Project [I3]: Offboard LRF + EKF Localization

André Alves (nº 80881), Luís Almeida (nº 81232), Francisco Pereira (nº 81381) e Joaquim Silva (nº 81882)

Resumo—Neste paper é apresentado um sistema de localização de um robot utilizando apenas um LRF Offboard. Para a sua correta localização são utilizados dois principais algoritmos: EKF e Corner Detection. O problema abordado é baseado num caso real, a localização de um veículo autónomo do projeto ITER.

Index Terms—LRF, Offboard, EKF, Corner Detection, ITER.

1 Introdução e Motivação

Um dos grandes desafios nos dias de hoje é a produção de energia limpa, com pouco impacto no meio ambiente, satisfazendo, no entanto, o cada vez maior consumo global. Uma das formas de energia limpa e com maior potencial de escalabilidade é a "energia das estrelas" – a energia de fusão nuclear. O **ITER** é um dos maiores e mais ambiciosos projetos de engenharia no mundo. Trata-se da construção de um reator de fusão nuclear experimental em Cadarache, no sul de França, envolvendo a colaboração de 35 países.

Durante o tempo de vida deste reator, será necessária a manutenção do mesmo. Devido aos elevados níveis de radiação, essas operações terão de ser controladas remotamente e realizadas por um veículo autónomo, *Cask and Plug Remote Handling System*, cujo peso pode atingir 100 toneladas. Para que seja possível o transporte de cargas muito pesadas pelos corredores estreitos do edifício Tokamak do ITER, representados na Figura 1, com pequenas margens de segurança é necessário um sistema de localização muito preciso. Para que isso seja possível recorre-se a um Laser Range Finder, que devido às condições do meio, será colocado *offboard*, ou seja, em sítios fixos do edifício (Figura 1), pois as suas medições e componentes eletrónicos são afetados pela radiação a que se encontram expostos.

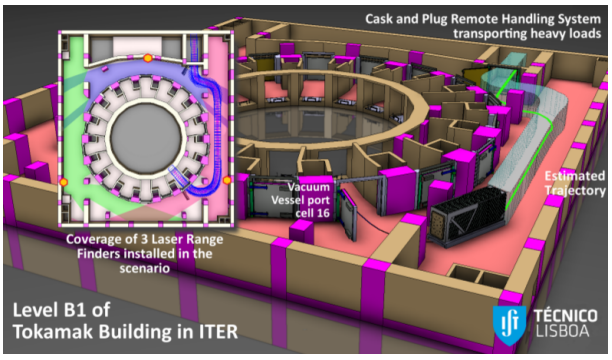

Figura 1. Tokamak Building - ITER [2]

O nosso trabalho consiste na localização em tempo real de um robot com uma escala de 1:25 relativamente ao original, através de um Laser Range Finder *offboard*, com o qual se obtêm as coordenadas polares dos objetos detetados. Estas serão trabalhadas no algoritmo de deteção de cantos, de forma a aplicar posteriormente o algoritmo **EKF** (Extended Kalman Filter) para efetivamente obter a sua posição e orientação.

Este paper está organizado da seguinte forma: a Secção 1 introduziu o problema abordado. A Secção 2 aborda os Métodos e Algoritmos usados durante o projeto, assim como o LRF (Subsecção 2.1), o Algoritmo de Deteção de Cantos (Subsecção 2.2) e o EKF (Subsecção 2.3). Na Secção 3 são descritos os detalhes das implementações do Controlo do Robot NXT (Subsecção 3.1) e do EKF (Subsecção 3.2). Por fim, são apresentados os Resultados Experimentais na Secção 4 e as Conclusões sobre o projeto na Secção 5.

2 Métodos e Algoritmos

2.1 Laser Range Finder (LRF)

O *Laser Range Finder* é um dispositivo que emite radiação infravermelha e quando essa radiação é refletida numa superfície, sabendo-se o tempo que "viajou" e a velocidade da luz, retorna uma distância que indica com precisão o quão afastado está o objeto do *laser*. O LRF tem um espelho interno que permite a emissão de múltiplos raios espaçados de $\Delta\theta = 0.36^{o}$ (incremento angular) permitindo fazer-se um *scan* completo da área desejada, limitado apenas pelo *range* angular máximo do *LRF* usado, que, neste caso, foi 90^{o}.

Preferiu-se a escolha do laser *Hokuyo* ao invés do *Sick*, uma vez que o alcance deste, 4 metros, é suficiente para as dimensões do nosso problema e o seu uso é relativamente mais fácil. Através dos dados recolhidos pelo *Hokuyo URG-04LX-UG01* obtêm-se as coordenadas polares dos vários objetos detetados pelo *laser*, mas para tal, é necessário que exista um *node publisher* no tópico */scan* para que posteriormente se consiga aceder aos dados através de um *node subscriber*.

2.2 Algoritmo de Deteção de Cantos

O veículo autónomo usado pelo ITER contém nos seus quatro cantos, marcadores que serão aplicados ao EKF. Para simular este caso é necessário usar um algoritmo que detete os cantos do robot. O algoritmo descrito tem como base o algoritmo Recursive Line Fitting descrito em [4].

Este algoritmo tem como entrada as coordenadas polares dadas pelo LRF e apresenta como resultado as coordenadas cartesianas dos quatro cantos do robot, que, posteriormente inseridas no algoritmo do EKF, darão a *pose* do robot. O algoritmo é descrito pelos seguintes passos:

1) Com o LRF, obter dados do espaço vazio (*background*), i.e., sem o robot estar presente.

$$d_{without} = [d^{(1)}_{without} \dots d^{(n)}_{without}] \tag{1}$$

2) Com o LRF, obter dados do espaço, desta vez já com o robot presente.

$$d_{with} = [d^{(1)}_{with} \dots d^{(n)}_{with}] \tag{2}$$

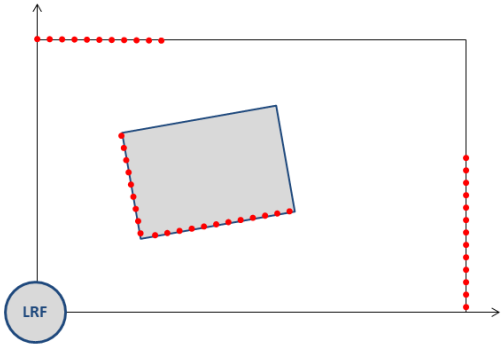

Figura 2. Pontos obtidos já com o robot.

3) Fazer a diferença entre os dois vetores. Desta forma, os dados referentes *background* serão nulos.

$$d_{dif} = d_{with} - d_{without} = [d^{(1)}_{dif} \dots d^{(n)}_{dif}] \tag{3}$$

4) Eliminar componentes ruídosas e *outliers*.
   - Caso um elemento do vetor de diferenças seja inferior a um certo *threshold* (muito pequeno), coloca-se esse elemento a zero.

$$d^{(i)}_{dif} \leq \epsilon \Rightarrow d^{(i)}_{dif} = 0 \tag{4}$$

   - Caso no vetor de diferenças haja um ponto não nulo cujos vizinhos são iguais a zero, então esse ponto é um *outlier* e portanto será colocado a zero.

$$d^{(i-1)}_{dif} = 0 = d^{(i+1)}_{dif} \wedge d^{(i)}_{dif} \neq 0 \Rightarrow d^{(i)}_{dif} = 0 \tag{5}$$

   - Caso no grupo de *foreground* haja um ponto nulo cujos vizinhos são diferentes de zero, então esse ponto é um *outlier* e portanto esse ponto será igual à média dos vizinhos.

$$d^{(i-1)}_{dif} \neq 0 \neq d^{(i+1)}_{dif} \wedge d^{(i)}_{dif} = 0 \Rightarrow d^{(i)}_{dif} = \frac{d^{(i-1)}_{dif} + d^{(i+1)}_{dif}}{2} \tag{6}$$

5) Obter o primeiro e último elementos não nulos do vetor das diferenças, que são pontos do *foreground*.
6) Eliminar grupos de pontos do *foreground* que não pertençam ao robot, que é o maior grupo de pontos do *foreground*.
7) Converter índices dos pontos do robot para ângulos.

$$\text{angle increment } (\Delta\theta) = \frac{\text{scan angle range } (90^o)}{\text{number of measurements}} \tag{7}$$

$$\text{angle } (\theta) = \Delta\theta \times i$$

8) Calcular as coordenadas cartesianas (x,y) dos pontos do robot.

$$\begin{aligned} x_i &= d^{(i)}_{dif} \cdot cos\theta \\ y_i &= d^{(i)}_{dif} \cdot sin\theta \end{aligned} \tag{8}$$

9) Obter primeiro (x_1, y_1) e último (x_n, y_n) pontos do grupo do robot. À partida, assume-se que estes dois pontos são cantos do robot.
10) Com esses dois pontos, traçar uma reta $y = m \cdot x + b$.

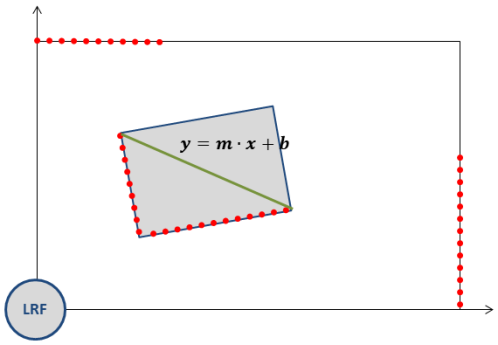

Figura 3. Reta (a verde) entre o primeiro e último pontos do robot detetados pelo LRF.

$$\begin{aligned} m &= \frac{y_n - y_1}{x_n - x_1} \\ b &= y_n - m \cdot x_n \end{aligned} \tag{9}$$

11) Traçar uma reta perpendicular $y = m_i^{\perp} \cdot x + b_i^{\perp}$, da qual faz parte o ponto *i*, à reta da alínea anterior.

$$\begin{aligned} m_i^{\perp} &= -\frac{1}{m} \\ b_i^{\perp} &= y_i - m_{perp} \cdot x_i \end{aligned} \tag{10}$$

12) Calcular os pontos de interseção das duas retas.

$$\begin{aligned} x_i^{inter} &= \frac{b_i^{\perp} - b}{m - m_i^{\perp}} \\ y_i^{inter} &= m_i^{\perp} \cdot x_i^{inter} + b_i^{\perp} \end{aligned} \tag{11}$$

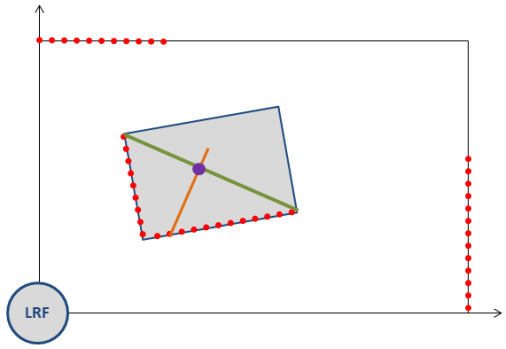

Figura 4. Exemplo de uma das retas perpendiculares (a laranja) à reta anterior que passa pelo ponto (x_i, y_i) e ponto de interseção entre as duas retas $(x_i^{inter}, y_i^{inter})$ (a roxo).

13) Calcular distância entre o ponto medido (x_i, y_i) e o ponto de interseção correspondente $(x_i^{inter}, y_i^{inter})$.

$$d_i = \sqrt{(x_i^{inter} - x_i)^2 + (y_i^{inter} - y_i)^2} \tag{12}$$

14) Se a maioria dessas distâncias for maior que um certo *treshold*, a maior distância corresponde ao ponto de um canto do robot. Assim, sabendo as coordenadas de três cantos e as dimensões do robot, pode-se calcular o quarto canto. Caso contrário, o LRF apenas "vê" um lado do robot, e os previamente assumidos cantos do robot são, efetivamente, os cantos do robot.

Neste algoritmo recorreu-se a um algoritmo *Breakpoint Detector* [5], de forma que se começou por desenvolver um algoritmo que tivesse em conta a distância entre os pontos consecutivos do grupo do robot, e que ao comparar com um certo *treshold*, para as distâncias que fossem superiores a esse valor, os pontos correspondentes não eram considerados pontos do robot. Esta comparação é apenas realizada para os primeiros e últimos pontos do robot, ou seja, nos seus limites.

2.3 Extended Kalman Filter (EKF)

O algoritmo EKF tem como objetivo a determinação das coordenadas cartesianas do centro do veículo assim como a sua orientação relativamente a um eixo de referência - *pose*. O resultado da aplicação do algoritmo é um vetor de estado $x = [x, y, \theta]$. O cálculo destas variáveis baseia-se na atualização sucessiva dos parâmetros de uma distribuição gaussiana onde a sua média indica o valor mais provável do vetor x e a medida de incerteza acerca do estado é expressa na variância da distribuição.

A atualização destes parâmetros da distribuição gaussiana é realizada nas etapas *Prediction*, em que se obtém uma predição para a *pose* do robot, dada ao determinar-se os seus cantos, e *Update*, em que a predição é atualizada tendo em conta os dados obtidos através do algoritmo de deteção de cantos. Na fase *Matching* é feita a validação entre os resultados da previsão do *observation model* e do *action model*.

As equações deste algoritmo resultam da adaptação do Kalman Filter para sistemas não lineares, sendo que as matrizes apresentadas de seguida correspondem a linearizações locais. As equações utilizadas na fase *Prediction* são apresentadas em (13).

$$\begin{aligned} &\overline{\mu}_t = g\big(\overline{\mu}_{t-1}, u\big), \\ &\overline{\sum_t} = G_t^x \sum_{t-1} (G_t^x)^T + R_t, \end{aligned} \tag{13}$$

onde $\overline{\mu}_t$ e $\overline{\sum}_t$ correspondem ao valor esperado e covariância da gaussiana que representa o estado x; g é a função associada ao *action model* e G a sua Jacobiana. O termo R_t inclui os erros associados aos dados de odometria do robot.

Depois de apresentado o resultado da fase *Prediction*, a predição atual da *pose* do veículo é atualizada tendo em conta as observações do LRF. Este passo considera-se, assim, uma média ponderada, refletida pelo Kalman Gain (K_t), entre os dados das observações do laser e os dados de odometria do robot, sendo as equações utilizadas apresentadas em (14).

$$\begin{aligned} K_t &= \overline{\sum_{t-1}} (H_t^x)^T \Big((H_t^x)^T \overline{\sum_{t-1}} H_t^x + Q_t \Big)^{-1}, \\ \mu_t &= \overline{\mu_t} + K_t \big[z_t - h\big(\overline{\mu_t}\big) \big], \\ \sum_t &= \overline{\sum_{t-1}} - K_t \Big((H_t^x)^T \overline{\sum_{t-1}} H_t^x + Q_t \Big) (K_t)^T, \end{aligned} \tag{14}$$

onde Q_t é a matriz de covariância que inclui os erros de observação, h é a matriz associada ao *observation model* e H a sua Jacobiana. Verifica-se que se a matriz Q contribuir com maior peso relativamente ao termo $(H_t^x)^T \overline{\sum}_{t-1} H_t^x$, o *Kalman Gain* tenderá a anular-se pelo que a estimação do estado x será determinada pela estimativa do *action model*.

3 IMPLEMENTAÇÃO

3.1 Controlo do Robot NXT

Para facilitar o controlo do *Robot NXT* durante as experiências efetuadas (tanto para o cálculo das matrizes de covariância como para efetuar a sua localização em tempo real), houve a necessidade de criar um programa em *Matlab* que permitisse controlar o ângulo das rodas e a sua velocidade através do teclado.

Um dos objetivos do nosso trabalho era calcular a posição do Robot em tempo real, mas devido à impossibilidade de se conseguir criar uma conexão por via *Bluetooth* no sistema operativo *Linux* e consequentemente, a respetiva publicação dos dados de odometria por um *node publisher*, houve a necessidade de criar um ficheiro *.txt* para armazenar os dados de odometria utilizado no nosso algoritmo.

De modo a recolher-se os dados de odometria do robot é necessário utilizar-se as funções que permitem o controlo das velocidades angulares e ângulos das rodas e a posterior leitura desses dados. Implementaram-se *scripts* em *Matlab* que efetuavam cada uma dessas funções separadamente e o objetivo seria ler-se os dados de odometria enquanto simultaneamente se controlava a trajetória do robot. Uma vez que o *script* que controla a trajetória do veículo utilizava uma janela GUI - *Graphical User Interface* -, não nos foi possível executar o *script* de leitura dos dados de odometria e consequente escrita num ficheiro *.txt* em simultâneo. A solução encontrada foi introduzir-se um comando que pausa a execução do *script* de armazenamento dos dados no ficheiro durante 100 ms - *pause(0.1)* - o que permitiu a execução em paralelo dos programas. Para além de se registar as velocidades angulares (em rotações por minuto) e os ângulos das rodas do veículo (em graus), escreve-se no ficheiro também os instantes em que se fez a leitura uma vez essa informação é relevante para a análise da frequência da aquisição dos dados.

3.2 Implementação do EKF

De forma a localizar o Robot NXT com a melhor precisão possível, conciliaram-se os dados obtidos pelo *LRF* e os dados provenientes da odometria do *Robot NXT*. Para tal, procedeu-se à implementação do algoritmo *Extended Kalman Filter*, descrito anteriormente na Subsecção 2.3. Nesta subsecção apresenta-se de que forma se aplicou o conhecimento teórico do algoritmo EFK ao problema a tratar, definindo-se o *action model*, o *observation model* e de que forma se tratou a incerteza destes modelos.

Apesar do vetor de estado $x = [x, y, \theta]$ ser o vetor que indica a *pose* do robot, na atualização da posição e orientação do veículo feita no EKF determina-se a evolução espacial da posição de cada canto - par (x, y) - sendo que sabendo a localização dos quatro cantos do robot é possível determinar-se as coordenadas do seu centro de massa e a sua orientação relativamente ao eixo horizontal, definido como a referência.

3.2.1 Action model

A formulação do ***action model*** , i.e., a função de cinemática do robot é apresentada em (15), onde W é a distância entre as rodas e t_{od} é o instante em que o comando $u_{tod} = [\theta_F \; V_F \; \theta_R \; V_R]^T$ foi aplicado.

$$\overline{x}_t = \overline{x}_{t_{od}} + \frac{t - t_{od}}{2} \begin{bmatrix} V_F \cos(\theta_r^t + \theta_F) + V_R \cos(\theta_r^t + \theta_R) \\ V_F \sin(\theta_r^t + \theta_F) + V_R \sin(\theta_r^t + \theta_R) \\ \frac{V_F \sin(\theta_F) - V_R \sin(\theta_R)}{W/2}, \end{bmatrix} \quad (15)$$

onde V_F e V_R são as velocidades angulares da roda dianteira e traseira, respetivamente e θ_F e θ_R os seus ângulos.

Esta expressão representa um integrador na medida em que transforma a velocidade linear das rodas (obtida a partir da velocidade angular e do diâmetro das rodas) numa distância que é uma função crescente com o tempo. Este deslocamento segundo as direções horizontal e vertical é adicionado à posição anterior, obtendo-se uma estimativa para o próximo estado do robot. Os dados são fornecidos através da leitura de um ficheiro *.txt*.

3.2.2 Observation model

O *observation model* permite mapear o estado x para o espaço de observações. A matriz h utilizada em (14) é definida em (17).

$$h_i = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \tan(\frac{y}{x}) \\ \phi \end{bmatrix} \quad (16)$$

onde x e y correspondem às coordenadas x e y de cada canto do veículo, ϕ indica a orientação do mesmo e o índice i indica o canto. A matriz utilizada na fase *Update* do EKF é a concatenação de quatro vetores h, cada um contendo informação acerca de cada canto, sendo definida em (17).

$$h = \begin{bmatrix} h_1 \; h_2 \; h_3 \; h_4 \end{bmatrix} \quad (17)$$

Na Figura 5 apresenta-se uma ilustração exemplificativa da implementação do Extended Kalman Filter.

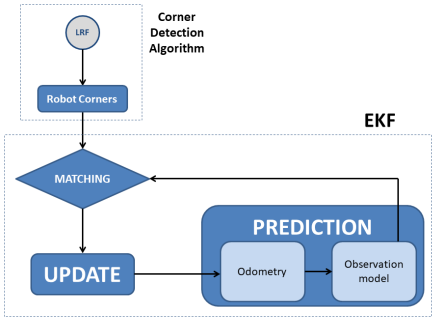

Figura 5. Resumo da implementação do EKF.

3.2.3 Incerteza: Matrizes de Covariância

Para a estimação do estado x é necessário considerar-se o efeito de dados ruidosos. O algoritmo EKF fornece uma aproximação ótima para o estado na presença de ruído de processo e ruído de observações, assumindo que estes são independentes e com distribuição normal com média nula e matrizes de covariância R_t e Q_t, respetivamente.

No *Prediction Step* do algoritmo considera-se o cálculo da matriz de covariância do erro de processo que está associado aos erros de odometria. Estes erros incluem os erros de derrapagem das rodas assim como a sua vibração. Na determinação desta matriz de covariância realizou-se uma experiência com o robot ITER para se avaliar a credibilidade dos dados recolhidos de odometria. Através dos dados obtidos do robot ao realizar uma trajetória retilínea – velocidades angulares de ambas as rodas e os respetivos ângulos – calculou-se a distância experimental correspondente através da fórmula $v = w.r$, sendo r o raio da roda do veículo, em metros, e w a sua velocidade angular - rad/s.

Após um número significativo de testes, ao comparar-se esta distância experimental com a distância real percorrida pelo robot, calculou-se a média das distâncias experimentais determinadas e a variância foi definida através do valor médio do quadrado dos desvios das distâncias experimentais em relação à média. A variância determinada é $\sigma_x^2 = 1.034$ m e é idêntica para ambas as coordenadas x e y, sendo que a variância da orientação do robot, θ, foi também determinada. A trajetória retilínea realizada pelo robot apresentou um desvio significativo em relação ao eixo de referência sendo que este ângulo foi calculado com base nas distância real percorrida segundo as direções dos eixos coordenados xy. Através da fórmula $\tan(\theta)$ e do valor das distâncias que correspondem aos catetos do triângulo retângulo definido, calculou-se a orientação do robot que idealmente seria nula, face à experiência feita. A variância da orientação do robot calculada é $\sigma_\phi^2 = 0.0037 rad$. Os valores determinados são integrados na atualização do estado do robot através da matriz R_t, definida em (18).

$$R_t = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\phi^2 \end{bmatrix} \quad (18)$$

Também as medições do *laser* apresentam ruído pelo que é necessário incluir esta informação no EKF. Ao utilizar-se o *laser scanner* assume-se que a cada medição está associada uma incerteza quanto à distância detetada pelo *laser* assim como quanto ao ângulo correspondente. Após analisar-se o *datasheet* do dispositivo, verificou-se que a incerteza radial do mesmo era uma função da distância radial sendo que para valores próximos dos 4 m - distância próxima do alcance máximo utilizado - era razoável considerar-se a incerteza radial 3% da distância detetada, sendo $\sigma_\rho^2 = 0.12$ m. A variância angular de uma medição - um varrimento completo (*scan*) inclui múltiplas medições - depende da abertura angular do varrimento assim como do número de medições por *scan*. O valor de incerteza angular que se considerou é próximo de $\sigma_\theta^2 = 0.76$ μrad o que indica que esta incerteza é modelada por uma distribuição normal "estreita". A matriz de covariância Q_t é definida em (19) e na Figura 6 apresenta-se uma representação do *sensor model*.

$$Q_t = \begin{bmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & \sigma_\phi^2 \end{bmatrix} \quad (19)$$

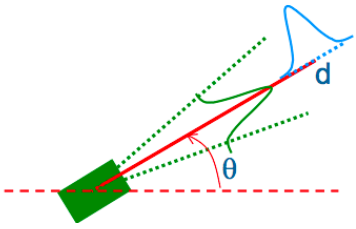

Figura 6. *sensor model* - incerteza radial e angular

3.2.4 Matching

Ao fazer-se a estimação dos quatro cantos em cada iteração do EKF é necessário fazer a correspondência entre o estado x de cada canto e o resultado do algoritmo de deteção de cantos, uma vez que o número de cada canto - 1º canto, 2º canto,... - pode variar de iteração para iteração. Assim, foi desenvolvida uma função que corrige a matriz z tendo

em conta a distância mínima entre cada canto e a estimação anterior. Cada coluna da matriz z tem informação acerca das coordenadas (x, y) do canto estimado pelo algoritmo de deteção de cantos. Determina-se para cada ponto (x, y) da coluna i da matriz z a distância euclidiana ao ponto (x, y) da coluna j de x. A coluna corrigida i de z irá corresponder à coluna j de x cuja distância calculada é mínima. De seguida repete-se o procedimento para as quatro colunas de z, incrementando-se a variável i. A Figura 7 é uma representação gráfica deste método.

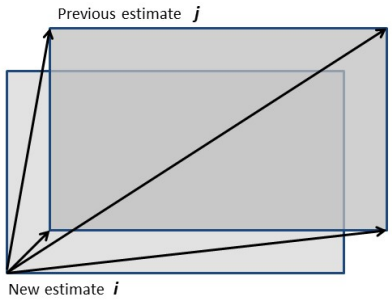

Figura 7. Correspondência entre cantos - correção

Uma vez corrigida a matriz z pode ser feita a verificação se as observações podem ser validadas ao serem comparadas com as suas previsões dadas por h.

4 Resultados Experimentais

Ao longo da execução do projeto a aplicação RViz foi a principal fonte de verificação do algoritmo de deteção de cantos e do EKF. Toda a análise e correção de ambos os algoritmos efetuou-se guardando os dados das experiências realizadas através do comando *rosbag record /scan*. De modo a analisar-se a evolução da posição dos cantos criou-se uma função que permitia a visualização dos mesmos no RViz, através da publicação das coordenadas para o tópico */marker*. Utilizando uma função que cria retas entre dois pontos foi possível simular no RViz o veículo e a sua evolução temporal estimada pelo algoritmo de deteção de cantos, simultaneamente com as medições do *laser scanner*. Na Figura 8 apresenta-se uma imagem retirada do RViz que ilustra o resultado do algoritmo de deteção de cantos assim como os pontos detetados pelo *laser*.

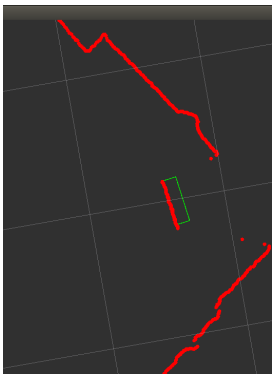

Figura 8. Visualização do veículo e medições do *laser* através da publicação nos tópicos */marker* e */scan*

Mostra-se também na Figura 9 a estimação da posição quer pelo algoritmo de deteção de cantos (retângulo verde) quer pelo EKF (retângulo azul).

Figura 9. Estimação da posição do veículo pelo EKF e o algoritmo de deteção de cantos.

Na Figura 9 observa-se um erro significativo entre as estimações de ambos os métodos.

A trajetória do veículo numa das experiências efetuadas é apresentada na Figura 10.

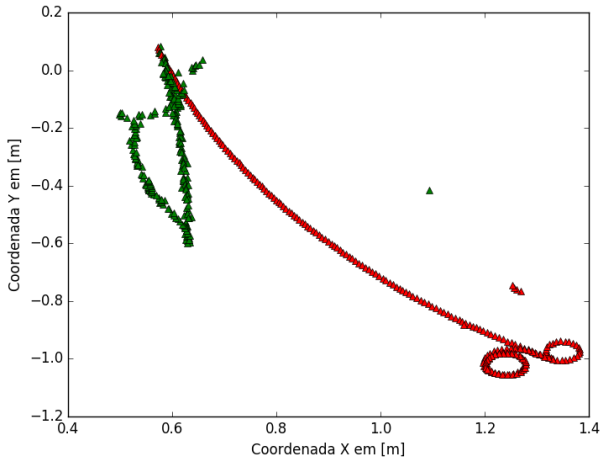

Figura 10. Comparação do centro de massa do robot estimado pelo algoritmo de deteção de cantos, a verde e a estimação do EKF, a vermelho.

Não tendo oportunidade de testar com precisão os resultados da estimação dos cantos pelo algoritmo de deteção de cantos com recurso ao *Motion Capture System*, considera-se a *ground truth* a estimação produzida pelo algoritmo. Ao observar-se o gráfico da Figura 10 verifica-se que a estimação do EKF tem um erro significativo face à estimação dada pelo algoritmo de deteção de cantos. Uma dificuldade que se enfrentou foi o facto de a frequência de leitura de dados do *laser* e dos dados de odometria do robot serem bastante diferentes. O veículo pode publicar os dados a uma frequência aproximadamente cinco vezes superior à frequência do *laser* sendo que de seguida mostra-se o método que se implementou de forma a reduzir o atraso entre medições: Sabendo que uma linha do ficheiro de odometria é constituída pelos dados das velocidades angulares de ambas as rodas, os seus ângulos e o instante de tempo em que a medição foi efetuada:

1) Ler a primeira linha do ficheiro de odometria e guardar o tempo em que essa medição foi feita numa variável auxiliar, *time_100ms*.
2) Ler nova linha e guardar o tempo correspondente na variável *get_time* e adicionar 100 milisegundos a *time_100ms*, escrevendo num novo ficheiro a linha anterior, enquanto se verifica *get_time* > *time_100ms*.
3) Escrever a nova linha no novo ficheiro, que corresponde ao instante *get_time* e atualizar a linha anterior atribuindo-lhe os valores da nova linha.
4) Repetir os passos (2) e (3) até que a leitura do ficheiro original seja completa.

O novo ficheiro corresponde ao ficheiro "corrigido"e será a partir desse que se irão usar os valores de odometria no algoritmo EKF. Este ficheiro tem um maior número de linhas que o original e garante-se que, à mesma frequência do *laser*, há uma leitura dos dados de odometria.

Como possível solução para o erro obtido no algoritmo EKF sugere-se que o *prediction step* e o *update step* sejam executados de forma assíncrona, isto a partir de um ficheiro de odometria com uma amostragem maior. Isto seria possível se não se adiciona-se o atraso na escrita do ficheiro no *Matlab*.

5 Conclusões

Este projeto permitiu-nos trabalhar diretamente com um robot, aprender a controlá-lo remotamente e perceber melhor o modo de como, combinando dados de natureza diferente - odometria e LRF -, se pode obter a sua localização num determinado ambiente.

No entanto, fomo-nos deparando com várias dificuldades ao longo da realização do trabalho, algumas delas já explicadas anteriormente, pelo que os resultados do Kalman Filter apresentam algum desvio relativamente à *pose* real do robot.

É importante também referir que este trabalho apenas abrange o caso em que o robot se encontra num espaço "aberto", ou seja, em que o LRF consegue alcançar, condicionado à sua posição, todos os pontos possíveis desse mesmo espaço e do robot. Não abrange, portanto, situações como por exemplo um espaço com corredores, com mudanças de direção, em que o LRF deixa de "ver"o robot.

Referências

[1] A. Vale, I. Ribeiro and J. Ferreira, *Localization of cask and plug remote handling system in ITER using multiple video cameras*, Fusion Engineering and Design, Elsevier, 2013.
[2] A. Vale, J. Ferreira and R. Ventura, *Vehicle localization system using offboard range sensor network*, IFAC, 2013.
[3] R. Ventura, *Derivation of the discrete-time Kalman filter*, IST, 2017.
[4] J. Xavier, M. Pacheco, D. Castro, A. Ruano and U. Nunes, *Fast Line, Arc/Circle and Leg Detection from Laser Scan Data in a Player Driver*.
[5] G. A. Borges and M. Aldon, *Line extraction in 2D range images for mobile robotics*, Journal of Intelligentand Robotic Systems 40, pp. 267-297, 2004.