial do mapa. O tópico "inicialpose"indica a posição inicial do robô no referencial do mapa, adicionando este valor à pose estimada pela odometria consegue-se estimar a posição do robô no referencial do mapa. Desta maneira é relacionada a odometria com o modelo de observação. Uma vez que os dados da odometria, obtidos através do *rosbag*, estão no sistema de coordenadas cartesiano, estes têm de ser transformados para o sistema de coordenadas do tipo (x,y,θ) [4].

2) Update: O *Update* no algoritmo do EKF permite obter a nova estimação da posição, z_t' fazendo o *matching* entre a informação da observação efectuada pelo laser com o mapa e o modelo de observação proveniente do *Predict*. Tal como é enunciado em [2], o EKF assume que o mapa é representado por uma colecção de *features* e que a cada instante t, o laser observa uma determinada gama destas *features* conseguindo se localizar no mapa. De modo a efectuar isso, é necessário definir o que corresponde ao *Observation ou Measurement Model*. Assume-se portanto que $z_t' = (x_t', y_t', \theta_t')$ corresponde à transformação que é preciso aplicar aos pontos do laser de forma a alinha-los com o mapa. Posto isto e de acordo com a formulação anterior, o *Update* deixa de ser um problema de EFK, podendo-se considerar agora o algoritmo de Kalman Filter. Tomando estas considerações, pode-se concluir por fim que tanto o modelo de medida h corresponde à pose estimada pelo prediction e a sua matriz jacobiana H corresponde à matriz identidade.

Definido ao que corresponde o *Measurement Model*, é necessário realizar o *matching* entre a informação obtida do laser com o mapa, utilizando para isso o ICP descrito na secção seguinte. Pode-se concluir finalmente que a estimativa da pose do robô efectuado o pelo *update* corresponde à seguinte equação:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} \bar{x_t} \\ \bar{y_t} \\ \bar{\theta_t} \end{bmatrix} + K_t \left(\begin{bmatrix} x_t' \\ y_t' \\ \theta_t' \end{bmatrix} - \begin{bmatrix} \bar{x_t} \\ \bar{y_t} \\ \bar{\theta_t} \end{bmatrix} \right) \tag{5}$$

sendo que K_t corresponde ao ganho de Kalman Filter e

$$\begin{bmatrix} \bar{x_t} \\ \bar{y_t} \\ \bar{\theta_t} \end{bmatrix} \tag{6}$$

corresponde à estimativa da pose do robô vinda do *Predict*.

D. Interactive Closest Point

O *Iteractive Closest Point* [3] (ICP) é um algoritmo utilizado para minimizar a diferença entre duas *Point Clouds*. De modo a conseguir sobrepor o máximo de pontos de uma *Point Cloud* para a outra, o ICP define iterativamente uma transformação com a translação e rotação dos pontos, necessária para minimizar o erro métrico entre as duas.

Existem 4 passos importantes neste algoritmo:

_ para cada ponto ou grupo de pontos na *Point Cloud* de origem, associar o ponto ou grupo de pontos mais próximo(s) na *Point Cloud* de referência;

_ estimar e escolher a rotação e translação entre os pares escolhidos no passo anterior que minimizam o erro métrico entre as *Point Clouds*;

_ transformar todos os pontos de origem com essa transformação;

_ repetir.

Este algoritmo recebe duas *Point Clouds* e uma matriz de transformação de coordenadas que transforma uma das *Point Clouds* caso ambas não estejam no mesmo referencial.

Para usar este algoritmo, como nosso modelo de observação, é necessária existência de um mapa do ambiente em que o robô se movimenta. Este mapa é construído com o package GMapping do ROS que constrói um Occupancy Grid Map do ambiente. Para este mapa pudesse ser utilizado pelo ICP é necessário transformá-lo numa *Point Cloud* em 2D. Essa

transformação consiste em guardar apenas as coordenadas dos pontos correspondentes a zonas ocupadas no mapa. A outra *Point Cloud* usada no ICP é obtida através das leituras do LiDAR. Cada leitura do lidar corresponde a um conjunto de pontos em coordenadas polares que correspondem a obstáculos detectados. Este conjunto de pontos foi transformado em coordenadas cartesianas para ser usado no ICP.
Uma vez que a *Point Cloud* obtida com o LiDAR se encontra nas coordenadas do robô é necessário transformá-la para as coordenadas do mapa, isso é realizado pelo algoritmo sendo apenas necessário passar como argumento a posição actual do robô. O algoritmo retorna uma matriz correspondente à transformação afim, 7, que melhor regista os pontos obtidos pelo LiDAR no mapa. Esta matriz contem em si portanto uma estimativa da posição e orientação do robô.
A função implementada para a realização do ICP aplica à posição anterior do robô a transformação proveniente do *matching*, o ICP retorna uma matriz de transformação do tipo:

$$\begin{bmatrix} cos(\delta\theta) & -sin(\delta\theta) & \delta x \\ sin(\delta\theta) & cos(\delta\theta) & \delta y \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

sendo que a estimativa da posição do robô no mapa proveniente do *matching*, como visto na secção anterior, é

$$\begin{bmatrix} x'_t \\ y'_t \\ \theta'_t \end{bmatrix} = \begin{bmatrix} \delta x \\ \delta y \\ \delta\theta \end{bmatrix} + \delta_t, \tag{8}$$

com $\theta \in [0, 2\pi]$.

E. Robot Operating System

Para testar os algoritmos anteriormente especificados é necessário extrair os dados da odometria, os pontos do mapa e as leituras efectuadas pelo laser. Por esse motivo é utilizado o *Robot Operating System* (ROS), um programa de *open source* que permite ao utilizador testar o seu código para qualquer tipo de robôs desde que este funcione com o ROS, concentrando-se inteiramente no software que deseja implementar e não nas especificações ou presença do hardware do robô. O ROS é composto por nós, sendo um nó correspondente a um executável que pode publicar e/ou subscrever a tópicos. Partindo do principio que um tópico funciona como uma variável, um nó pode tanto escrever valores nessa variável ou ler os valores dessa variável. É dessa forma que diversos nós comunicam uns com os outros. O programa ROS possui também várias bibliotecas que facilitam a execução do programa desenvolvido neste trabalho. Uma delas sendo o *rosbag*, um conjunto de ferramentas para gravar e reproduzir mensagens publicadas pelos diferentes tópicos durante a execução do programa no robô. Deste modo, o utilizador consegue testar o seu software sem ter que utilizar o robô, Outra biblioteca consiste no *Rviz*, permitindo a visualização virtual de dados como o plano do mapa, a pose do robô, as leituras do laser, etc.

III. Implementação

A. ROS

Como foi especificado anteriormente, para a implementação deste projecto foi utilizado o programa ROS melodic. As bibliotecas usadas foram o Rviz e Rosbag. Os vários rosbags efectuados ao longo do semestre, serviram fundamentalmente para legitimar o desempenho do projecto desenvolvido.

B. MBot, Mapa e laser

Para a realização deste projeto utilizou-se o *MBot* que é um robô omnidirecional. Esse tipo de robô tem máxima manobrabilidade no plano, isto é, o robô pode mover-se em qualquer direção sem necessidade de se reorientar [6].

Um grau de liberdade de um corpo são todas as opões de movimento que esse corpo tem, tanto de translação como de rotação. Neste tipo de problemas de robótica móvel, uma vez que o robô apenas se consegue movimentar no plano *xy*, apenas existem translações em *x* e *y*, e rotações em torno do eixo *z*, pelo que podemos concluir que estamos perante problemas com 3 graus de liberdade.

O sensor utilizado utilizado neste projeto é o Laser Range Finder. O LIDAR é um dispositivo que contêm um laser com o qual se pode medir a distância a um objeto. Normalmente, este tipo de dispositivo funciona com o método de tempo de voo direto ou com o método de mudança de fase. Cada laser utilizado tem 270º de *scanning range*, pelo que, como o *MBot* é constituído com 2 destes sensores, um na parte da frente e outro na parte de trás do robô, e tendo sido ambos utilizados, é possível recolher os dados sobre o ambiente total que o rodeia.

O objetivo deste projeto é estimar em tempo real a pose (posição + orientação) de um robô móvel num ambiente já conhecido. É utilizado por isso, para qualquer experiência efectuada, o mapa correspondente ao piso 8 da torre norte do técnico. É de salientar que o espaço de teste é um espaço real, onde utilizamos o *MBot* para realizar diversos rosbags para diferentes experiências efectuadas. Como já dito anteriormente, este é construído com o package GMapping do ROS que constrói um Occupancy Grid Map do ambiente.

C. EKF e ICP

Foi desenvolvido um programa em Python com a versão 3.6.8. O computador onde foram testados as diversas experiências efectuadas, tem as seguintes características: Intel-Core i7-6500U CPU @2.50GHz

Foram desenvolvidos dois scripts para a realização deste projeto:

- **EKF**, que se comporta como main do projeto e contém diversas funções para a realização correta do algoritmo Extended Kalman Filter.
- **icp2**, que realiza o *matching* entre os pontos oriundos do lazer e o mapa utilizado.

É preciso ter em conta que, a implementação do algoritmo ICP em Python for realizada partindo da implementação de Clay Flannigan [4]. Porém, esta implementação foi devidamente alterada para que, quando recebe-se duas *Point Clouds* de

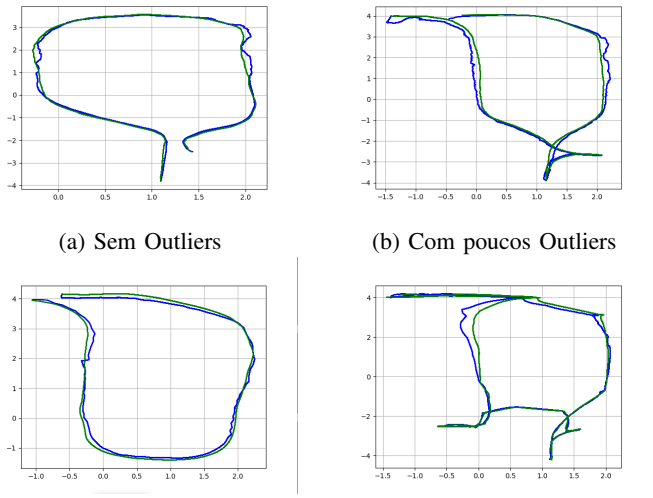
tamanhos diferentes, este as conseguisse processar e efectuar o *matching* entre ambas.

IV. Resultados Experimentais

Os resultados experimentais neste trabalho foram testados com recurso a vários *rosbags*, obtidos na mesma localização mas cada um com diferentes características como a trajectória, velocidades e/ou a existência de pontos *outliers* no ambiente em questão. Para cada um destes, registou-se os dados da odometria, mapa, laser e posição inicial do robô testando-se a performance do algoritmo através da medida do erro entre a localização estimada e a considerada como "ground-truth"(amcl).

A. Consistência dos Resultados

Uma vez posto o algoritmo ICP a funcionar, este devolve a localização (x'_t, y'_t, θ'_t) estimada através das leituras do laser. É então possível efectuar o *Update* com os dados da observação recorrendo à equação (5). Com o EKF implementado obtêm-se os resultados das figuras a, b, c e d de 2 para quatro trajectórias diferentes.Pode-se assim confirmar a consistência do programa desenvolvido, já que a performance deste não varia muito segundo as características do ambiente ou segundo as trajectórias escolhidas. Como se pode observar, o EKF converge rapidamente para a localização correcta do robô. Existem casos de pouca divergência, como no caso da figura 2d, devidos à presença de *outliers*, mas que no entanto é rapidamente corrigida.

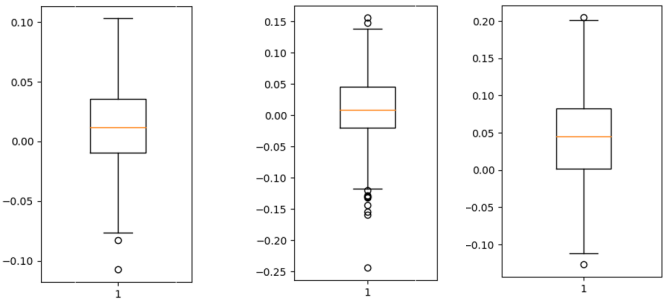


(a) Sem Outliers (b) Com poucos Outliers

(c) Com velocidade = 0.6 m/s (d) Com outliers Importantes

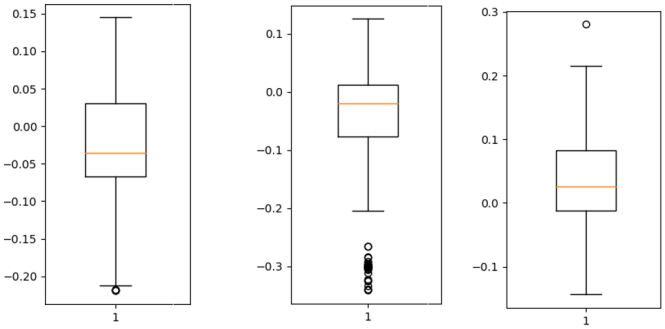
Figura 2: Localização (verde) amcl e (azul) ekf, para diferentes percursos

São também estudados os erros das estimações da pose para testar a precisão do programa na estimativa a localização.



(a) x (b) y (c) θ

Figura 3: Boxplot de x, y, θ para o caso da figura 2a



(a) x (b) y (c) θ

Figura 4: Boxplot de x, y, θ para o caso da figura 2b

Pode-se observar que os erros respectivos às trajectórias das figuras 2a e 2b, obtidos através do comando *boxplot*, entre o nosso algoritmo e o groud-truth(AMCL), são muito próximos de zero, mantendo-se na ordem de grandeza dos 0.10 metros para a maioria dos instantes.

B. Performance com e sem Outliers

A trajectória da figura 2a é efectuada num ambiente controlado, sem *outliers*, i.e., não existem objectos ou pessoas presente no piso 8 durante a gravação do *rosbag* que não estejam apresentado(a)s no plano do mapa. Desta maneira o laser não lê dados que não pertençam ao mapa do piso 8 e que podem ser mal interpretados pelo ICP, causando a divergência do EKF. O caso seguinte, apresentado na figura 5, consiste numa experiência onde existem *outliers*(pessoas e/ou objectos) que perturbam o correto funcionamento do ICP. Nesta zona onde são existentes, o ICP não obtém uma transformação exacta que ajuste na perfeição a leitura do laser com os pontos do mapa. De facto, as leituras do laser que não correspondem com os dados do mapa vão induzir erros que influenciaram a matriz de transformação entre a point cloud do laser e do mapa e consequentemente a estimativa da posição não será a correta, ficando o robot assim perdido naquela zona da sala.

Esta divergência é também observada através do erro apresentado na figura 6 . A experiência é feita de 0 a 540 segundos, pelo que os pontos a partir de 540s no gráfico não se alteram mais. No entanto pode-se observar que o erro da posição x e y, começa novamente a diminuir após a divergência, quando este

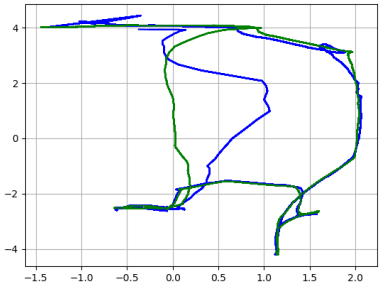


Figura 5: Localização do MBot com (azul)EKF implementado e (verde) AMCL

se desloca para uma zona sem outliers. Antes da divergência ocorrer observa-se que o erro para x, y e theta é praticamente nulo isto porque para o ângulo θ, o valor 6,3 que aparece no gráfico pode ser entendido com $\simeq 2\pi \simeq 0$. No entanto, embora se observe uma divergência do EKF resultante destes *outliers*, o algoritmo volta a convergir pouco a pouco ate voltar à trajectória correcta. Pode-se assim confirma o bom funcionamento do *Update* na correcção da posição prevista.

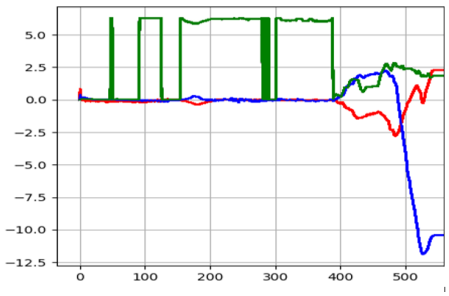


Figura 6: Erro do x (azul), y(vermelho) e θ(verde) da Localização do MBot entre EKF implementado e o AMCL

De modo a evitar que a divergência seja tão grande com a presença de *outliers*, é, não só aumentado o número de Iterações do ICP como também é diminuído o Threshold para minimizar o erro da junção das point cloud. A estimação da pose do robô melhora, como se pode observar na figura 7 onde a diferença entre a estimativa da sua pose e a "ground-truth"(AMCL) é muito mais reduzida.

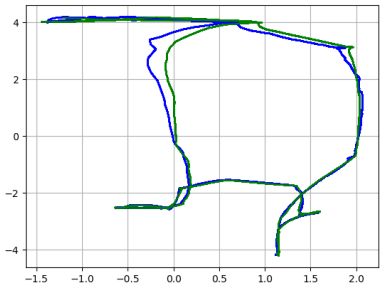


Figura 7: Localização do MBot com (azul) EKF implementado e (verde) AMCL com alteração dos parametros do ICP

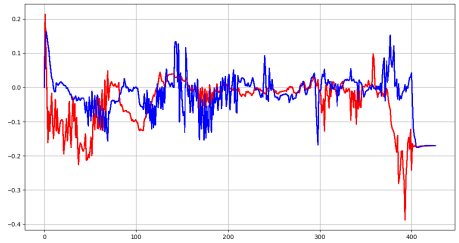
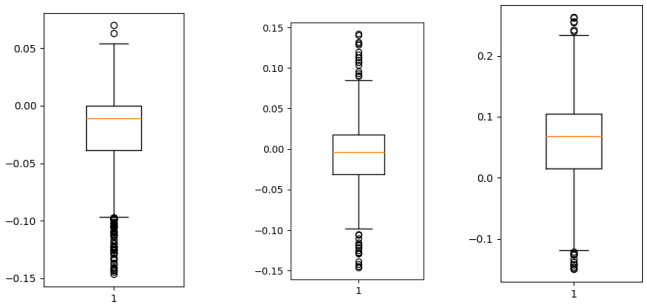


Figura 8: Erro do x (azul), y(vermelho) da Localização do MBot entre EKF implementado e o AMCL



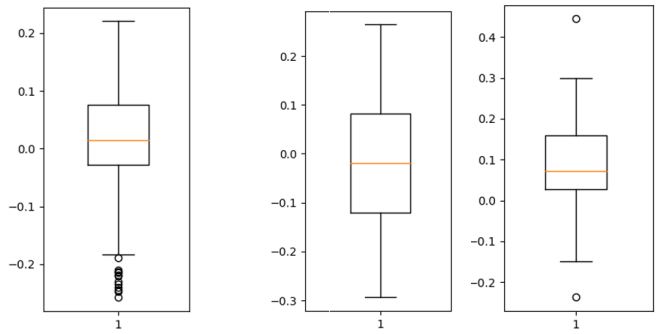
(a) x (b) y (c) θ

Figura 9: Boxplot de x, y, θ para o caso da figura 7

Os erros da trajectória sujeita a *outliers* diminuem quando alteramos os parâmetros do ICP. De facto , isto pode-se observar na diferença da trajectória estimada entre a figura 6 sem alterações dos parâmetros do ICP, e a figura 8 com mais restrições no ICP. Observa-se, na mesma na figura 8, uma ligeira divergência a partir dos 400 segundos que volta a convergir antes do *rosbag* acabar. Os erros mantêm-se, em média, na ordem de grandeza dos 0,2 metros. Através do boxplot para x, y e θ na figura 9, pode-se concluir que com outliers existe um erro maior que nos boxplot estudados para as trajectórias 2a e 2b na figura 3, que não incluem *outliers* predominantes.

C. Performance com e sem Velocidade

De maneira a testar a robustez do programa desenvolvido face à velocidade em que a trajectória é efectuada, foram feitas duas experiências, uma com velocidade média $v_{avg} = 0.8m/s$, correspondendo à trajectória da figura 2a e outra com uma velocidade média $v_{avg} = 0.6m/s$, correspondendo à trajectória da figura 2c. As trajectórias escolhidas para estas duas experiências são semelhantes, consistindo numa volta ao mapa. Os resultados do erro obtidos para a experiência com velocidade maior estão apresentados na figura 10.



(a) x (b) y (c) θ

Figura 10: Boxplot de x, y, θ para o caso da figura 2c

Comparando ao erro obtido em 3, observa-se que a mediana do erro aumenta de 0.01 em x, 0.02 em y e 0.02 em θ. A distribuição dos quartis aumenta para mais que o dobro do erro em y. O número de valores mal estimados para x aumenta consideravelmente. Pode-se assim concluir que a precisão da localização diminui com o aumento da velocidade. No entanto, como observado na figura 2c, este erro não impede o funcionamento do programa uma vez que este efectua a localização na mesma, não se desviando muito da localização de referência.

D. Kidnapped Problem

Como forma a testar a robustes do nosso algoritmo num caso extremo procedeu-se à realização do Kidnapped problem. Neste problema foi fornecida ao robot uma pose inicial que não correspondia à sua pose real. Para a análise destes resultados, é importante referir a pose real inicial bem como a sua pose inicial dada, que será usada para alimentar tanto o EKF como o ICP nas iterações iniciais. As posições são as seguintes:

- pose real ($x_{real} = -1.58, y_{real} = 4.08, \theta_{real} \simeq 0$)
- pose dada ($x_{dado} = -1.35, y_{dado} = -1.32, \theta_{dado} \simeq 0$)

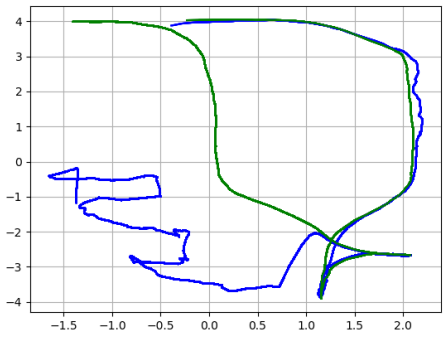


Figura 11: Localização do MBot com EKF implementado (azul) e AMCL (verde) - Kidnapped Problem

O resultado deste problema pode ser verificado na figura 11. Apesar de ser considerada uma posição inicial errada para o robot, este foi capaz de convergir para a sua posição real (dada pelo AMCL). Ainda que este exemplo em concreto convirga, não se pode assumir que o algoritmo é robusto para todos os problemas deste tipo. Isto prende-se pelo facto de que existência, por exemplo, de outliers pode impedir a convergência para a sua posição real.

V. Future Work

Como forma de melhorar o algoritmo desenvolvido, é de especial importancia a inclusão de um algoritmo de remoção de outliers. Não obstante a realização de mais problemas de Kidnapping deve ser realizada, tanto no ambiente onde foram realizadas as experiências como em ambientes novos, considerando a inclusão ou não de outliers, assim como novas trajectórias. Uma comparação deve ser realizada entre experiências

VI. Conclusão

Depois de analisar todos os resultados, existem diversas conclusões a ser retiradas.

Efectuaram-se diversos *rosbags* para testar e legitimar o desempenho do algoritmo EKF desenvolvido. Pode-se observar pelos resultados experimentais uma rápida convergência para a localização correta do robô no mapa, que confirma a consistência do programa desenvolvido. Em média, o erro da estimativa da posição do robô anda na ordem dos 0.15m o que é sensivelmente pouco.

Porém, possíveis perturbações regulares do dia-a-dia no ambiente prejudicam o cálculo da estimativa da posição do robô no mapa, uma vez que, a presença de outliers perturbam o funcionamento do ICP. Uma maneira de solucionar este tipo de problemas consiste em aumentar o número de iterações do ICP e aumentar o threshold para o erro da junção das point clouds. Outra maneira de solucionar este tipo de problemas é através da utilização do ransac, que é um método iterativo que permite a eliminação de outliers.

Pode-se observar também que o algoritmo desenvolvido mostra robustez face ao aumento da velocidade do robô, mesmo com a pequena diminuição da precisão da localização. Não obstante a realização de mais problemas de Kidnapping deve ser realizada, sendo comparado o efeito da implementação do ransac em experiências que sem o mesmo não convergiram.

Por fim, devido a tudo o que foi dito em cima e com os diversos resultados experimentais mostrados na secção anterior, fica provado a validade e a legitimidade do algoritmo desenvolvido ao longo do semestre.

Referências

[1] J. Sequeira, P. Lima, A. Saffiotti, V. Gonzalez-Pacheco, and M. A. Salichs, "MOnarCH: Multi-robot cognitive systems operating in hospitals," in ICRA 2013 Workshop on Many Robot Systems, 2013.

[2] Probabilistic Robotics: S. Thrun, W. Burgard e D. Fox 2005 MIT Press

[3] Uma Introdução aos Robôs Móveis, Dr. Humberto Secchi, Edição: Agosto de 2008