UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

RAUL CATALUÑA ATHAYDE - 00246762

LOCALIZAÇÃO DE ROBÔS MÓVEIS NO ROS

Porto Alegre
2021

RAUL CATALUÑA ATHAYDE - 00246762

LOCALIZAÇÃO DE ROBÔS MÓVEIS NO ROS

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

ORIENTADOR:
Prof. Dr. Walter Fetter Lages

Porto Alegre
2021

RAUL CATALUÑA ATHAYDE - 00246762

LOCALIZAÇÃO DE ROBÔS MÓVEIS NO ROS

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Walter Fetter Lages, UFRGS
Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Banca Examinadora:

Prof. Dr. Walter Fetter Lages, UFRGS
Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Prof. Dra. Mariana Luderitz Kolberg, UFRFS
Doutora pela Pontifícia Universidade Católica do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Renato Ventura Bayan Henriques, UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

_____
Mario Sobczyk
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, novembro de 2021.

DEDICATÓRIA

Dedico este trabalho aos meus pais e meus amigos, em especial pelo e apoio em todos os momentos difíceis.

AGRADECIMENTOS

À Universidade Federal do Rio Grande do Sul, UFRGS, pela oportunidade de realização de estudos.

Aos colegas de curso pelo seu auxílio nas tarefas desenvolvidas durante o curso e apoio na revisão deste trabalho.

Agradeço ao professor Walter pela orientação e paciência.

RESUMO

Este trabalho aborda a implementação e ajuste de parâmetros de um sistema de localização baseada em filtro de partículas para um robô móvel, utilizando a arquitetura da Pilha de Navegação proposta pelo ROS. O modelo do robô, assim como seus sensores e o ambiente no qual a localização será feita foram desenvolvidos em trabalhos anteriores realizados no Laboratório de Robótica e Sistemas Embarcados da UFRGS. Para este trabalho foi integrado ao robô móvel um nodo para estimar a pose do mesmo através da odometria visual, a qual foi comparada qualitativa e quantitativamente com a pose estimada por odometria tradicional. O nodo de localização global escolhido para este projeto foi integrado com os nodos já existe e seus parâmetros sintonizados de acordo com o modelo do robô utilizado. Por fim foram realizadas simulações do processo de localização, nas quais pode se verificar que o erro de localização estava limitado.

Palavras-chave: Robótica Móvel, Localização, ROS, Algoritmo de Monte Carlo.

ABSTRACT

This document addresses the implementation and parameters adjustment of a localization system based on a particle filter for mobile robots, utilizing the "Navigation Stack" architecture proposed by ROS. The robot model, as well as its sensors and the environment in which localization shall take place was obtained during the execution of previous papers produced in the Robotics and Embedded Systems Laboratory in UFRGS. For this project a node was integrated to the software of the mobile robot to estimate the pose of the robot through visual odometry, which was qualitative and quantitative compared to the traditional odometry results. The chosen node for global localization was integrated with the existing project nodes and its parameters were tuned in accordance with the robot model utilized. Finally simulation of the localization process was performed, in which it was verifiable that the localization error was limited.

Keywords: Mobile Robotics, Localization, ROS, Monte Carlo Algorithm.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

ROS	Robot Operating System (sistema operacional de robôs)
SLAM	Simultaneous Localization and Mapping (mapeamento e localização simultâneos)
UFRGS	Universidade Federal do Rio Grande do Sul
AMCL	Adaptive Monte Carlo Localization (localização adaptativa de Monte Carlo)
RGB-D	Red Green Blue - Depth (vermelho verde azul - profundidade)

1 INTRODUÇÃO

1.1 Robótica Móvel

Robótica é a ciência da percepção e manipulação do ambiente físico através de mecanismos controlados por computadores. Exemplos de sistemas robóticos de sucesso incluem plataformas moveis para exploração planetária, braços robóticos em linhas de montagem, caros que navegam autonomamente em autoestradas, e braços robóticos para assistir cirurgiões. O que todos esses sistemas robóticos tem em comum é que eles são situados no mundo físico, percebem seu ambiente através de sensores, e manipulam esse ambiente com partes moveis (THRUN; BURGARD; FOX, 2005).

Dos exemplos citados anteriormente pode-se dividir a robótica em duas áreas, a robótica móvel e a robótica de manipuladores. A principal característica de um robô móvel é que a sua base pode se mover em relação ao seu ambiente, enquanto na robótica de manipuladores, a base do robô é fixa em relação ao seu ambiente e as partes do robô se movem em relação a esta base. Dentro da família de robôs moveis estes podem ser divididos de acordo com o número e tipo de rodas utilizadas. As rodas de um robô móvel, que determinam a classe do mesmo, podem ser fixas de eixo comum, orientáveis centradas, e universais, cujas combinações geram até cinco classes diferentes de robôs moveis, sendo a mais comumente utilizada a classe dos robôs com acionamento diferencial.

Além da variabilidade do tipo de robô móvel, também deve se fazer a diferenciação entre quais os tipos de ambientes em que um dado robô móvel é capaz de operar. Pode-se desejar que o robô seja capaz de se movimentar por terrenos acidentados, ou escorregadios, ou talvez seja desejável que ele consiga transitar sobre terrenos inclinados, como seria o caso para um carro autônomo, andando por uma cidade. Essa diferenciação é relevante pois ela influencia em quais os sensores que serão adotados pelo robô, e qual o tratamento que o sinal desses sensores deverá passar para ser considerado útil. Para o caso de um robô que deve lidar com terrenos muito variados, encoders nas rodas do robô podem não ser o mais adequado para realizar a localização, visto que estes terrenos podem aumentar o erro no cálculo da odometria a partir da leitura dos mesmos. No exemplo de um carro autônomo, ao precisar subir por uma rampa, ou passar por baixo de um viaduto, uma câmera pode não ser o melhor sensor a ser utilizado, uma vez que, essas situações implicam na mudança de luminosidade naturalmente disponível no ambiente, o que altera a qualidade da odometria visual, podendo torna-la excessivamente ruidosa. Com essas considerações, e o fato de, atualmente, o Twil, robô móvel utilizado para o desenvolvimento deste trabalho, o qual pode ser visto na Figura 1, ter uma câmera de profundidade e um encoder em quadratura por roda, este trabalho se restringe a abordar a localização no plano, assumindo variabilidade mínima de terreno (sem variações de altitude, inclinação e atrito).

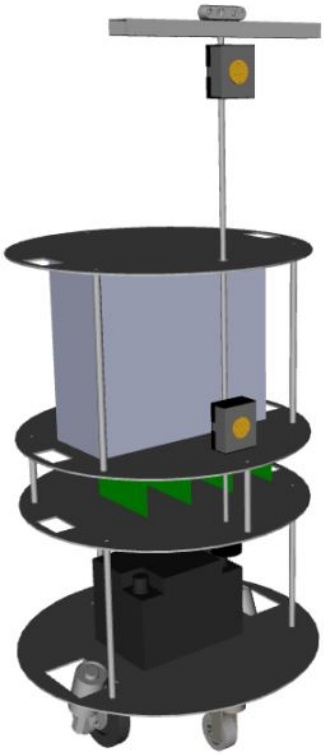


Figura 1: *Renderização em 3D do robô Twil.*

1.2 Motivação

Uma vez que foi estabelecido o escopo de mobilidade a ser utilizado, deve se abordar qual o grau de interferência humana que se deseja ter durante a operação do robô para poder executar com sucesso um dado objetivo. Uma opção seria que o robô fosse controlado como um carro, precisando sempre de comandos de um humano para determinar a trajetória a ser tomada, também cabendo a este humano determinar quando um objetivo foi atingido, o que implicaria em um laço único de controle, controlando a velocidade do robô. Este tipo de controle poderia ser obtido utilizando um *joystick* para comandar o robô móvel, e a trajetória do mesmo seria gerada pela interferência humana. Outra opção é que a trajetória a ser tomada seja determinada computacionalmente pelo próprio robô, de modo que fosse implementado no robô um controle em cascata, controlando tanto a velocidade do robô, quanto a sua pose. Para o robô Twil foi implementada a Pilha de Navegação proposta pelo ROS, para a qual deve se fornecer um mapa do ambiente em que o robô está, e um ponto para o qual o robô deve se mover, e a trajetória que o robô deve seguir para atingir este ponto é gerada automaticamente (PETRY, 2019).

No entanto, o algoritmo de navegação utilizado assumia que a pose do robô em seu ambiente poderia ser aproximada pela integral dos deslocamentos detectados pelo encoders das rodas do robô, técnica comumente conhecida como odometria. O problema que este trabalho tem como enfoque resolver é que a primeira hipótese sobre a pose do robô pode ser aproximada pela odometria das rodas deste robô só é válida para pequenas distâncias, uma vez que o erro de localização da odometria tende a crescer de forma ilimitada, com o deslocamento do robô, o que limita consideravelmente o comprimento das trajetórias que podem ser tomadas pelo robô de forma precisa. Isso acontece pois a odometria é uma leitura incremental que integra uma série de pequenas leituras de rotação das rodas. Em

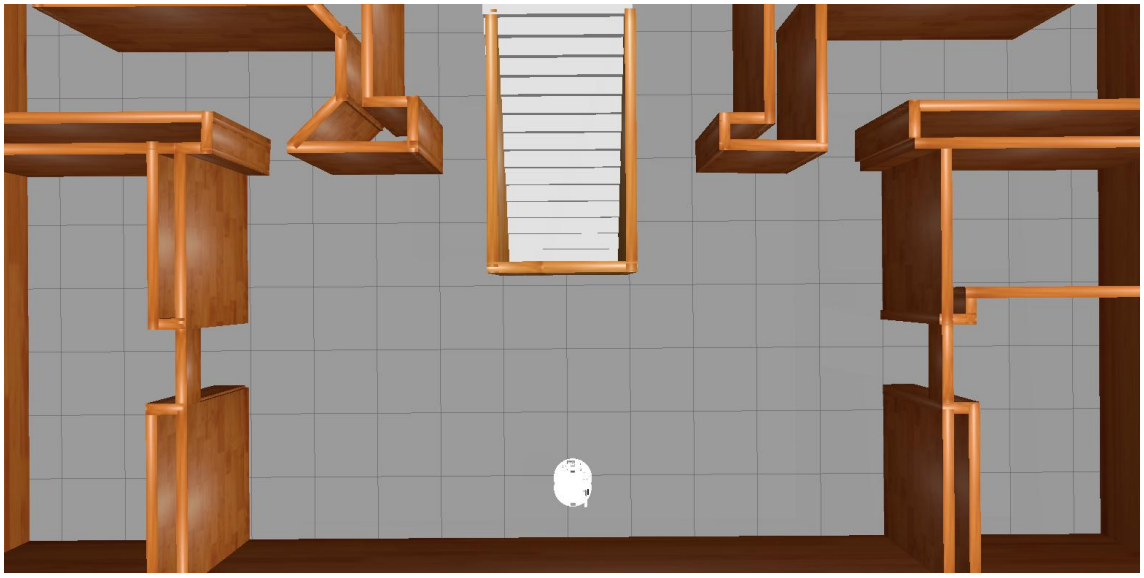


Figura 2: *Ambiente simulado pelo Gazebo.*

um mundo ideal esta aproximação não teria problemas, mas no mundo não ideal cada uma dessas pequenas leituras tem erro, cuja média é diferente de zero, associado a elas, e quando se integra um sinal com erro, o erro da medida final tende a divergir, tornando a medida final virtualmente inútil.

1.3 Objetivo

O ambiente de simulação utilizado neste trabalho pode ser visto na Figura 2, assim como na Figura 3 se tem a visualização de dados obtidos pelo sensoriamento do robô móvel antes da integração proposta por este trabalho. Este ambiente foi desenvolvido por Barros (2014) e Petry (2019) com o objetivo da implementação de controle de juntas no ROS, e a integração parcial da Pilha de Navegação do ROS no Twil, respectivamente. Logo o principal objetivo deste trabalho é permitir ao robô se localizar neste ambiente, de modo a permitir a navegação no ambiente simulado, uma vez que, antes da realização deste trabalho a navegação era possível em mapa, mas com ambiente livre (PETRY, 2019). Esta localização será feita através da integração total da Pilha de navegação, utilizando o nodo `/amcl`, o qual implementa os algoritmos probabilísticos descritos por Thrun, Burgard e Fox (2005). Deste modo futuros desenvolvimentos poderão utilizar rotas e trajetórias mais complexas com o robô, uma vez que, ao fim deste projeto, se espera que o robô possa navegar pelo ambiente simulado, do prédio centenário da UFRGS, sem erro crescente de pose.

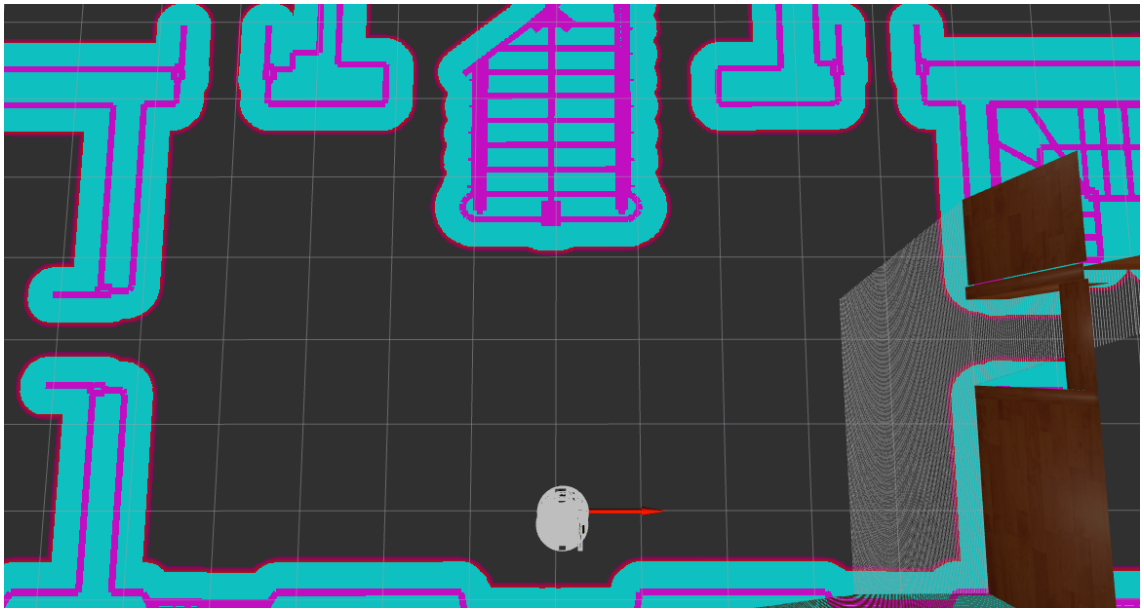


Figura 3: *Representação visual do sensoriamento presente no robô móvel, com o mapa, odometria e nuvem de pontos de uma câmera 3D.*

2 REVISÃO DA LITERATURA

2.1 Localização

A localização no âmbito da robótica móvel pode ser dividida em três diferentes problemas, cada um com maior complexidade que o anterior, podendo um ser considerado uma generalização do anterior.

2.1.1 Localização Local

A localização local é o caso mais simples de localização, uma vez que neste caso se parte do pressuposto que se conhece a pose inicial do robô, e que a pose atual do mesmo será determinada através de medidas incrementais de deslocamento do robô. Essa simplicidade vem com serias limitações, a mais significativa delas sendo a limitação da validade desta localização, uma vez que a soma de medidas incrementais acumula erro, se o mesmo não tiver média zero, e, por consequência, cresce de forma ilimitada.

2.1.2 Localização Global

A localização global retira a primeira grande restrição da localização local, de modo que não é mais obrigatório conhecer a pose inicial, e também passa a se permitir sinais com uma maior variância de ruído, apesar do mesmo ainda precisar ser limitado. As mudanças de restrições citadas se devem a mudança do tipo de sensor utilizado para solucionar este caso de localização, uma vez que para este caso precisa se utilizar sensores absolutos, ao invés dos sensores incrementais utilizados para solucionar casos de localização local.

2.1.3 Robô Sequestrado

Assim como a localização global é uma generalização da localização local, o caso do robô sequestrado é uma generalização da localização global. Nesta generalização a restrição removida é a de que o robô deve conhecer a variância do seu sensor utilizado para a localização, de modo a saber se sabe que está no local certo ou não. Apesar do nome, este caso é bastante comum, uma vez que robôs não são, normalmente, sequestrados, mas o próprio algoritmo de localização pode levar o robô a concluir, com alta confiança, que está em uma posição errada.

2.2 Odometria

Odometria significa medição do caminho, portanto consiste em calcular a pose a partir da medição do caminho percorrido (DUDEK; JENKIN, 2010). Outra característica importante da odometria é que a mesma é obtida pelo somatório de medidas incrementais

em sequência, de modo a aproximar uma medida absoluta.

A odometria depende diretamente do modelo cinemático do robô. Para o robô Twil, o qual possui uma configuração diferencial, pode se calcular a odometria de dois modos, ou assumindo que a sua trajetória é composta por sequencias de segmentos de retas, ou que é composta por segmentos de curva em sequência. A diferença dentre esses dois métodos é que ao assumir que a trajetória é composta por apenas segmentos de reta, se perde precisão, pois alguns segmentos de trajetória, entre intervalos amostrais, são realmente curvas, enquanto que ao assumir que todos os segmentos da trajetória são curvas, não se perde precisão quando algum segmento é realmente uma reta, pois a reta é uma curva com raio infinito (LAGES, 2018). Assim, conforme mostrado na Figura 4, a atualização da pose do robô é dada por:

$$\hat{x}_o(k) = \hat{x}_o(k-1) + \Delta\hat{x}_o(k) \tag{1}$$

$$\Delta\hat{x}_o(k) = \begin{bmatrix} \Delta s(k)\cos(\theta(k) + \frac{\Delta\theta(k)}{2}) \\ \Delta s(k)\,\mathrm{sen}(\theta(k) + \frac{\Delta\theta(k)}{2}) \\ \Delta\theta(k) \end{bmatrix} \tag{2}$$

$$\Delta s(k) = \Delta D(k)\frac{\mathrm{sen}\,\frac{\Delta\theta(k)}{2}}{\frac{\Delta\theta(k)}{2}} \tag{3}$$

$$\begin{bmatrix} \Delta D(k) \\ \Delta\theta(k) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(\Delta\phi_r(k)r_r + \Delta\phi_l(k)r_l) \\ \frac{1}{2b}(\Delta\phi_r(k)r_r - \Delta\phi_l(k)r_l) \end{bmatrix} \tag{4}$$

onde $\hat{x}_o(k) = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$, $\Delta\phi_r(k)$ e $\Delta\phi_l(k)$ são os deslocamentos angulares das rodas direita e esquerda, respectivamente, medidos pelos encoders, enquanto r_r e r_l são o raio das rodas direita e esquerda, respectivamente, e b é a distância entre as rodas do robô.

2.3 Odometria Visual

A odometria visual segue o mesmo princípio da odometria obtida pela contagem de pulsos de encoders, apresentada anteriormente. Isso acontece, pois, a pose atual do robô ainda é determinada pela integração de diversas medidas de deslocamento, e, por consequência, a odometria visual mantém a mesma restrição identificada para a odometria tradicional. A pose calculada pela odometria visual diverge da pose real do robô conforme o robô se desloca, até que o resultado de localização por odometria visual se torna inútil, já que o erro neste método também é ilimitado. Quanto tempo este resultado levará para tornar-se inutilizável irá depender de qual a variância de erro nas medidas incrementais, mas ainda é inevitável.

A odometria visual utiliza sensores ópticos, os quais podem variar entre câmeras 3D, câmeras estéreas (duas câmeras não 3D fixas uma em relação a outra), e com câmeras não 3D, utilizando processamento de imagem para calcular o fluxo óptico entre imagens. Com isso o robô tem o seu deslocamento entre dois instantes amostrais e pode realizar a integração desse sinal, conforme descrito para a odometria tradicional.

Assim como na odometria tradicional a qualidade do sinal depende do atrito entre as rodas e o solo, e da resolução dos encoders, na odometria visual a qualidade do sinal depende da resolução da câmera utilizada e das condições de cena em que o robô se

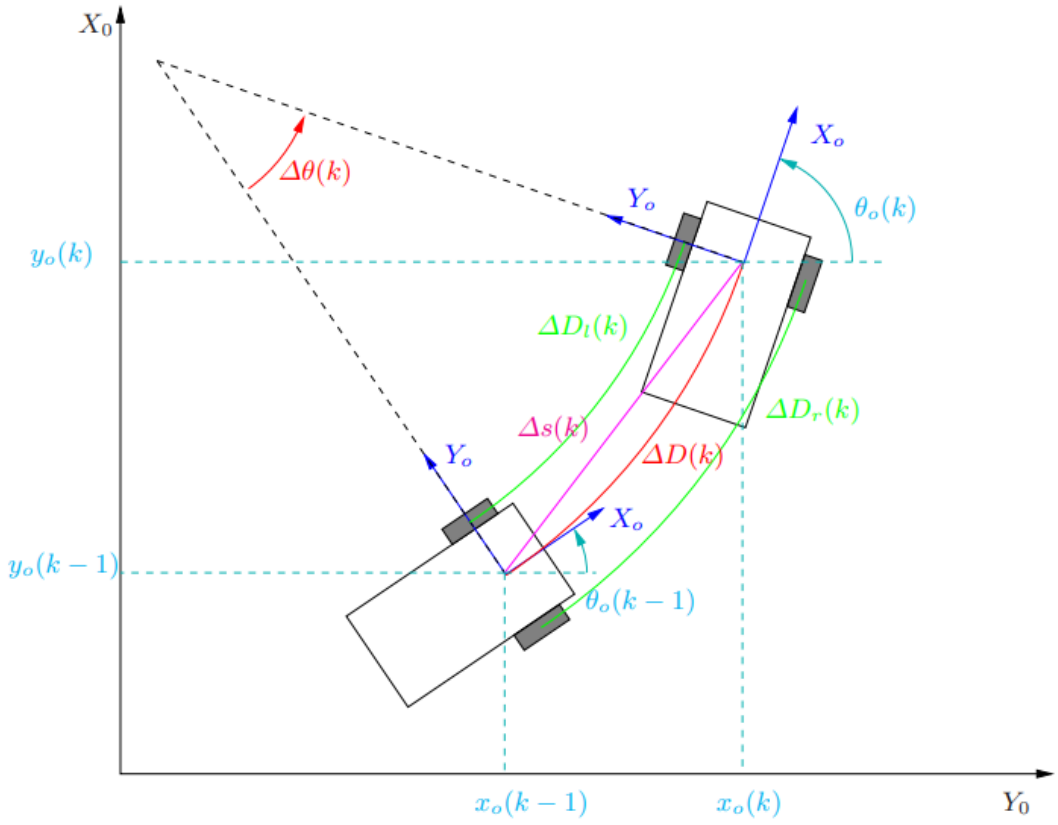


Figura 4: *Odometria supondo uma trajetória com arco de circunferência (*LAGES, *2018).*

encontra. No entanto a qualidade da cena pode variar muito mais drasticamente que o atrito entre superfícies, pois, caso o robô esteja utilizando iluminação natural, e entre instantes amostrais, essa iluminação seja alterada, é possível que as características pelas quais o fluxo óptico estava sendo calculado não sejam mais distinguíveis, o que pode levar o robô a perder a sua localização mais rapidamente que com a odometria tradicional.

2.4 Localização Probabilística

Robótica probabilista é a abordagem da robótica em que se utiliza o conhecimento de que tanto as percepções, quanto as ações de um robô tem um nível de incerteza associado as mesmas. Assim, ao invés de se determinar qual valor mais provável para a leitura de um sensor e utiliza-lo como verdade em todos os cálculos subsequentes, como é feito na robótica clássica, essas informações são representadas como distribuições de probabilidades, o que permite ao algoritmo utilizado saber a incerteza de uma dada informação. Para se aproximar do real estado do robô é preciso utilizar filtros probabilísticos para determinar qual é o estado com maior densidade de probabilidade, levando em consideração a configuração de estados anterior, o sinal de controle utilizado para se mover de um estado ao seguinte, e as medidas dos sensores obtidas nesta transição, assim como as respectivas densidades de probabilidades (THRUN; BURGARD; FOX, 2005).

2.4.1 Filtro de Bayes

O filtro de Bayes é um método utilizado para calcular a densidade de probabilidade de um estado, dado as medições atuais e o sinal de controle previamente aplicado. O método deste filtro calcula a crença de que o estado no instante $k+1$ do robô seja $X(k+1)$, dado que o estado no instante n é $X(k)$, e que o sinal de controle gerado para atingir $X(k+1)$ é $u(k)$, e as medições do robô no instante $k+1$ são $Z(k+1)$. Este processo é repetido de forma recursiva, uma vez que $X(k)$ depende de $X(k-1)$ (THRUN; BURGARD; FOX, 2005).

A crença da pose de um robô pode ser definida como a probabilidade de cada possível hipótese relativa ao real estado do robô. Distribuições de crença são probabilidades posteriores, ou condicional, sobre variáveis condicionadas pela informação disponível, a qual pode ser denotada por

$$bel(X(k)) = P(X(k)|Z(1:k), u(1:k)) \tag{5}$$

onde $X(k)$ é a pose do robô, $Z(1:k)$ são todas as medições dos sensores do robô até o instante k e $u(1:k)$ são todos os sinais de controle gerados até o instante k, de modo que a crença no estado $X(k)$ é a probabilidade condicional do estado $X(k)$ ocorrer dado que as medições e o sinais de controle gerados até o instante k são $Z(1:k)$ e $u(1:k)$, respectivamente, e o sinal de controle $u(k)$, utilizado aqui e no decorrer deste trabalho, é definido como $\left[\bar{x}(k-1)\bar{y}(k-1)\bar{\theta}(k-1)\bar{x}(k)\bar{y}(k)\bar{\theta}(k)\right]^T$. A probabilidade posterior é a distribuição de probabilidade do estado $X(k)$ no instante k condicionada pelas medições anteriores $Z(k)$ e controles anteriores $u(k)$.

Segundo Thrun, Burgard e Fox (2005) a crença de um dado estado é dada por

$$bel(X(k)) = \eta P(Z(k)|X(k)) \int P(X(k)|u(k), X(k-1)) bel(X(k-1)) dX \tag{6}$$

onde $P(Z(k)|X(k))$ é a probabilidade condicional das medições $Z(k)$ ocorrerem dado que o estado é $X(k)$, e $P(X(k)|u(k), X(k-1))$ é a probabilidade condicional do estado atual ser $X(k)$, dado que o controle gerado foi $u(k)$, e o estado anterior foi $X(k-1)$, η é o coeficiente utilizado para normalizar a distribuição, e representa uma densidade de probabilidade que reflete a estimativa do estado. Esta equação deve ser calculada para todos os possíveis estados de $X(k)$, para então obter a distribuição probabilística em um dado momento.

2.4.2 Método de Monte Carlo

O método de Monte Carlo pode ser utilizado para calcular a pose de um robô móvel pelo método Localização de Monte Carlo, o qual é um método onde a distribuição de probabilidade do estado do robô é representada por partículas. Este método é aplicável tanto a problemas de localização local quanto global, e em alguns casos problemas de robô sequestrado, e é um dos métodos mais utilizados para a localização, na robótica móvel. O método de Monte Carlo é composto por duas operações diferentes, sendo elas a amostragem de partículas em um modelo de movimento, e a aplicação de um modelo de medição para determinar a importância de cada uma dessas partículas. Por fim a partícula que tiver a maior probabilidade será a considerada como a estimativa do estado, e a distribuição de partículas obtidas na iteração atual é usada como ponto de partida para a iteração seguinte.

2.4.2.1 Modelo de Movimento

O modelo de movimentação utilizado pelo Algoritmo de Monte Carlo utiliza o deslocamento detectado pela odometria, no intervalo $[k-1,k]$, uma amostra da distribuição de probabilidade do estado $X(k-1)$ dada por $\begin{bmatrix} x(k-1) & y(k-1) & \theta(k-1) \end{bmatrix}^T$, e gera uma amostra distribuição de probabilidade do estado em $X(k)$.

A pose inicial medida pela odometria é $\begin{bmatrix} \bar{x}(k-1) & \bar{y}(k-1) & \bar{\theta}(k-1) \end{bmatrix}^T$ e a pose final medida pela odometria é $\begin{bmatrix} \bar{x}(k) & \bar{y}(k) & \bar{\theta}(k) \end{bmatrix}^T$.

O modelo supõe que o movimento pode ser descrito por uma rotação de $\bar{\theta}(k-1)$ até a direção de translação dada por

$$\delta_{rot1}(k) = \text{atan2}(\bar{y}(k) - \bar{y}(k-1), \bar{x}(n) - \bar{x}(k-1)) - \bar{\theta}(k-1) \tag{7}$$

seguida de uma translação de $\begin{bmatrix} \bar{x}(k-1) & \bar{y}(k-1) \end{bmatrix}^T$ para $\begin{bmatrix} \bar{x}(k) & \bar{y}(k) \end{bmatrix}^T$, dada por

$$\delta_{trans}(k) = \sqrt{(\bar{x}(k) - \bar{x}(k-1)^2 + (\bar{y}(k) - \bar{y}(k-1))^2} \tag{8}$$

e uma segunda rotação para a orientação final $\bar{\theta}(k)$ dada por

$$\delta_{rot2}(k) = \bar{\theta}(k) - \bar{\theta}(k-1) - \delta_{rot1} \tag{9}$$

Considerando-se as incertezas associadas às respectivas operações de rotação e translação, tem-se

$$\hat{\delta}_{rot1}(k) = \delta_{rot1}(k) + \omega_{rot1}(k) \tag{10}$$

$$\hat{\delta}_{trans}(k) = \delta_{trans}(k) + \omega_{trans}(k) \tag{11}$$

$$\hat{\delta}_{rot2}(k) = \delta_{rot2}(k) + \omega_{rot2}(k) \tag{12}$$

onde ω_{rot1}, ω_{trans} e ω_{rot2} são as incertezas associadas às respectivas operações de rotação e translação dadas por

$$\omega_{rot1}(k) \sim \mathcal{N}(0, (\alpha_1 \delta_{rot1} + \alpha_2 \delta_{trans})^2) \tag{13}$$

$$\omega_{trans}(k) \sim \mathcal{N}(0, (\alpha_3 \delta_{trans} + \alpha_4 (\delta_{rot1} + \delta_{rot2}))^2) \tag{14}$$

$$\omega_{rot2}(k) \sim \mathcal{N}(0, (\alpha_1 \delta_{rot2} + \alpha_2 \delta_{trans})^2) \tag{15}$$

Portanto, a amostra do estado no instante k é dada por

$$x(k) = x(k-1) + \hat{\delta}_{trans} \cos(\theta(n-1) + \hat{\delta}_{rot1}) \tag{16}$$

$$y(k) = y(k-1) + \hat{\delta}_{trans} \text{sen}(\theta(k-1) + \hat{\delta}_{rot1}) \tag{17}$$

$$\theta(k) = \theta(k-1) + \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \tag{18}$$

2.4.2.2 *Modelo dos Sensores*

O modelo de medição utilizado pelo Algoritmo de Monte Carlo, para descrever $P(Z(k)|X(k))$, considera o sinal de um sensor a laser pode ser modelado pela composição de quatro modelos de ruído diferentes.

Para modelar a leitura dos sensores quando detecta objetos descritos pelo mapa se utiliza

$$P_{hit}(Z(k)|X(k),m) = \begin{cases} \eta \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{(Z(k)-\bar{Z}(k))^2}{2\sigma_{hit}^2}}, & \text{if } 0 \leq Z(k) \leq Z_{max} \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

onde m é o mapa do ambiente onde estes dados foram obtidos, $P_{hit}(Z(k)|X(k),m)$ é a probabilidade condicional de que o valor real do dado seja $Z(k)$, dado o estado $X(k)$ e o mapa m, $\bar{Z}(k)$ é o valor medido pelo sensor, Z_{max} o alcance máximo do sensor e σ_{hit}^2 é a variância do ruído esperado neste modelo.

Para modelar a leitura dos sensores quando detecta objetos que não são descritos pelo mapa se utiliza

$$P_{short}(Z(k)|X(k),m) = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short}\bar{Z}(k)}, & \text{if } 0 \leq Z(k) \leq \bar{Z}(k) \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

onde λ_{short} é um parâmetro intrínseco deste modelo. Este modelo é utilizado para modelar objetos ausentes no mapa utilizado, mas presente no ambiente real.

A leitura dos sensores quando não é detectado nenhum obstáculo dentro do alcance do sensor é modelada por

$$P_{max}(Z(k)|X(k),m) = \begin{cases} 1 & \text{if } Z(k) = Z_{max} \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

A leitura dos sensores também pode apresentar erros sem qualquer relação com o valor que seria detectado pelo sensor, o que pode ser modelado por

$$P_{rand}(Z(k)|X(k),m) = \begin{cases} \frac{1}{Z_{max}} & \text{if } 0 \leq Z(k) \leq Z_{max} \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

Por fim os quatro modelos são somados, cada um com seu respectivo peso, que representa a influência da incerteza que o mesmo representa no ruído nas medições do sensor. O modelo dos sensores $P(Z(k)|X(k),m)$ é então obtido por

$$\begin{aligned} P(Z(k)|X(k),m) = C1P_{hit}(Z(k)|X(k),m) + C2P_{short}(Z(k)|X(k),m) + \\ C3P_{max}(Z(k)|X(k),m) + C4P_{rand}(Z(k)|X(k),m) \end{aligned} \tag{23}$$

onde $C1$, $C2$, $C3$ e $C4$ são o peso de cada comportamento modelado anteriormente no sensor, os quais serão discutidos na seção , 3.3.1.2 e pode então ser utilizado para determinar a distribuição de probabilidade que uma dada medida tenha ocorrido com uma dada pose em um dado mapa.

3 METODOLOGIA

Neste capítulo serão apresentados os métodos utilizados para integrar os algoritmos de localização previamente discutidos ao robô Twil. Para isso primeiramente serão abordados cada método de forma individual, discutindo as peculiaridades da implementação utilizada, bem como a sintonização dos mesmos, e por fim será abordada a integração dos mesmos com a arquitetura da pilha de navegação já integrada com o Twil.

3.1 Pilha de Navegação

A pilha de navegação do ROS é um conjunto de pacotes que permitem ao robô móvel se locomover por ambientes com obstáculos. Para isso a navegação pode ser dividia em duas partes, planejamento global e local, como pode ser visto na Figura 5. O planejamento global é referente a trajetória ao longo prazo, utilizando todo o ambiente, que neste caso é um mapa junto com as leituras de sensores, à esquerda da Figura 5, enquanto o planejamento local é referente ao desvio de obstáculos presentes dentro de um dado círculo centrado no robô, e para isso se utiliza somente os sensores para determinar o ambiente. Além disso, se utiliza a odometria do robô, junto com as transformações entre os sistemas de coordenadas dos sensores do robô para determinar onde o robô está em relação aos ambientes, tanto local quanto global, utilizados. Por fim, soma-se a isso a referência de pose recebida através do tópico `move_base_simple/goal`, é possível gerar a trajetória que o robô deverá seguir, e com o tópico `cmd_vel` para publicar as referências que o controlador de velocidade do robô deverá seguir.

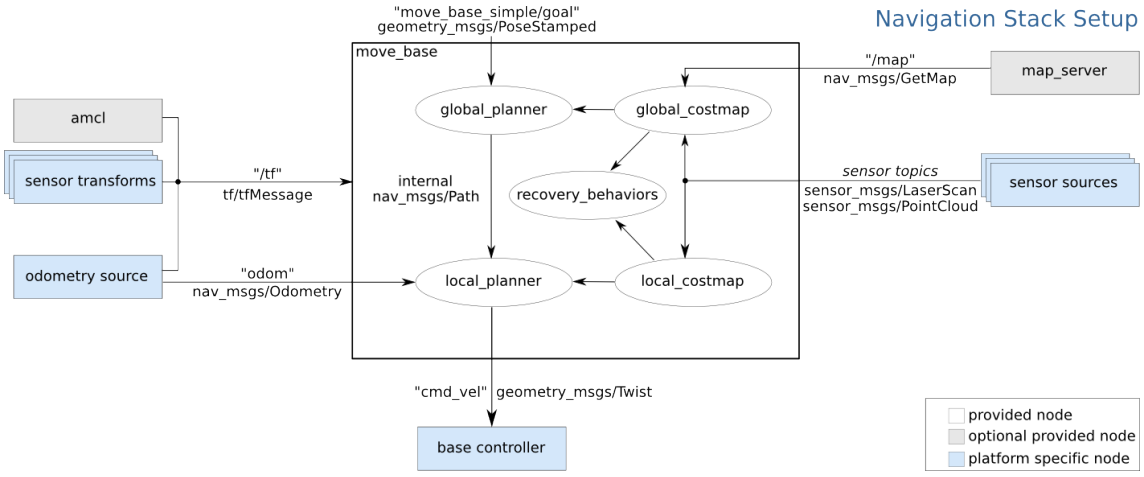


Figura 5: *Diagrama da pilha de navegação proposta pelo ROS.*

Fonte: Marder-Eppstein (2021)

O escopo deste trabalho é fornecer à pilha de navegação a correção da localização obtida por odometria para que o controle de posição feito pela pilha de navegação utilize uma pose com erro limitado. Essa correção é feita pelo nodo `/amcl`, que assina os tópicos `/map`, `/odom` e `/sensor_msgs/LaserScan` para estimar a pose real do robô, e publica no tópico `/tf` a correção da odometria em relação ao mapa utilizado. O `/amcl` na verdade publica tanto a transformação do sistema de coordenadas `/map` e `/odom`, quanto a pose do robô diretamente sobre o sistema de coordenadas `/map`, mas como a pilha de navegação utiliza diretamente a odometria, `/odom`, publicada em relação ao sistema de coordenadas `/odom`, se dará maior enfoque na publicação da transformação entre o sistema de coordenadas `/map` e `/odom`, o que será abordado com mais detalhe a seguir. No entanto, é importante ressaltar que os gráficos apresentados neste trabalho farão a comparação entre a pose real do robô, obtida diretamente da simulação, a pose do robô calculada por odometria, em relação ao sistema de coordenadas `/odom`, para identificar qual seria o erro caso a correção não fosse feita, e a pose do do robô, publicada pelo nodo de localização, em relação ao sistema de coordenadas `/map`.

Para permitir a integração de diferentes *drivers*, *softwares*, e bibliotecas desenvolvidas em momentos diferentes por pessoas diferentes, que, em muitos casos, nunca se conheceram, foi convencionado na REP (*ROS Enhancement Proposal*) 105 (MEEUSSEN, 2010) os sistemas de coordenadas utilizados para descrever a movimentação da base de um robô móvel entre o ambiente em que ele está. Essa convenção estabeleceu sistemas de coordenadas específicos: `/earth`, `/map`, `/odom`, e `/base_link`, cujas relações podem ser vistas na Figura 6.

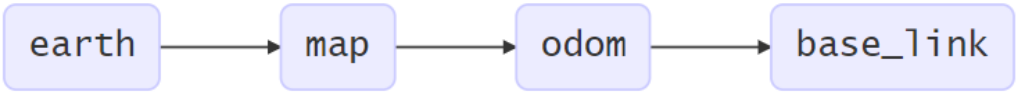


Figura 6: *Relação entre os sistemas de coordenadas do ROS.*

O sistema de coordenadas `/base_link` é aquele cuja distância e orientação com relação a base do robô móvel não se altera. Isso significa que a transformada desse sistema de coordenadas para um ponto fixo da estrutura do robô móvel será sempre a mesma, para um dado robô.

O sistema de coordenadas `/odom` é o sistema de coordenadas sob o qual as medidas de odometria de um dado robô móvel são publicadas. Em um mundo ideal com um único robô, este seria o único sistema de coordenadas utilizado, porém, como foi visto na seção 2.2, no mundo não ideal a odometria não é uma medida de localização única e suficiente. Foi, então, proposta a criação de um sistema de coordenadas `/map` no qual a odometria estaria referenciada de acordo com a localização global. Assim, qualquer algoritmo de localização em ROS deve publicar a transformação do sistema de coordenadas `/map` para o sistema de coordenadas `/odom`, de modo a corrigir o erro de localização obtido pela odometria.

Uma característica importante desta diferenciação de sistemas de coordenadas é que, assim como o planejamento da trajetória é dividido em global e local, esses sistemas de coordenadas também podem ser divididos desta forma, sendo `/odom` o sistema de coordenadas local e `/map` o sistema de coordenadas global. A pose de um robô, quando

referenciada no sistema de coordenadas `/odom`, é sempre continua, o que a torna útil como uma referência precisa de curto prazo, enquanto que a pose de um robô, quando referenciada no sistema de coordenadas `/map`, pode variar de formar não contínua, o que a torna uma escolha ruim para o âmbito local, mas muito boa para o global, uma vez que essas descontinuidades limitam o erro de localização a longo prazo.

Por fim o ultimo sistema de coordenadas é o `/earth`, o qual tem como objetivo permitir a utilização de múltiplos robôs, cada um com o seu respectivo mapa, mas como este trabalho utilizará apensa um robô, o sistema de coordenas `/earth` não será utilizado.

O nodo de localização sugerido, pela documentação da pilha de navegação, para a integração com a mesma, é o `/amcl`, ou `Adaptive Monte Carlo Localization`, o qual será abordado em mais detalhe em 3.3. No entanto, antes de usá-lo cegamente é importante avaliar as ferramentas de SLAM (*Simultanius Location and Mapping*) existem no ROS, uma vez que essas abordagens, como o nome sugere, devem resolver o problema de localização simultaneamente com o problema de mapeamento, o qual foi mantido fora do escopo inicial deste trabalho. Segundo Labbé e Michaud (2018), dentre as abordagens de SLAM mais comumente utilizadas em ROS, o `Real-Time Appearance-Based Mapping` é a implementação que integra a maior quantidade de sensores, o que deveria possibilitar a maior precisão em sua localização, mas a validade desse argumento será abordada em 3.2.

3.2 Real-Time Appearance-Based Mapping

O pacote `rtabmap_ros` fornece diversos nodos para localização de robôs moveis e mapeamento de ambientes, sendo `rtabmap_ros/rtabmap` e `rtabmap_ros/rgbd_odometry` os dois nodos avaliados neste trabalho. O nodo `rtabmap_ros/rtabmap` é, primariamente, um nodo de mapeamento, podendo ser utilizado em conjunto com o nodo `rtabmap_ros/rgbd_odometry` para implementar *SLAM* em um robô móvel (LABBE, 2021b). Como neste trabalho está se partindo do pressuposto que já existe um mapa do ambiente em que o robô deve se localizar, a localização pode ser feita utilizando apenas o nodo `rtabmap_ros/rgbd_odometry`.

Como o nome sugere, este nodo publica a pose do robô, com uma mensagem de odometria, e para isso espera receber imagens do tipo RGB-D. Para integrar este nodo à pilha de navegação foi preciso remapear os tópicos em que o nodo espera receber as imagens, tanto a de profundidade quanto a colorida, para os tópicos em que estas imagens estão sendo publicados pela simulação, além de informar o nome dos sistemas de coordenadas em que esta odometria deveria ser publicada, e o sistema de coordenadas fixo a base do robô. Para isso os tópicos `/rgb/image`, `/rgb/camera_info`, e `/depth/image` foram remapeados para os tópicos `/camera/color/image_raw`, `/camera/depth/camera_info`, `/camera/depth/image_raw`, respectivamente. Por fim os sistemas de coordenadas utilizados para a odometria e para a base do robô foram `/odom` e `/twil_origin`, respectivamente.

Além destes parâmetros para permitir ao nodo se comunicar com o resto da simulação, utilizou-se o parâmetro `/Odom/Holonomic` para definir o tipo de de rodas utilizadas pelo robô, e o parâmetro `/Odom/FilteringStrategy` para definir o tipo de filtro que seria utilizado no cálculo desta odometria. Para o parâmetro `Odom/Holonomic` foi atribuído o valor de *false*, uma vez que o robô Twil tem acionamento diferencial, e caso fosse atribuído *true*, seria considerado que o robô tem rodas universais. Já com o parâmetro `/Odom/FilteringStrategy` pode se escolher três tipos opções diferentes para a filtragem dos dados, sendo elas: sem nenhum filtro, filtro de Kalman e filtro de partículas. Ambas as

opções de filtro têm como finalidade suavizar a odometria visual gerada pelo nodo, e não corrigir o desvio decorrente de integrar medições incrementais para obter uma localização absoluta (LABBE, 2021a).

Optou-se por utilizar a pose estimada pelo nodo `/rtabmap_ros/rgbd_odometry` como uma segunda fonte de odometria, caso a odometria obtida pelas rodas se mostrasse inadequada, uma vez que uma fonte de odometria com alta precisão não impede que o erro de localização da odometria seja ilimitado, mas apenas aumenta o intervalo de tempo em que estas informações são úteis.

3.3 Adaptive Monte Carlo Localization

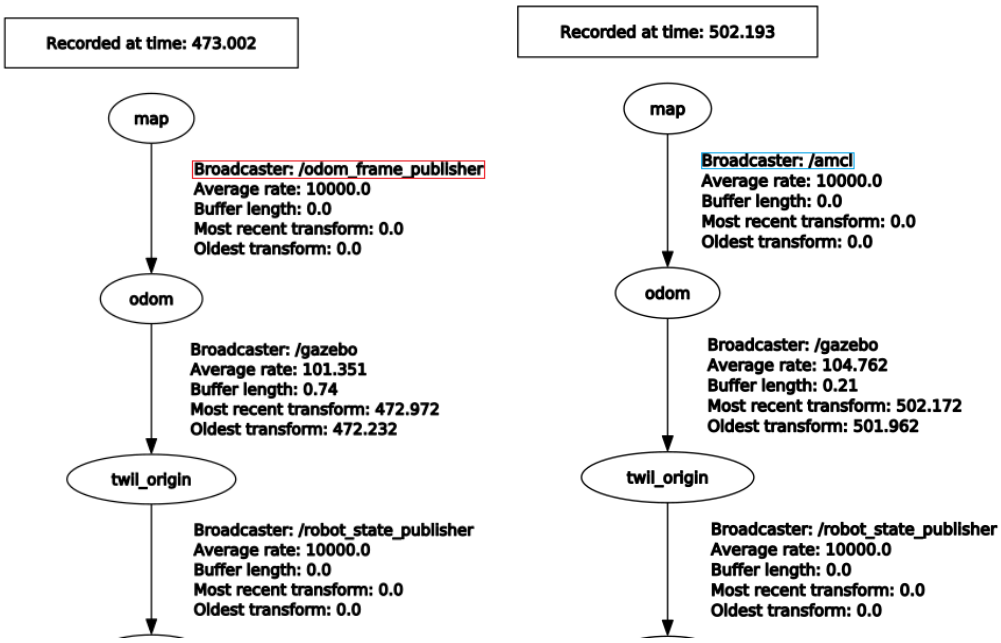
Com o pacote de localização a ser utilizado escolhido, foi preciso responder dois questionamentos importantes, sendo eles quais os tópicos que este nodo utiliza, tanto para receber dados quanto para publica-los, e quais parâmetros de configuração estão disponíveis para sintonizar o algoritmo de localização.

Responder primeiro o questionamento relativo aos tópicos utilizados é o mais adequado, uma vez que para visualizar o efeito dos parâmetros de configuração é importante que o nodo já esteja integrado com o robô. De acordo com Gerkey (2021), o nodo `/amcl` publica a pose estimada, junto com a sua covariância, em relação ao mapa utilizado, a nuvem de partículas de onde esta pose foi obtida, e a transformação do sistema de coordenadas `/map` para o `/odom`, de forma a corrigir a odometria de acordo com a pose estimada, conforme visto na seção 3.1. Para calcular essas informações o nodo subscreve um mapa do ambiente utilizado, as transformações entre sistemas de coordenadas existentes, a pose inicial do robô, a qual também pode ser definida por parâmetros de configuração, e uma leitura de um escâner laser.

Para integrar o nodo de localização à arquitetura de *software* já presente no robô é preciso remapear os tópicos em que estas informações são esperadas pelo `/amcl`, para os tópicos em que estas informações são publicadas na arquitetura de *software* atual no robô. Dentre os quatro tópicos listados anteriormente apenas dois deles precisaram ser remapeados, uma vez que as transformações entre sistemas de coordenadas estão nos tópicos padrão, e a pose inicial será fornecida por parâmetros de configuração. O tópico relativo ao mapa utilizado foi remapeado para o tópico `/map`, já que este é o tópico publicado pelo `/map_server` já presente na arquitetura de *software* do robô. Um ponto importante a ser ressaltado é que neste momento o tópico do mapa não apresenta grandes desafios visto que está sendo utilizando um mapa estático, mas caso se deseje ampliar integração do robô para poder navegar por ambientes não mapeados, executando localização e mapeamento simultâneos, será preciso alterar este ponto de integração.

Por fim o `/amcl` espera uma mensagem do tipo `/sensor_msgs/LaserScan`, porem o robô atualmente não é integrado com um sensor que publica este tipo de mensagem, mas sim com uma câmera 3D que publica mensagens do tipo `/sensor_msgs/Image`. Para converter a imagem de profundidade, publicada pela câmera 3D, para uma mensagem do tipo `/sensor_msgs/LaserScan`, utilizou-se o nodo `/depthimage_to_laserscan`, o qual faz exatamente o que o nome sugere, converte uma imagem com um canal de profundidade em uma mensagem de tipo `/sensor_msgs/LaserScan`.

Os parâmetros de configuração dos tópicos `/depthimage_to_laserscan` são relativamente simples, precisando apenas indicar o alcance do laser e o sistema de coordenadas no qual o mesmo deverá ser referenciado. Uma vez que o intuito com este tópico é fornecer ao `/amcl` a melhor leitura a laser possível, dentro dos limites que o robô real é capaz de for-



(a) Antes da integração. (b) Depois da integração.

Figura 7: *Transformações entre sistemas de coordenadas antes e depois da integração com o nodo de localização.*

necer, e não simular um leitor a laser específico, utilizou-se os limites de alcance mínimo e máximo da câmera *Intel RealSense* informado em Intel RealSense (2019), sendo eles $0,3m$ e $10m$, respectivamente. Além disso, para o sistema de coordenadas, se utilizou o sistema de coordenadas `camera_link`, o qual foi definido no arquivo `twil.urfd.xacro`. Por fim, foi possível alimentar o `/amcl` com a leitura de escâner laser do ambiente simulado, sem precisar remapear nenhum tópico, visto que ambos os nodos utilizam o mesmo tópico padrão.

Com isso o nodo de localização foi integrado à pilha de navegação previamente implementada no robô móvel conforme pode ser visto na Figura 8, uma vez que, dentre os tópicos publicados, os únicos utilizados pelo resto da pilha de navegação são as transformações entre os sistemas de coordenadas do robô, mas como este estão mapeados para o tópico padrão, não foi preciso remapear nenhum tópico para finalizar esta integração. Pode-se identificar na Figura 7(a) a relação entre os sistema de coordenadas antes da integração com o nodo de localização, e, na Figura 7(b), a relação entre os sistema de coordenadas após a integração do `/amcl`, sendo a diferença entre ambas que antes a transformação entre `/map` e `/odom`, era publicada por uma transformação estática, de modo que o desvio da odometria não era corrigido, e após a integração essa mesma transformação é publicada pelo nodo `/amcl`.

3.3.1 Parâmetros de Configuração

O nodo `/amcl` implementado no ROS possui múltiplos parâmetros diferentes para sintonizar o desempenho do filtro de Monte Carlo, os quais podem ser divididos em três diferentes tipos de parâmetros. Parâmetros genéricos de filtro especificam quantas partículas serão utilizadas, bem como as tolerâncias, e intervalos de atualização utilizados. Parâmetros do modelo de laser descrevem a amplitude e variância do modelo de ruído utilizado para o sensor a laser. Parâmetros do modelo de odometria descrevem o a variância

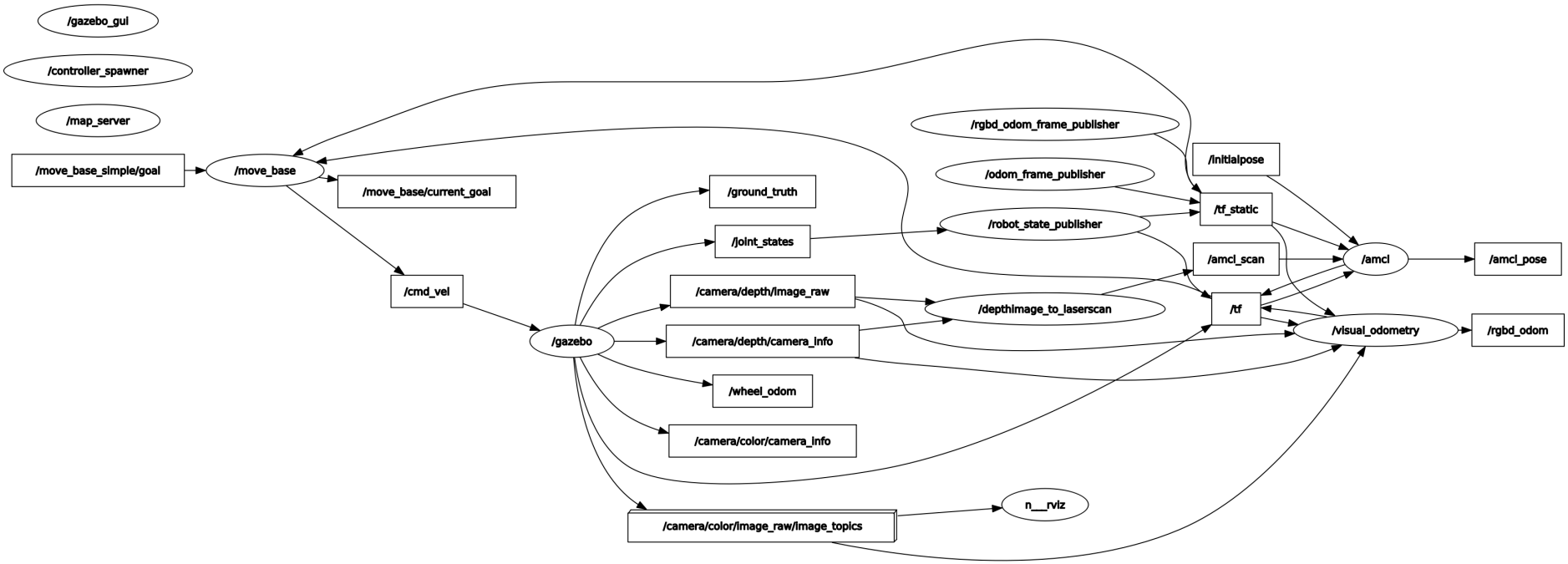


Figura 8: *Grafo computacional dos nodos existentes no Twil, após a integração com o nodo de localização.*

do ruído da odometria, bem como o tipo de configuração de rodas do robô móvel.

3.3.1.1 Parâmetros genéricos do filtro de Partículas

Os parâmetros genéricos do filtro são parâmetros relativos ao funcionamento do filtro de partículas utilizado pelo Algoritmo de Monte Carlo, especificamente neste caso foram calibrados os parâmetros relativos à quantidade de partículas utilizadas pelo filtro e quando a atualização do filtro deve ocorrer.

O primeiro desses parâmetros a ser determinado foi o número máximo e mínimo de partículas. Isso foi feito seguindo o resultado obtido nos experimentos de Fox et al. (1999), onde o melhor desempenho foi observado utilizando entre 1000 e 5000 partículas, de modo que estes foram os valores utilizados para os parâmetros `/min_particles` e `/max_particles`, respectivamente.

Os próximos parâmetros a serem determinados foram os parâmetros de atualização do filtro, os quais foram determinados empiricamente. Nesta implementação especifica do filtro de Monte Carlo, a atualização do mesmo é feita pela variação de pose percebida na leitura da odometria recebida, ou seja, para que o filtro seja atualizado a variação de pose, a translação ou a rotação devem ultrapassar os valores definidos nestes parâmetros.

Os primeiros ensaios para determinar o valor destes parâmetros foram feitos utilizando o valor padrão dos mesmos, $0,2$m e $30°$. Neste experimento o robô foi comandado para se mover da origem do mapa até o corredor localizado no canto superior direito do mapa, conforme pode ser visto na Figura 9. Nesta figura é possível perceber que os valores padrão para os parâmetros de atualização do filtro de partículas não são adequadas para esta implementação, uma vez que ao comparar a trajetória real com a trajetória estimada pode se perceber que a rotação do robô não está sendo identificada pelo estimador, de modo que deve se aumentar a resolução relativa à rotação. Com isso o próximo ensaio foi feito com `/update_min_d`, parâmetro relativo a miníma translação para atualizar o filtro, como $0,2$m e `/update_min_a`, parâmetro relativo a miníma rotação para atualizar o filtro, como $18°$. O resultado deste ensaio pode ser visto na Figura 10, na qual é possível perceber que não houveram desvios significativos entre a trajetória real e a trajetória estimada pelo filtro de Monte Carlo.

3.3.1.2 Parâmetros do modelo de laser

Os parâmetros do modelo de laser, como o nome sugere, especificam como as medições obtidas pelo escâner laser, são interpretadas pelo Algoritmo de Monte Carlo. Para este caso foram calibrados os parâmetros relativos ao alcance do laser, bem como os parâmetros relativos ao ruído esperado nos dados do laser, levando em consideração que os dados da mensagem do tipo `/sensor_msgs/LaserScan` são obtidas pela conversão da mensagem de tipo `/sensor_msgs/Image` para `/sensor_msgs/LaserScan`.

Primeiramente se determinou `/laser_min_range` e `/laser_max_range`, os quais são parâmetros para especificar o alcance máximo e mínimo do escâner laser, respectivamente. Estes foram determinados com o intuito de manter a coerência entre os nodos utilizados, nominalmente `/amcl` e `/depthimage_to_laserscan`, e aproximar o comporta mente simulado do comporta mente que se obteria ao utilizar o robô real. Com isso em mente o alcance do laser esperado pelo `/amcl` foi utilizado como o mesmo alcance do laser gerado pelo `/depthimage_to_laserscan`, de modo que `/laser_min_range` e `/laser_max_range` foram utilizados como $0,3$m e 10m respectivamente.

Os próximos parâmetros considerados para a calibração foram os parâmetros relativos ao ruído esperado nos dados de um escâner laser integrado com o `/amcl`, nominalmente

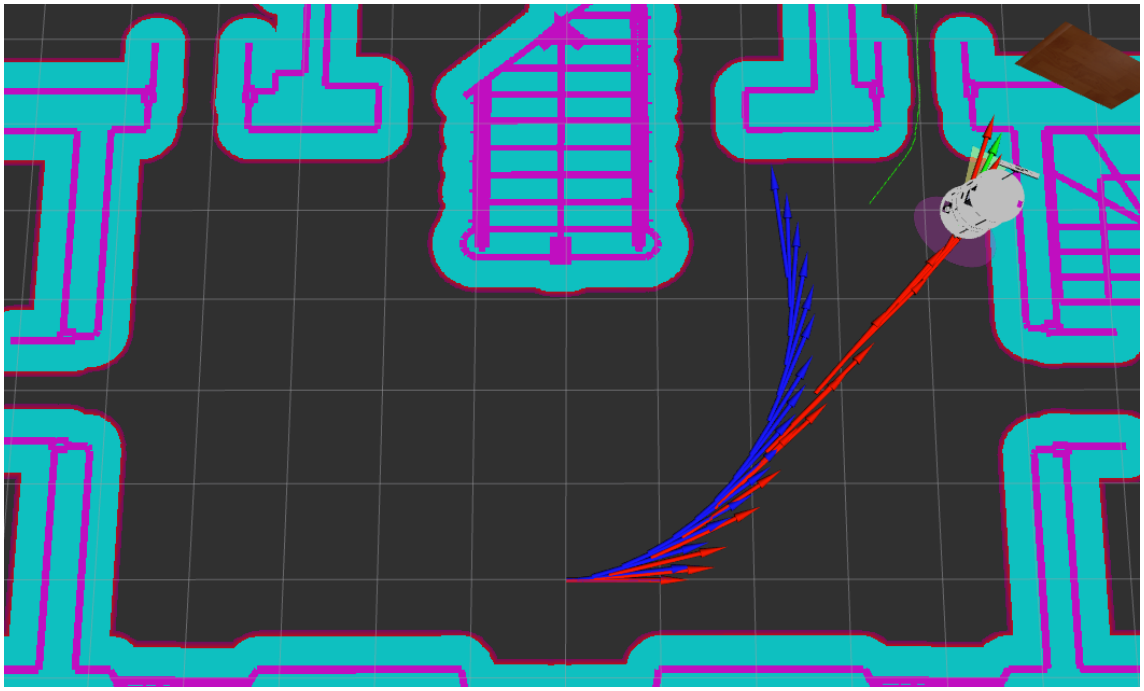


Figura 9: *Representação gráfica da trajetória real, em azul, e da trajetória estimada pelo filtro de Monte Carlo, em vermelho, com os parâmetros de atualização default do filtro de partículas.*

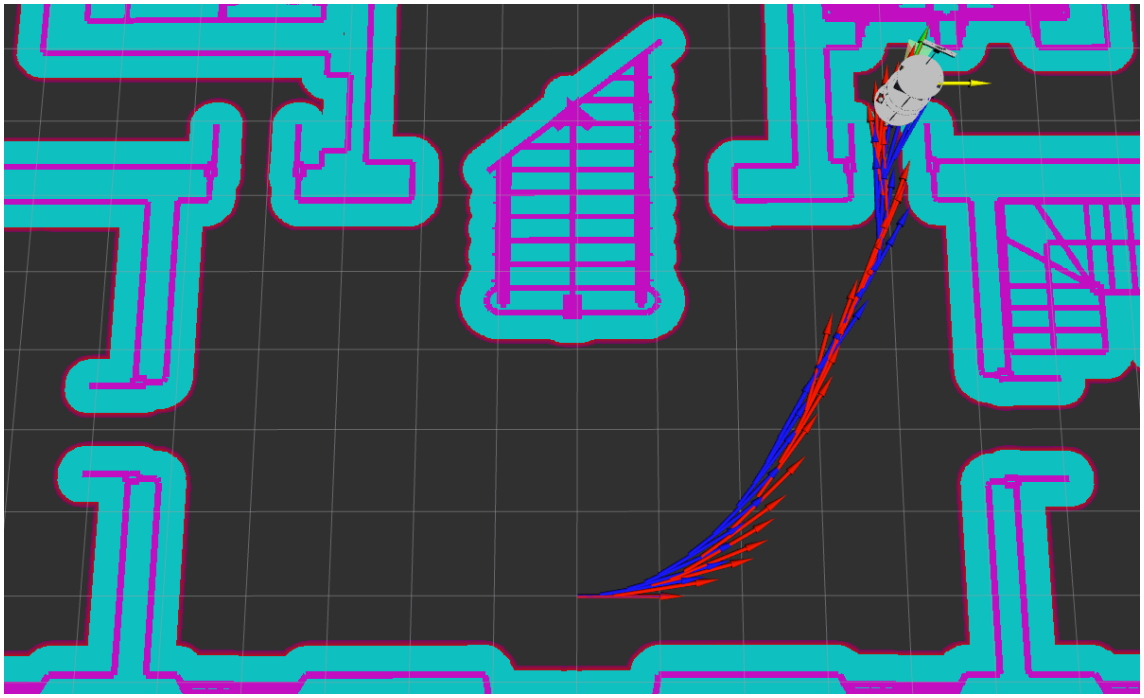


Figura 10: *Representação gráfica da trajetória real, em azul, e da trajetória estimada pelo filtro de Monte Carlo, em vermelho, com os parâmetros de atualização do filtro de partículas recalibrados.*

/laser_z_hit, /laser_z_short, /laser_z_max e /laser_z_rand. Cada desses parâmetros é o peso de um dado modelo de ruído diferente que pode estar presente em um leitor a laser tradicional, e são utilizados para combinar diferentes tipos de ruído para melhor aproximar o comportamento real do sensor. O parâmetro /laser_z_hit é o peso de uma distribuição gaussiana em torno do valor verdadeiro desta medida. O /laser_z_short é o peso de uma exponencial decrescente representado a probabilidade de um obstaculo não presente no mapa estar presente no alcance do escâner laser do robô. O /laser_z_max é o peso relaciona a ocorrência de uma falha no sensor, que pode causar com que a leitura do mesmo seja representada pelo valor máximo do sensor. Por fim, /laser_z_rand é o peso relacionado a probabilidade de ocorrência de valores aleatórios inexplicados na leitura do sensor.

A forma mais segura de realizar o ajuste destes parâmetros é através de ensaios no qual o robô é manualmente movimentado até uma dada posição, e nesta posição a leitura do sensor é comparada com a informação de um mapa. Então isso é repetido múltiplas vezes, com múltiplas posições diferentes, para, por fim realizar o ajuste pela minimização do erro entre a leitura do sensor e a informação do mapa, manipulando estes pesos. Porém, para o cenário simulado a maioria destes pesos não é aplicável. Durante a simulação todos os obstáculos existentes são as paredes do ambiente, as quais estão todas no mapa, o que torna /laser_z_short inaplicável. Além disso, na simulação não se tem realmente um escâner laser, nem uma câmera RGB-D cuja imagem é processada para gerar o sinal a laser. A imagem da câmera é, na verdade, obtida diretamente dos objetos presentes dentro do campo de visão especificado para esta dada câmera, o que torna /laser_z_max também inaplicável, visto que o mesmo representa um comportamento espúrio e não simulado.

Os únicos parâmetros aplicáveis são /laser_z_hit e /laser_z_rand, uma vez que os comportamentos representados pelos respectivos modelos são influenciados somente pela iluminação do ambiente, a qual foi a única influencia no sinal laser simulada para a simplificação do modelo utilizado. No entanto a condição de iluminação utilizada neste trabalho é a uma condição em que todos os objetos recebem iluminação ortogonal a eles, de modo a não ter sombras, exceto quanto um objeto sobrepõe o outro, o que torna a influência de ruído branco no sinal do sensor a laser, pequena e existente. Após algumas iterações os valores para o parâmetro /laser_z_hit e /laser_z_rand que geraram os melhores resultados de localização foram $0,99$ e $0,01$ respectivamente.

3.3.1.3 Parâmetros do modelo de odometria

Os parâmetros do modelo de odometria definem a variância do ruído da odometria, assim como os sistemas de coordenadas em que o robô e a odometria estão, além de qual transformação entre sistemas deve ser publicada.

Primeiro se determinou os parâmetros relativos ao sistema de coordenadas utilizados. Os parâmetros /odom_frame_id foi ajustado para /odom, uma vez que este é o sistema de coordenadas em que a odometria é publicada pelo controlador de velocidade utilizado. O parâmetro /base_frame_id foi ajustado para /twil_origin uma vez que este é o nome do sistema de coordenadas associado a base do robô e definido no arquivo twil.urfd.xacro. O parâmetro /global_frame_id foi utilizado como /map, uma vez que este é o sistema de coordenadas em que o mapa fornecido pelo nodo /map_server é publicado. Por fim, o último parâmetro, relativo a transformações entre sistemas de coordenadas é o /tf_broadcast, o qual foi definido como *true*, para que o nodo /amcl publicasse a transformação entre o /map e o /odom.

A seguir foram determinados os parâmetros relativos à variância do ruído que o fil-

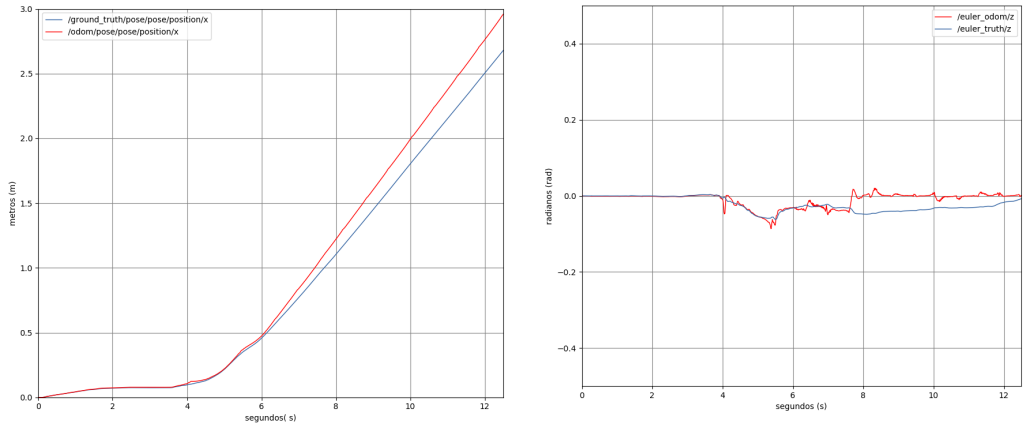
tro de Monte Carlo deveria esperar que estivesse no sinal de odometria. O primeiro destes parâmetros especifica qual o tipo de configuração de rodas que o robô utiliza. O `/odom_model_type` aceita quatro valore possíveis, mas apenas duas configurações de rodas são suportadas. Os valores suportados são `diff`, `omini`, `diff-corrected`, e `omini-corrected`. Os dois primeiros distinguem entre robôs com acionamento diferencial e robôs omnidirecionais, ou seja, que pode se mover em todas as direções independente da sua orientação. Já os dois últimos são equivalentes aos primeiros, mas foram incluídos no nodo alguns anos após a sua disseminação para corrigir um *bug* encontrado no cálculo deste modelo, e para não obrigar a recalibração de robôs que já haviam sido calibrados com este *bug* em consideração, os parâmetros originais foram mantidos como compatibilidade retroativa. Levando em consideração o que cada um destes possíveis valores significa, escolheu-se por utilizar o `/odom_model_type` como `diff-corrected`, uma vez que o robô Twil tem acionamento diferencial, e para uma nova implementação não se precisa utilizar a compatibilidade retroativa presente no algoritmo.

Os próximos, e últimos, parâmetros ajustados são `/odom_alpha1`, `/odom_alpha2`, `/odom_alpha3`, `/odom_alpha4`, os quais especificam a variância da incerteza das componentes de rotação e translação em relação ao valor real das componentes de rotação e translação da variação de pose do robô móvel, entre dois instantes amostrais. Nominalmente, `/odom_alpha1` especifica a variância de ruído na componente de rotação da odometria, causada pela componente de rotação da movimentação do robô. Já `/odom_alpha2` especifica a variância de ruído na componente de rotação da odometria, causada pela componente de translação da movimentação do robô. Enquanto `/odom_alpha3` especifica a variância de ruído na componente de translação da odometria, causada pela componente de translação da movimentação do robô. E, por fim, `/odom_alpha4` especifica a variância de ruído na componente de translação da odometria, causada pela componente de rotação da movimentação do robô.

Para ajustar estes parâmetros foram realizados dois ensaios, um em que o robô foi acionado para andar em linha reta indefinidamente, e outro em que o robô foi acionado para rotacionar em volta do seu próprio eixo indefinidamente. Esses ensaios foram feitos com o objetivo de isolar a influência de cada componente de movimentação do robô sobre cada componente de localização da odometria.

O primeiro ensaio foi feito com a translação do robô, e nele pode se observar que para, aproximadamente, 3 m de translação, conforme pode ser visto na Figura 11(a), ocorreu um erro de 0,4 m. Este erro pode ser normalizado para determinar o erro esperado por intervalo amostral de translação, ao ser dividido pelo deslocamento real, para obter o erro por unidade, e então ser multiplicado pelo intervalo amostral de translação, de modo a se obter $0,022$, que foi o valor atribuído ao parâmetro `/odom_alpha3`. Neste mesmo ensaio foi medido o efeito da translação sobre a componente de rotação da odometria na Figura 11(b), na qual pode ser observar um erro de medição de, aproximadamente, 5,73° durante os 3,4 m de translação. Esse erro, após ser normalizado e corrigido pelo intervalo amostral de translação, uma vez que representa o efeito da translação sobre a rotação, o parâmetro `/odom_alpha2` foi determinado como $0,005$.

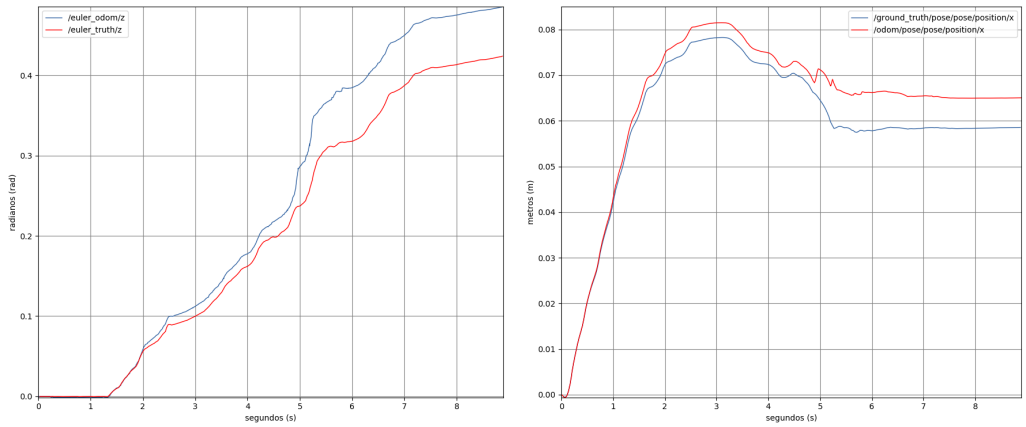
No segundo ensaio mediu-se o efeito da rotação real do robô sobre a rotação e a translação retornadas pela odometria. Conforme pode ser visto na Figura 12(a) pode se identificar um erro de localização no componente de orientação da odometria de 1,15°, os quais ocorreram durante uma rotação real de 10,89°. Este erro, ao ser normalizado, resulta em $0,105$, o qual ao ser corrigido pelo intervalo amostral de rotação de 5,73°, resulta que o parâmetro `/odom_alpha1` deve ser $0,0105$. Por fim, ainda no ensaio de rotação, pode



(a) Influência da translação real na translação medida.

(b) Influência da translação real na rotação medida.

Figura 11: *Translação e rotação real do robô comparadas com translação e rotação medida por odometria durante o ensaio de translação pura.*



(a) Influência da rotação real na rotação medida.

(b) Influência da rotação real na translação medida.

Figura 12: *Translação e rotação real do robô comparadas com translação e rotação medida por odometria durante o ensaio de rotação pura.*

se identificar na Figura 12(b) que o erro de localização no componente de translação da odometria, causado pela rotação é de 0,005 m, durante os 10,89° de rotação do ensaio. Este erro após ser normalizado e ajustado pelo intervalo amostral de rotação, resulta que o parâmetro `/odom_alpha4` é $0,0025$.

4 RESULTADOS

Neste capítulo serão apresentados os resultados dos experimentos descritos no capítulo anterior, assim como, como o algoritmo utilizado se comporta quando comparado com diferentes técnicas de localização. As técnicas de localização escolhidas para a comparação foram a odometria tradicional, obtida pela leitura dos pulsos de encoders fixados nas rodas, o algoritmo de Monte Carlo, o qual foi discutido com profundidade no capítulo anterior, e por fim a odometria visual, a qual se utiliza das leituras da câmera RGB-D para calcular o fluxo óptico do ambiente, e estimar a variação de posição do robô neste ambiente.

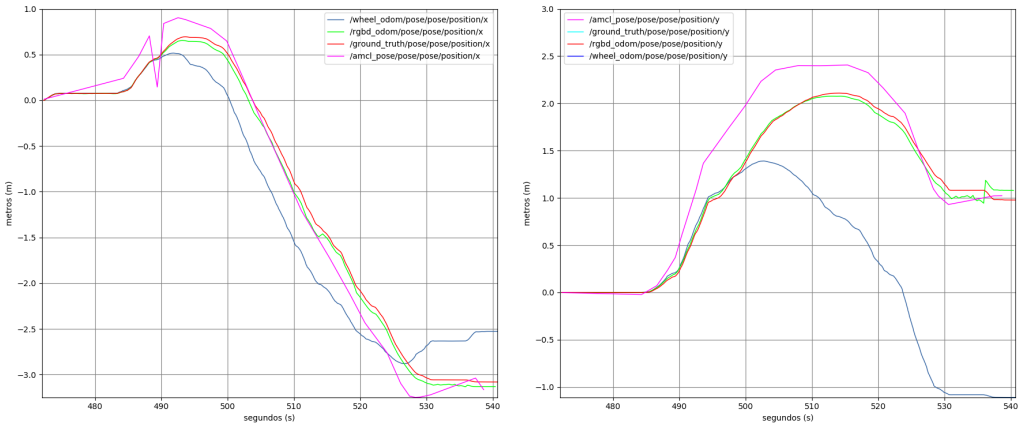
4.1 Localização por Odometria tradicional

A odometria tradicional não foi implementada durante a execução deste trabalho, mas foi o sensoriamento inicialmente utilizado no robô. Como este método de localização foi dito como inadequado, essa alegação deve ser ilustrada e este desempenho utilizado como comparativo de desempenho do método de localização utilizado na execução deste trabalho.

Conforme o esperado, a localização por odometria tradicional apresentou erro de posição não limitado, e crescente com o deslocamento do robô, conforme podo ser visto na Figuras 11(a), 11(b), 12(a) e 12(b). Outro resultado esperado foi que a odometria por encoders se mostrou mais precisa que o algoritmo de Monte Carlo enquanto o deslocamento do robô se manteve pequeno, uma vez que o erro de localização por odometria é crescente com o deslocamento, e com deslocamentos pequenos, se obteve erros pequenos, o que pode ser visto nas Figuras 16(a), 16(b), 17(a) e 17(b). Este problema de precisão dos métodos de localização está altamente co-relacionado com o atrito entre as rodas do robô e a superfície na qual o robô se movimenta, e não com a resolução do sensor. Isso pois o simulador utilizado (Gazebo), simula o atrito entre superfícies, de modo que nem toda a rotação detectada pelos encoders é devida ao deslocamento do robô, e vice-versa.

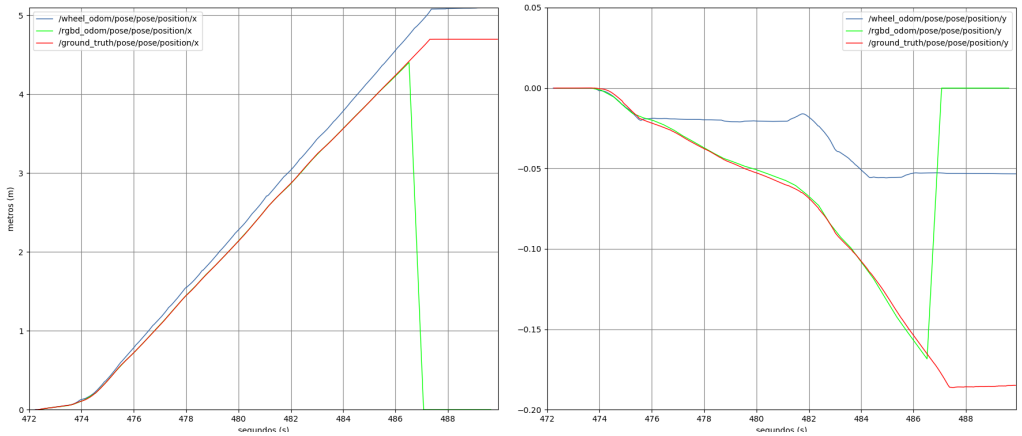
4.2 Localização por Odometria visual

A odometria visual apresentou uma precisão ainda melhor que a odometria tradicional, em simulação, sendo o método de localização local mais preciso, dentre os métodos abordados neste trabalho, conforme pode ser visto nas Figuras 13(a) e 13(b). No entanto este método apresentou dois empecilhos, a dependência de características visuais para a identificação e rastreamento, e a necessidade de se conhecer a posição inicial do robô dentro do ambiente, uma vez que, por se tratar de odometria, ainda fica limitado à localização local, por depender de um sensor incremental.



(a) Localização no eixo X. (b) Localização no eixo Y.

Figura 13: *Gráfico da localização, obtido pelos algoritmos de Monte Carlo, Odometria Tradicional e Odometria Visual.*



(a) Localização no eixo X. (b) Localização no eixo Y.

Figura 14: *Odometria visual passa a retornar valores inválidos ao não ter nenhum objeto dentro do campo de visão.*

Outro ponto problema encontrado nesta abordagem foi que o algoritmo utilizado para obter a odometria visual, apesar do ótimo desempenho em condições ideais de cena, quando as mesmas não eram satisfeitas, a odometria perdia qualidade ao se aproximar o suficiente de uma superfície para que a mesma saísse do campo focal da câmera. Em algumas ocasiões, quando nenhum objeto se encontrava dentro do campo focal da câmera a odometria era retornada como zero, o equivalente a um valor invalido nesta implementação, como pode ser visto nas Figuras 14(a) e 14(b). A condição mais comum para observar este fenômeno foi obtida aproximando o robô de uma parede, limitando seu campo de visão somente a parede, a qual estava além do campo focal, de modo que nenhuma característica era detectável pela câmera. O grande problema deste comportamento é que, quando isso acontece, isso gera uma descontinuidade do sinal da odometria, o que não é aceitável, conforme discutido em 3.1, de modo que a mesma não foi utilizada como entrada do Algoritmo de Monte Carlo. No entanto isto poderia ser resolvido fazendo a fusão entre a odometria obtida através dos encoders e a odometria visual, utilizando o filtro de Kalman.

Conforme comentado anteriormente a qualidade da iluminação em que o robô se

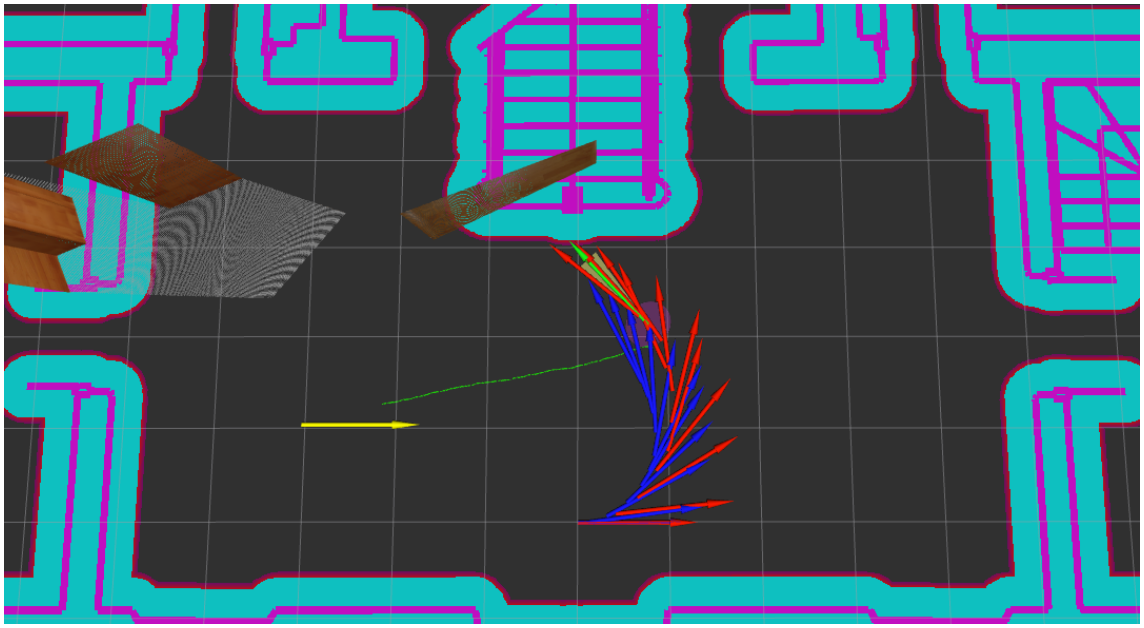


Figura 15: *Trajetória curvilineá estimada e real apresentada no mapa do ambiente simulado.*

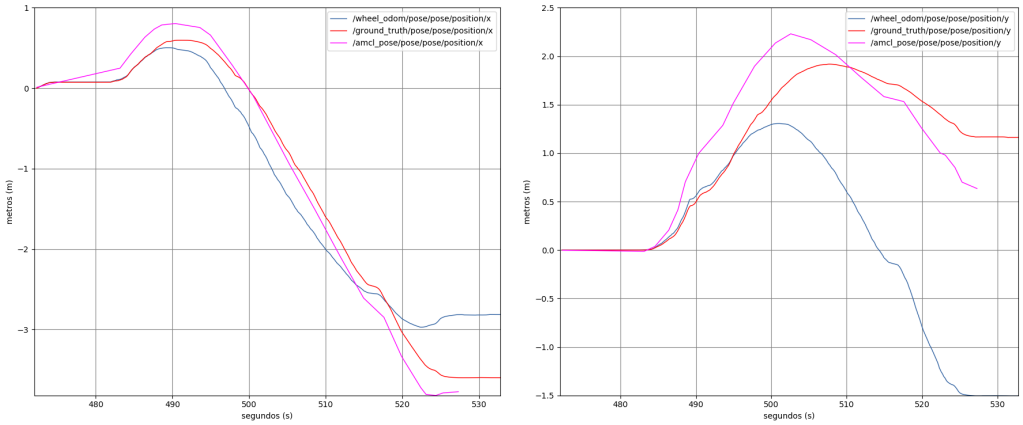
encontra influencia drasticamente na qualidade do sinal da odometria visual, e apesar de na simulação isso ocorrer apenas em casos em que o robô se aproximou das paredes o suficiente para as mesmas se encontrassem fora do campo focal da câmera, em situações reais esses complicadores seriam muito mais comuns. Isso porque na simulação pôde se configurar a iluminação de modo que todos os objetos fossem irradiados da mesma forma, o que drasticamente diminui a probabilidade de que entre duas imagens consecutivas nenhuma caraterística seja mantida para calcular o deslocamento.

4.3 Localização por Monte Carlo

O algoritmo de Monte Carlo apresentou uma precisão intermediaria, quando comparando com as duas odometrias citadas anteriormente, tendo o erro máximo em torno de 50 cm. Este erro, no entanto, não divergiu, ao contrário da odometria tradicional, por outro lado, em dados momentos, este erro se corrigiu, devido as propriedades probabilísticas deste filtro de partículas. Outra vantagem apresentada por este método, foi que o mesmo utiliza entradas de um escâner laser, o que poderia remover a preocupação de iluminação adequada, que é requisitada pela odometria visual, caso os dados do escâner laser não fossem obtidos por uma imagem RGB-D, e sim dirimente de um escâner.

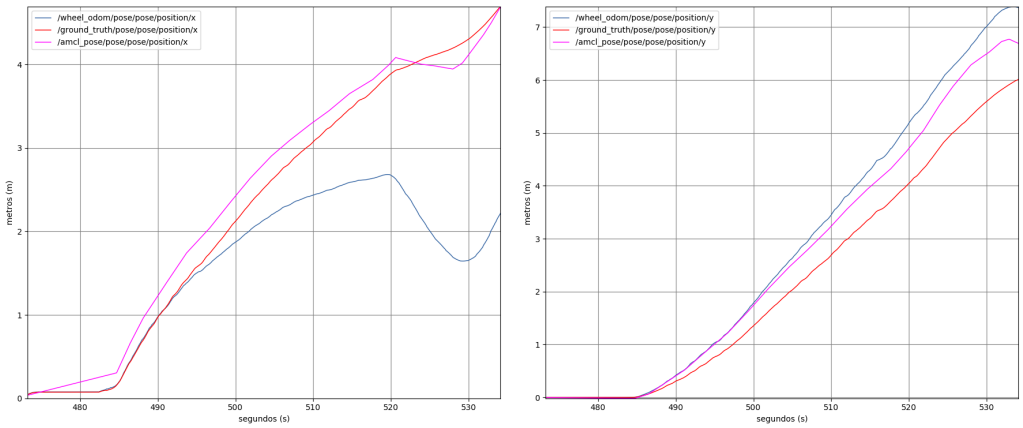
4.3.1 Monte Carlo aplicado à Localização Global

Nas Figuras 16(a), 16(b), 17(a) e 17(b) pode-se verificar que o erro máximo deste método de localização, antes do filtro convergir, em um dado eixo foi de, aproximadamente 0,5 m. Além disso, este erro foi corrigido após o filtro de Monte Carlo receber leituras do escâner laser com características mais identificáveis. Isso acontece pois, no ponto da trajetória com maior erro, o qual pode ser visto na Figura 15, o campo de visão do robô está ocupado, quase que completamente, por superfícies planas, as quais não são diferenciáveis do resto do ambiente devido à distância das mesmas para o robô, e então fornecem uma baixa confiança para a localização do robô.



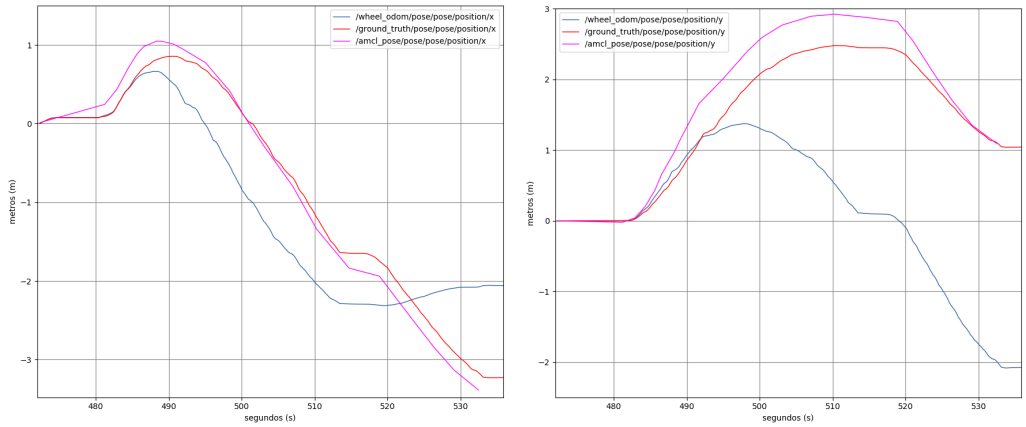
(a) Localização no eixo X. (b) Localização no eixo Y.

Figura 16: *Localização pelo algoritmo de Monte Carlo em uma trajetória curvilineá.*



(a) Localização no eixo X. (b) Localização no eixo Y.

Figura 17: *Localização pelo algoritmo de Monte Carlo em uma trajetória retilínea.*



(a) Localização no eixo X. (b) Localização no eixo Y.

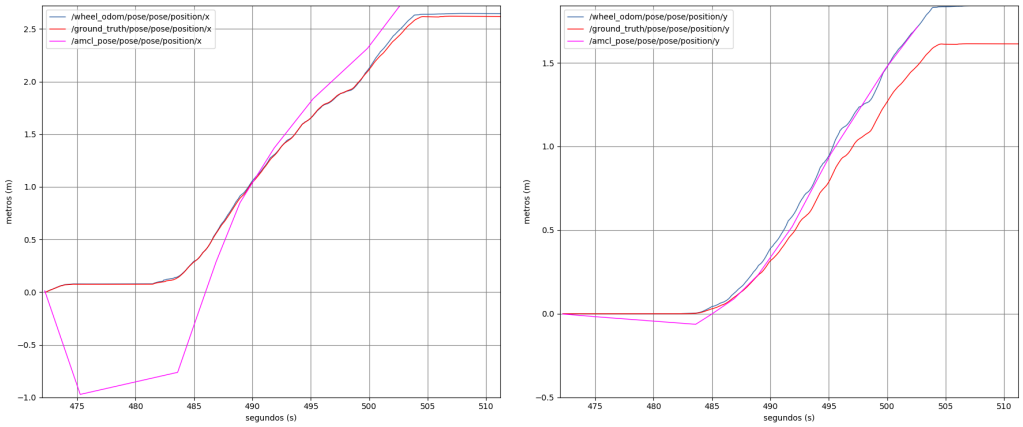
Figura 18: *Localização pelo algoritmo de Monte Carlo em uma trajetória curvilineá, após a alteração nos parâmetros do modelo de movimento.*

Os resultados obtidos para a localização em uma trajetória curvilínea, vistos nas Figuras 16(a) e 16(b), demonstraram que o intervalo de convergência do filtro de partículas estava aquém do esperado. Para corrigir este desempenho se optou por aumentar o parâmetro α_1 em aproximadamente quatro vezes, para permitir que as partículas utilizadas pelo filtro pudessem representar erros de rotação detectados por odometria com maior amplitude. Após esta modificação a localização pelo método de Monte Carlo, para a mesma trajetória curvilínea apresentou melhor desempenho, especialmente na correção da componente rotacional da odometria, conforme pode ser visto nas Figuras 18(a) e 18(b).

4.3.2 Monte Carlo aplicado ao Robô Sequestrado

Por fim, o benefício mais impactante deste método foi a capacidade de localização global e a resiliência do método, quando submetido a casos de robô sequestrado. O método de Monte Carlo não tem garantia de solucionar o caso do robô sequestrado, mas em condições especificas essa solução é possível de ocorrer(THRUN; BURGARD; FOX, 2005), conforme pode ser visto nas Figuras 19(a) e 19(b). As condições observadas em que este método foi capaz de solucionar o caso do robô sequestrado foram quando o sequestro ocorreu para regiões próximas a região de coberta pelas partículas do filtro.

Isso se deve ao fato deste ser o único método, entre os abordados, que utiliza uma comparação entre os dados da leitura visual e o mapa fornecido, dentro do seu processo de localização. Nos ensaios simulados a maneira mais simples de excitar esse comportamento foi, durante a execução da trajetória planejada pela pilha de navegação, manualmente introduzir uma perturbação de localização, injetando uma nova pose como a localização atual do robô através do serviço *2D Pose Estimate*, o qual publica uma nova pose, a qual é então utilizada pelo filtro de partículas como estado anterior na sua próxima iteração. Na Figuras 19(a), 19(b) e 20 pode se identificar a inclusão desta perturbação, assim como a correção da localização após algumas iterações do algoritmo de localização. No entanto, vale a pena ressaltar que esta correção não ocorreu imediatamente, de modo que, caso a posição injetada fosse a posição de destino, o erro não seria rejeitado, até que o destino fosse atualizado. Isso ocorre, pois, o algoritmo de localização tem seu ciclo dependendo da variação de posição detectada pela odometria, e uma vez que o erro de posição, detectado pelo controle de posição, é zero, o robô cessa seu movimento, de modo que a odometria



(a) Localização no eixo X. (b) Localização no eixo Y.

Figura 19: *Gráfico da correção da localização pelo algoritmo de Monte Carlo.*

não varia mais, e o ciclo do algoritmo de localização não é executado para corrigir o erro de localização.

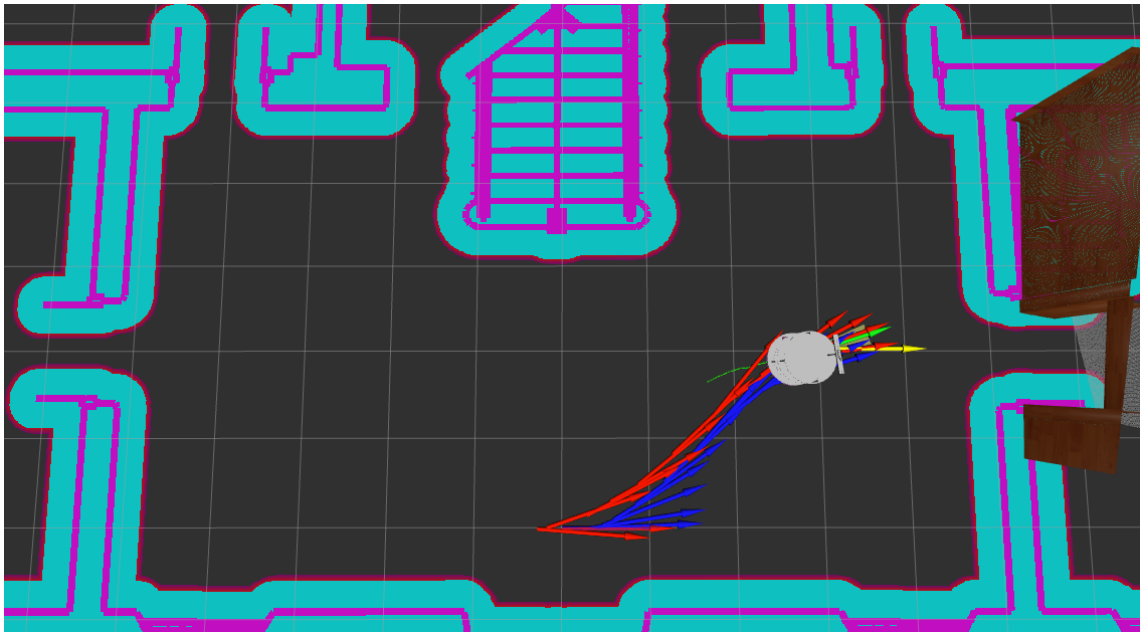


Figura 20: *Trajetória da correção da localização pelo algoritmo de Monte Carlo, representada sobre o mapa do ambiente.*

5 CONCLUSÃO

Ao final, observando o conjunto do que foi desenvolvido e testado ao longo desse trabalho, pode-se dizer que o objetivo inicial proposto de estudar um robô já existente e ampliar suas capacidades de software por meio da integração de um algoritmo de localização foi cumprido. O algoritmo integrado é capaz de solucionar os dois problemas de localização mais comuns, e, em casos específicos, solucionar o problema de localização do robô sequestrado.

A metodologia aplicada nesse trabalho incluiu o uso de diversas ferramentas e sistemas desenvolvidos por diferentes autores, como o simulador Gazebo, o modelo do robô Twil a pilha de navegação, diferentes fontes de odometria e o algoritmo de Monte Carlo. Isso demonstra como a modularidade proporcionada pelo ROS auxilia o desenvolvimento de projetos na área da robótica, já que seu *framework* comum facilita a integração de múltiplas funcionalidades sem que a questão da compatibilidade se torne um problema, podendo integrar diferentes nodos para um mesmo proposito, e avaliar qual oferece o melhor desempenho.

O uso de bibliotecas presentes no ROS que implementam soluções frequentemente necessárias em sistemas robóticos, como o `Adaptative Monte Carlo Location`, ilustra a possibilidade de trabalhar com conceitos complexos tais como a robótica probabilista através de ferramentas com alto nível de abstração, sendo assim acessíveis à diferentes tipos de desenvolvedores e usuários. A disponibilidade destes algoritmos autocontidos também oferece uma boa forma de aproximação a tópicos mais complexos, uma vez que é possível simular um robô, com todos os seus sensores, algoritmos de rotas, localização e mapeamento, sem que o usuário precise implementar todos estes métodos, e enquanto o desenvolvedor faz a integração destes diferentes algoritmos, ele pode se familiarizar com o funcionamento dos mesmos, para no futuro, poder contribuir com os seus próprios algoritmos.

Os experimentos realizados a fim de avaliar o desempenho do sistema de localização evidenciam a robustez do algoritmo escolhido bem como a amplitude de ferramentas disponíveis no ROS. A robustez do algoritmo já foi discutida na seção 4.3, mas as ferramenta de visualização de dados como o *rqt*, utilizado para gerar tanto os gráficos das coordenas dos métodos de localização, o grafo computacional e as relações entre os sistemas de coordenas, e o *rviz*, utilizado para visualizar o resultado dos diferentes métodos de localização referenciados em seus respectivos sistemas de coordenadas, se mostrou uma ferramenta muito útil, sem a qual este trabalho teria ordens a mais de complexidade

A partir do que foi desenvolvimento nesse projeto, um próximo trabalho possível seria a integração do algoritmo de localização com um algoritmo de mapeamento para permitir a utilização de diferentes ambientes, tanto simulados quanto reais. Uma segunda abordagem possível é o incremento de sensores para melhor a precisão da localização,

utilizando outras formas de sensores incrementais, como a própria odometria visual, ou sensores inerciais, para então fusionar a odometria destes diferentes sensores, e utilizar o Filtro de Monte Carlo para realizar correções mais suaves na localização do robô. Outro trabalho possível é algoritmo de localização ser integrado no robô real, podendo avaliar as diferentes formas de ruído que não foram modeladas e ajustar os parâmetros de variância de ruído de acordo. Por fim, o modelo simulado do robô, bem como os seus controladores e diversos algoritmos integrados a ele podem ser portados para o ROS 2.

REFERÊNCIAS

BARROS, T. T. T. *Modelagem e implementação no ros de um controlador para manipuladores móveis*. 2014. Dissertação de Mestrado – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil.

DUDEK, G.; JENKIN, M. *Computational Principles of Mobile Robotics*. 2nd. USA: Cambridge University Press, 2010. ISBN 0521692121.

FOX, D. et al. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In: HENDLER, J.; SUBRAMANIAN, D. (Ed.). *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*. [S.l.]: AAAI Press / The MIT Press, 1999. p. 343–349. Disponível em: <http://www.aaai.org/Library/AAAI/1999/aaai99-050.php>.

GERKEY, B. P. *amcl Documentation.* [S.l.: s.n.], 2021. Acessado em 25 de outubro de 2021 [Online]. Disponível em: <http://wiki.ros.org/amcl>.

INTEL REALSENSE. *Intel Depth Camera D435*. Disponível em: <https://www.intelrealsense.com/depth-camera-d435/>. Acesso em: 24 out. 2019.

LABBE, M. *rtabmap parameters.* [S.l.: s.n.], 2021a. Acessado em 23 de setembro de 2021 [Online]. Disponível em: <https://github.com/introlab/rtabmap/blob/master/corelib/include/rtabmap/core/Parameters.h#L161>.

LABBE, M. *rtabmap_ros Documentation.* [S.l.: s.n.], 2021b. Acessado em 20 de setembro de 2021 [Online]. Disponível em: <http://wiki.ros.org/rtabmap_ros>.

LABBÉ, M.; MICHAUD, F. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, v. 36, n. 2, p. 416–446, abr. 2018.

LAGES, W. F. A Backstepping Non-Smooth Controller for ROS-based Differential-Drive Mobile Robots. In: KOUBAA, A. (Ed.). *Robot Operating System (ROS): The Complete Reference (Volume 3)*. Cham, Switzerland: Springer International Publishing, 2018. (Studies in Computational Intelligence).

MARDER-EPPSTEIN, E. *Setup and Configuration of the Navigation Stack on a Robot.* [S.l.: s.n.], 2021. Acessado em 20 de setembro de 2021 [Online]. Disponível em: <http://wiki.ros.org/navigation/Tutorials/RobotSetup>.

MEEUSSEN, W. *Coordinate Frames for Mobile Platforms*. [S.l.: s.n.], 2010. Public Domain. Disponível em: <https://www.ros.org/reps/rep-0105.html>.

PETRY, G. R. *Navegação de um robô móvel em ambiente semi-estruturado*. 2019. Trabalho de Conclusão de Curso – Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005.