UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Pedro Henrique Rodrigues Procópio Corrêa

Localização de um robô autônomo com filtro de partículas em campo de futebol simulado

Uberlândia, Brasil
2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Pedro Henrique Rodrigues Procópio Corrêa

Localização de um robô autônomo com filtro de partículas em campo de futebol simulado

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Engenharia Mecatrônica.

Orientador: Prof. Dr. Rogério Sales Gonçalves

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Mecânica

Bacharelado em Engenharia Mecatrônica

Uberlândia, Brasil

2025

Pedro Henrique Rodrigues Procópio Corrêa

Localização de um robô autônomo com filtro de partículas em campo de futebol simulado

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Engenharia Mecatrônica.

Trabalho aprovado. Uberlândia, Brasil, 23 de maio de 2025:

Prof. Dr. Rogério Sales Gonçalves

Orientador

Prof. Dr. José Jean-Paul Zanlucchi de Souza Tavares

Profa. Dra. Guênia Mara Vieira Ladeira

Uberlândia, Brasil 2025

Agradecimentos

Agradeço primeiramente a meus pais, Maria Amélia e José Procópio, pela minha criação, pelo carinho, pelas oportunidades que me deram e, principalmente, pela paciência com minha graduação. O trajeto foi árduo, mas nunca senti desamparo, nunca senti abandono, somente amor e inspiração. Agradeço às minhas tias e minha avó Jane, sem elas este trabalho nunca seria produzido, pois sem seu apoio nunca chegaria aonde estou. Agradeço aos meus irmãos Giovanni, Lipe, João Victor, Ana Clara e Maria Paula que, apesar da distância, foram algumas das maiores motivações na minha jornada. Agradeço ao meu padrasto e madrasta, que sempre me trataram como filho. Agradeço ao restante da família e garanto que todos vocês foram importantes.

Deixo um agradecimento a todos que não poderei compartilhar esse momento, em especial à minha madrinha Tereza: os anos passam, mas ainda é impossível superar a falta.

À mulher com quem escolhi, e escolho todos os dias, compartilhar a vida, obrigado Brenda. Obrigado pela paciência, ternura, cuidado, amor. Obrigado por me lembrar de tomar água, dormir cedo e comer direito enquanto faço esse projeto, obrigado por estar do meu lado todos os dias incondicionalmente. Obrigado. Estendo estes agradecimentos aos meus sogros, Adalgiza e Jeferson, que praticamente me adotaram nos últimos anos.

Agradeço aos meus amigos Anthero, Noah, Renan, Raphael, Caio e tantos outros: vocês sempre tornaram a caminhada mais prazerosa. Obrigado pelo apoio e pelas risadas, espero ser para vocês o companheiro que foram para mim.

Ao meu orientador, professor Rogério, agradeço pela trajetória, desde equipe de competição até a finalização deste documento, pela paciência e compreensão em momentos de baixo rendimento, pelas palavras em situações de pânico e pelo auxílio nesse projeto. Estendo o agradecimento ao professor Pedro que, em um momento de poucos resultados, me fez entender que para fazer o complexo, primeiro precisamos entender o básico.

À EDROM agradeço pelas experiências, alegrias, desespero, frustração e risadas. Na equipe conheci amigos para a vida, conheci minha companheira, descobri minhas paixões na engenharia e meus ódios, evolui como profissional e pessoa. A inspiração deste trabalho é tentar devolver, mesmo que pouco, algo à equipe, pois a dívida que tenho com vocês é imensurável. "*Isso aqui não é uma família, somos meros colegas de trabalho"*. Foi um prazer trabalhar com todos vocês.

Por fim, agradeço a todos que fizeram parte da minha trajetória, mesmo que indiretamente. No primeiro período da mecatrônica meus veteranos me ensinaram que ninguém chega a lugar nenhum sozinho, e eu certamente não chegaria aqui sem vocês.

Resumo

Este trabalho propõe um sistema de localização de um robô autônomo, equipado com uma câmera, em um campo de futebol de duas dimensões simulado. O sistema se baseia na Localização de Monte Carlo (MCL), que utiliza um filtro de partículas para emular o comportamento do robô em um ambiente conhecido e estimar sua posição a partir de comparações das leituras da visão simulada com a biblioteca OpenCV das partículas e do robô. Para garantir convergência e solucionar o problema de sequestro de robô, foram utilizadas técnicas da *Extended* MCL para avaliar a qualidade das amostras e determinar o método correto de reamostragem a cada iteração. O projeto obteve uma localização funcional, porém dependente de identificar pontos de referência em casos de sequestro (ou na iteração inicial do código).

Palavras-chave: Localização de Monte Carlo, Visão computacional, Robótica móvel

Abstract

This paper proposes an autonomous robot, equipped with a camera, localization pipeline in a simulated two dimensions soccer field. The pipeline implements Monte Carlo Localization (MCL), using a particle filter to emulate a robots behavior in a known environment and estimate its position based on its, and the particles', simulated vision readings with OpenCV library. To guarantee proper convergence and to solve robot kidnapping cases, Extended MCL technics were implemented to correctly determine resampling method and assess samples quality in each iteration. The project obtained a working localization pipeline, although dependent on identifying landmarks in case of robot kidnapping (or in pipeline starting iteration).

Palavras-chave: *Monte Carlo Localization, Computer Vision, Mobile Robotics*

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

R.U.R.	*Rossumovi Univerzální Roboti (*Robôs Universais de Rossum*)*
IFR	*International Federation of Robotics*
MCL	*Monte Carlo Localization*
EMCL	*Extended Monte Carlo Localization*
ESS	*Effective Sample Size*
FOV	*Field of View*
SRL	*Sensor Resetting Localization*
POV	*Point of View*

Lista de símbolos

x	Coordenada de localização no eixo horizontal
y	Coordenada de localização no eixo vertical
θ	Direção da partícula ou robô no plano
k	Índice de iteração do filtro de partículas
$pos = (x, y, \theta)$	Vetor de localização de partícula ou robô
N ou $N_{particula}$	Quantidade de partículas ou amostras
i	Índice da partícula
W ou W_k	Conjunto de pesos das partículas em uma dada iteração
w ou w_i	Peso de uma determinada partícula
u	Conjunto de índices para reamostragem
$\hat{u}$	Defasagem aleatória para determinação de índices para reamostragem
ns	Número de novas partículas introduzidas ao filtro
$\widetilde{w}$	Média dos pesos não normalizados
w_t	Limite arbitrário para SRL
ESS	*Effective Sample Size*
$var()$	Cálculo de variância
$E()$	Cálculo de média simples
H	Entropia do conjunto de pesos normalizados
λ	Limite arbitrário para determinação de *over-convergence*
H_p	Entropia das partículas antes da etapa de avaliação dos pesos
H_c	Entropia das partículas depois da etapa de avaliação dos pesos
Δ_{pos}	Variação da posição do robô
$\overline{pos}$ ou $\overline{pos}_k$	Estimativa da posição do robô em uma determinada iteração
$N_{sensores}$	Quantidade de direções para medição de distância
j	Índice de direção da medição de distância
σ^2	Variância
σ	Desvio padrão
$()_{SR}$	Conjunto de partículas ou pesos para aplicação de *sensor resetting*

Sumário

1 Introdução

Na Grécia antiga, Diógenes de Sínope caçoou de Platão por definir o homem como um animal bípede e sem penas (R.D. Hicks, 1972). Tal afirmação, apesar de absurda no pensamento contemporâneo, recompensou o filósofo ateniense com aplausos de seus colegas de academia. A primeira aparição do termo "robô" também difere muito dos conceitos atuais, sendo citada pela primeira vez na peça de Karel Čapek (2001) R.U.R. e utilizada para referenciar criaturas artificiais criadas para substituir os humanos no âmbito do trabalho. Para as definições atuais, as criaturas da peça se assemelham a clones, enquanto a Federação Internacional de Robótica (IFR) adota a norma ISO 8373, categorizando robôs como um mecanismo mecânico programado com certo grau de autonomia para realizar locomoção, manipulação ou posicionamento.

Apesar da definição atual abranger algo amplo e robôs presentes no dia a dia realizarem tarefas simples e objetivas, o fascínio com androides continua no imaginário popular e motiva inúmeros avanços científicos. Baseado nisso, em 1997 foi criada a RoboCup, uma iniciativa científica internacional com o objetivo de fomentar o desenvolvimento tecnológico no ramo de robótica móvel através de competições de futebol de robôs, com o objetivo final de vencer uma seleção de futebol humana até o ano de 2050 (RoboCup, 2025). Tal objetivo carrega inúmeros desafios de robótica móvel, sendo um deles a localização autônoma dos robôs no campo.

Este trabalho visa o desenvolvimento, validação e experimentação de um sistema de localização autônomo baseado na Localização de Monte Carlo (Fox *et al.*, 1999), que será referenciada como MCL (*Monte Carlo Localization*) nesse documento. O objetivo final do trabalho é um correto rastreamento do robô, uma vez localizado, e sua recuperação após raptado, que será chamado de *robot kidnapping* (sequestro de robô). As duas métricas relevantes ao projeto são a performance, visto que a MLC exige um alto processamento para melhores resultados, e a complexidade do algoritmo. Como a MLC exige a avaliação de uma informação sensorial, este trabalho propõe a utilização apenas de uma câmera simulada como sensor para medições de distância e de medições de deslocamento para monitoramento da caminhada do robô. Conforme as regras da RoboCup, o algoritmo visa operar em robôs autônomos independentes, sem comunicação externa ou entre robôs.

2 Objetivos

A inspiração do trabalho parte da necessidade de rastrear a posição de um robô humanoide em campo para a competição da RoboCup de futebol de robôs humanoides. O projeto tem como finalidade simular em duas dimensões o comportamento de um robô de três dimensões, portanto certos parâmetros, como alcance de visão do robô e ambiente do campo de futebol, terão como embasamento o cenário real, operando como limitadores do desafio de localização.

Como objetivo, propõe-se o desenvolvimento de uma base para resolver o problema da localização autônoma na forma de um algoritmo capaz de ser adaptado para a competição. Este projeto não almeja a concepção de um código para implementação em robôs reais, mas a validação do MCL no cenário proposto, estudo de seus parâmetros e entendimento das diversas técnicas propostas para aprimoramento de seus resultados através de simulação.

Para avaliar o sucesso do tabalho, performance e complexidade do algoritmo serão as métricas qualitativas e, portanto, subjetivas. Como métricas quantitativas, deseja-se obter um sistema com rápida convergência, tempo entre execuções razoável e baixo erro em regime permanente.

3 Fundamentação Teórica

3.1 Localização de Monte Carlo

Dado um robô arbitrário em um espaço bidimensional conhecido, sua localização pode ser representada pela sua posição (x, y) no plano e uma direção θ. Fox *et al.* (1999) propõem um método denominado Localização de Monte Carlo para a localização e rastreamento desse robô. O método se baseia em técnicas de *importance resampling* (reamostragem por importância) (Rubin 1988), também chamado de *Particle Filter* (filtro de partículas).

Fox *et al.* (1999) definem em uma determinada iteração k uma posição no espaço apresentado como $pos = (x, y, \theta)$ e criam um conjunto de N partículas $Particulas_k = \{particula_i \mid i = 1, \cdots, N\}$, sendo cada partícula $particula_i = ((x, y, \theta), w)$, com w sendo um peso numérico do quão bem a partícula representa a localização real do robô, normalizado de modo que $W_k = \sum_{i=1}^{N} w_i = 1$. Inicialmente, e após cada operação de reamostragem, o peso das partículas é definido de maneira uniforme, com valor $1/N$.

A cada iteração do filtro de partículas são coletadas informações de deslocamento do robô, para realizar a **predição** do deslocamento das partículas (com propagação de erro baseado nos erros de leitura de deslocamento), e dos sensores do robô, para **avaliação** das partículas e determinação de seus novos pesos. Com os pesos normalizados, é feita a **reamostragem** das partículas de modo que pesos maiores resultam em maiores chances de uma cópia da partícula estar presente no novo conjunto $Particulas_k$ de N partículas. Tal abordagem resulta na eliminação de amostras indesejáveis e aumenta a quantidade de amostras em regiões cuja probabilidade de representação da localização real do robô é maior. Um esboço do algoritmo base é proposto por Shang e Sung (2007) na Tabela 1.

Tabela 1 - Algoritmo MCL Convencional

1.	**Etapa de predição:**
2.	Atualiza $Particulas_k$ com base em $Particulas_{k-1}$ e a locomoção lida do robô
3.	Reinicia os N pesos como $w = 1/N$
4.	**Etapa de atualização de sensor:**
5.	Atualiza W_k comparando as medidas de sensor das partículas com as do robô
6.	**Etapa de reamostragem:**
7.	Atualiza $Particulas_k$ baseado nos pesos W_k utilizando ***importance resampling***

Fonte: Shang e Sun (2007), adaptado

Havendo uma avaliação razoavelmente confiável dos sensores para atualização dos pesos o algoritmo converge com velocidade proporcional à quantidade de partículas utilizadas (Tanner, 1993). Uma quantidade pequena de partículas pode, além de afetar a velocidade de convergência, ocasionar resultados imprecisos ou incorretos, porém um alto número de amostras acarreta uma grande carga de processamento e demora na execução do código.

No artigo de Fox *et al.* (1999) sobre MCL ele aborda a utilização de uma quantidade variável de partículas, visto que para rastreamento do robô são necessárias menos amostras (resultando em melhor performance), enquanto no momento inicial ou em situação de *robot kidnapping* o algoritmo desempenha melhor com uma maior quantidade. Ele conclui que em cenários de grandes incertezas uma quantidade adaptável de partículas se mostra eficiente, o que não é o caso deste trabalho. Almeida, Neto e Bianchi (2018) posteriormente comparando novas abordagens de MCL também concluem que os ganhos, em cenários controlados, de um valor flutuante de partículas são pequenos. Visto o interesse em uma abordagem de menor complexidade, este trabalho utilizou uma quantidade **fixa** de partículas.

Parafraseando Shang e Sun (2007), o MCL tradicional enfrenta problemas quando as observações do robô divergem da distribuição proposta das partículas, seja por um distúrbio não modelado (*robot kidnapping*, por exemplo), uma quantidade insuficiente de partículas ou um sensor de baixo ruído que, apesar de contra intuitivo, resulta em performance pior do filtro de partículas, comumente resultando em uma rápida convergência a um local inadequado, cenário denominado *over-convergence* (convergência excessiva).

Para atacar as diversas situações possíveis no filtro de partícula, são necessárias técnicas adequadas para categorização de tais cenários e para correta reamostragem das partículas. Para isso, este trabalho emprega *Extended MCL* (Localização de Monte Carlo Estendida), abreviada como EMCL por Shang e Sung (2007), que será melhor explicada posteriormente, visando obter os melhores resultados.

3.2 Métodos de reamostragem

A Localização de Monte Carlo, conforme previamente apresentada, é um filtro de partículas e, portanto, depende de reamostragem para convergência. O método convencionalmente utilizado em grande parte da literatura é o *sequential importance resampling*, que utiliza os pesos das amostras calculados para determinar o novo conjunto de

amostras, aumentando a confiabilidade da solução e reduzindo gradualmente a variância dos dados (Doucet 1998).

Hol *et al.* (2006) comparam métodos de reamostragem em filtros de partícula e concluem que *systematic resampling* (reamostragem sistemática) é menos complexa, mais leve e gera resultados de maior qualidade em certos cenários. O documento exemplifica o método gerando N números a partir da equação (1) como índices das amostras atuais para compor o conjunto futuro de amostras, com $\hat{u}$ sendo um desvio aleatório, i o índice da partícula (variando então de 1 a N), e u_k o conjunto de posições a serem selecionadas do conjunto de pesos normalizados para reamostragem. Tal abordagem garante uma distância $1/N$ entre amostras e depende da geração de somente um número aleatório, resultando em uma reamostragem de qualidade e eficiente.

$$u_k = \frac{(i - 1) + \hat{u}}{N} \text{, com } \hat{u} \sim U[0,1) \tag{1}$$

Enquanto a *importance resampling* utiliza informações das partículas atuais e seus respectivos pesos para gerar um novo conjunto refinado, Lenser e Veloso (2000) propõem uma técnica denominada *Sensor Resetting Localization* (localização por recomposição de sensor) e referida na literatura como SRL que introduz uma maneira na MCL de gerar partículas com base puramente nas informações do sensor e o conhecimento do ambiente em que o robô está (ou em pontos de referência do mesmo). O método consiste na substituição de partículas no conjunto em regiões de maior probabilidade de localização do robô com base em cálculos com as leituras de sensores, como operações de triangulação baseadas em pontos de referência.

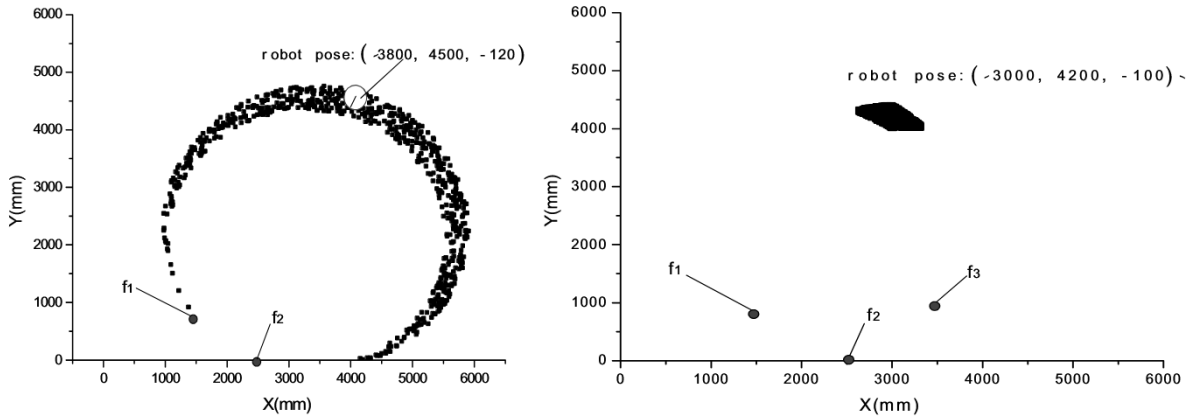
Gutmann e Fox (2002) simplificam a equação de cálculo da quantidade de amostras ns a serem substituídas conforme a equação (2), onde N é o número de partículas, $\widetilde{w}$ a média dos pesos das partículas normalizado pela quantidade de amostras e w_t um limite empírico.

$$ns = N * \max\left(0{,}1 - \frac{\widetilde{w}}{w_t}\right) \tag{2}$$

Um exemplo visual de SRL em prática pode ser observado na Figura 1 de Shang e Sung (2007), em que utilizam de pontos de referência, conseguindo delimitar regiões de alta

probabilidade com qualidades diferentes baseadas na quantidade de pontos de referência observados.

Figura 1 - Exemplo de sensor resetting com dois e três pontos de referência, respectivamente



Fonte: Shang e Sung (2007)

Contudo, SRL depende do identificar pontos de referência, o que torna o método inutilizável caso referências não estejam disponíveis. Com isso, faz-se necessária a utilização conjunta de *importance resampling*, eficaz no rastreio do robô, e *sensor resetting*, quando disponível, para recuperar a localização em caso de *kidnapping*.

3.3 *Extended Monte Carlo Localization*

Shang e Sung (2007) propõem uma etapa de validação dupla no momento de realizar a reamostragem do filtro, a fim de determinar o melhor método a ser empregado para gerar as novas partículas (*sensor resetting* ou *importance resampling*) bem como a quantidade de novas amostras.

A primeira etapa é a validação de *over-convergence* (*over-convergence validation*), que é realizada considerando $w(i)$ o conjunto de pesos normalizados e analisando a quantidade efetiva de amostras (*effective sample size,* ou *ESS*), calculado por Liu *et al.* (2001) com as equações (3) e (4), em que cv é o coeficiente de variação, $var(w(i))$ a variância dos pesos, $E(w(i))$ a média dos pesos e N a quantidade de partículas, e a entropia dos pesos normalizados das amostras de acordo com a equação (5).

$$ESS = \frac{N}{1 + cv^2} \tag{3}$$

$$cv^2 = \frac{var\big(w(i)\big)}{E^2\big(w(i)\big)} = \frac{1}{N}\sum_{1}^{N}(N * w(i) - 1)^2 \quad (4)$$

$$H = -\sum(w_i * \log w_i) \quad (5)$$

Caso a quantidade efetiva de amostras seja inferior à um percentual da quantidade total (calibrado empiricamente), *over-convergence* é confirmado e a quantidade de amostras adicionadas através de *sensor resetting* é calculada pela equação (6), em que c é uma constante. Caso a *ESS* seja superior ao limite proposto, é verificada a entropia dos pesos normalizados, calculados pela equação (5), antes e depois da etapa de avaliação das leituras de sensores das partículas de acordo com a equação (7), com $\lambda \in (0,1)$ sendo um limite (também empiricamente calibrado), H_p e H_c a entropia dos pesos antes e depois da avaliação das partículas, respectivamente. Caso confirmada *over-convergence*, a quantidade de amostras adicionadas é resultado da equação (8).

$$n_s = c(N_k - ESS) \quad (6)$$

$$\frac{H_c - H_p}{H_p} \geq \lambda \quad (7)$$

$$n_s = (1 - \lambda)(N_k - ESS) \quad (8)$$

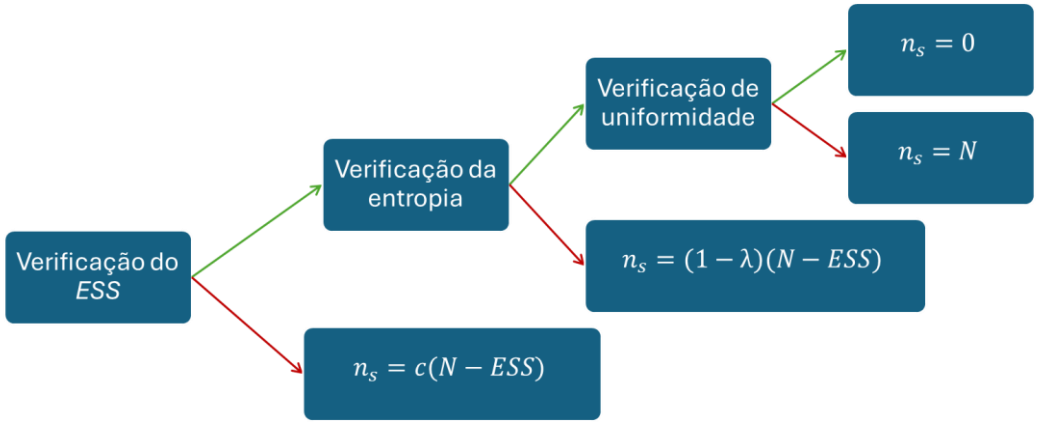
Quando não é observada a *over-convergence*, é realizada a segunda etapa de validação, chamada de *uniformity validation* (validação de uniformidade), comparando a somatória dos pesos não normalizados das amostras com um limite apropriado. Não atingindo o limite, todas as amostras sofrerão *sensor resetting*, isto é, $n_s = N_k$. Caso nenhuma das situações (*ESS* insuficiente, aumento na entropia das amostras ou disparidade entre medições do sensor das amostras com o robô) sejam observadas, o filtro de partículas se encontra em uma situação desejável, então todas as partículas passam somente por *importance resampling*. Um esboço do algoritmo é descrito na Tabela 2 e um fluxograma das validações está representado na Figura 2.

Tabela 2 - Algoritmo EMCL

1.	**Etapa de predição** (similar ao convencional)**:**
2.	Atualiza $Particulas_k$ com base em $Particulas_{k-1}$ e a locomoção lida do robô
3.	Reinicia os N pesos como $w = 1/N$
4.	**Etapa de atualização de sensor** (similar ao convencional)**:**
5.	Atualiza W_k comparando as medidas de sensor das partículas com as do robô
6.	**Etapa de reamostragem:**
7.	Se ocorreu *over-convergence* (*over-convergence validation*):
8.	Calcula-se n_s com base na situação validada
9.	Realiza-se *sensor resetting* em n_s amostras
10.	Realiza-se *importance resampling* em $N - n_s$ amostras
11.	Se não, se as amostras concordam com as observações (*uniformity validation*):
12.	Realiza-se *importance resampling* em todas as amostras
13.	Se não:
14.	Realiza-se *sensor resetting* em todas as amostras

Fonte: Shang e Sun (2007), adaptado

Figura 2 - Fluxograma de etapas de verificação

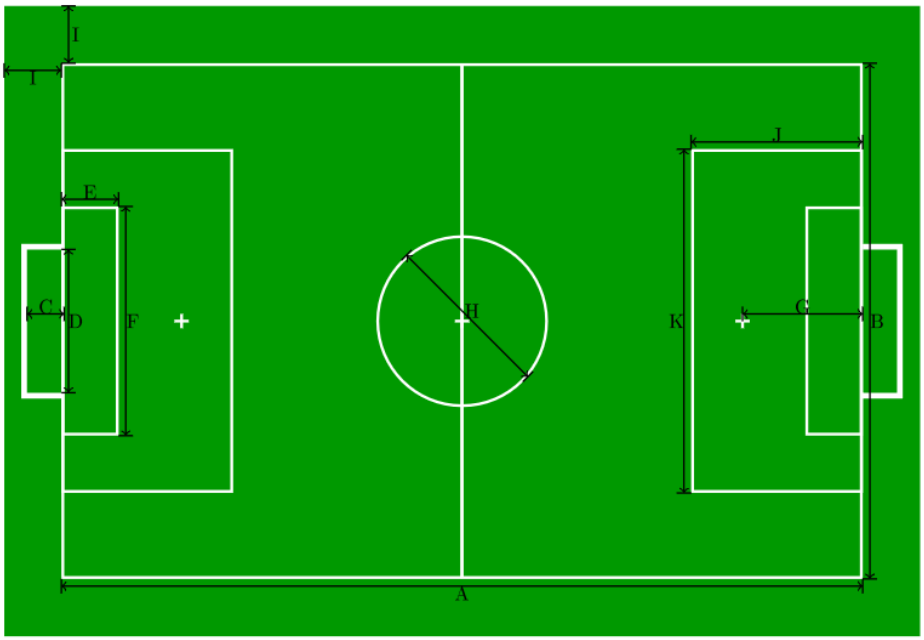


Fonte: do autor

3.4 Ambiente simulado

A Localização de Monte Carlo exige que o ambiente seja conhecido, portanto será utilizado um campo de futebol gerado com o código. Para realizar as simulações, foi utilizado como base o campo proposto para competição de futebol de robôs humanoides categoria kidsize da RoboCup 2025. A disposição do campo e suas medidas estão na Figura 3 e Tabela 3.

Figura 3 - Campo proposto para competição de futebol de robôs humanoides



Fonte: RoboCup (2025)

Tabela 3 - Dimensões do campo

Índice	Descrição	Tamanho kidsize	Tamanho adulto
A	Comprimento do campo	9 m	14 m
B	Largura do campo	6 m	9 m
C	Profundidade do gol	0,6 m	
D	Largura do gol	2,6 m	
	Altura do gol	1,2 m	1,8 m
E	Comprimento da pequena área	1 m	
F	Largura da pequena área	3 m	4 m
G	Marca de pênalti	1,5 m	2,1 m
H	Diâmetro do círculo central	1,5 m	3 m
I	Mínima distância para o campo	1 m	1 m
J	Comprimento da grande área	2 m	3 m
K	Largura da grande área	5 m	6m

Fonte: RoboCup (2025), adaptado

O campo foi simulado utilizando a biblioteca OpenCV do Python, sendo representado por uma imagem em que cada pixel equivale a 1 cm² do campo real, fazendo com que distâncias

medidas na imagem sejam equivalentes a distância proposta em centímetros. Tal opção faz com que as medidas, estimativas e tamanhos da simulação tenham resolução de 1 centímetro, o que se mostra suficiente devido ao escopo do cenário. Com isso, uma dada posição de robô ou partícula no campo é representada pelas coordenadas do pixel equivalente na imagem.

A cerca do sensor simulado, foi utilizado como base a câmera de profundidade Intel® RealSense™ D435, que possui um campo de visão (*field of view*, ou FOV) de 87° horizontal e 58° vertical. A câmera possui uma distância mínima para medições de profundidade de ~28 centímetros de acordo com seu manual e erro de 2% em medições quando a menos 2 metros de distância, com distância ideal de operação entre 0,3 e 3 metros.

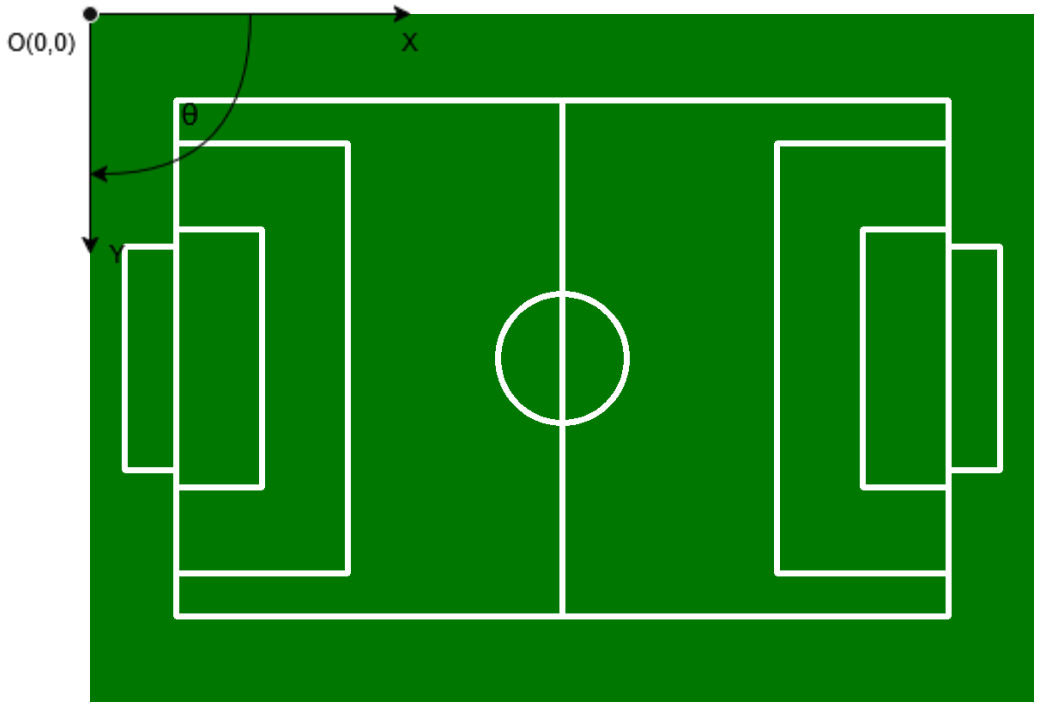
4 Desenvolvimento

4.1 Simulação

O código foi desenvolvido e testado em uma máquina virtual emulada com o software Oracle VirtualBox utilizando a distribuição do Ubuntu 22.04, configurada para utilizar 8GB de memória RAM, 6 núcleos de processador e 256 de memória de vídeo. Entretanto, não foi utilizado CUDA no código para a utilização da placa de vídeo, e o programa utiliza apenas um interpretador do Python e, consequentemente, depende apenas de um núcleo de processamento.

Conforme citado, o ambiente de simulação utilizado foi uma imagem do OpenCV gerada a partir dos parâmetros da RoboCup. O campo é formado a partir da definição de suas marcações com as medidas propostas, resultando na Figura 3. Por serem tratadas como matrizes, as coordenadas (x, y) do pixel que representa a localização de determinada partícula equivalem à $(coluna, linha)$ da matriz. Com isso, a origem do sistema de coordenadas é o vértice superior esquerdo, com o eixo Y crescendo na direção oposta do sentido convencional, conforme demonstrado na Figura 4.

Figura 4 - Campo simulado com OpenCV



Fonte: do autor

O ambiente de simulação e o filtro de partículas são executados em *threads* diferentes do Python. Apesar de não haver uma real separação dos processos, visto que o mesmo interpretador é responsável por todo o código, ainda foram observados ganhos de performance e principalmente de modularização dos serviços, pois em apenas uma *thread* gargalos no processamento do filtro de partículas resultavam em uma simulação lenta ou pouco responsiva. Com isso, a atualização visual da simulação ocorre a uma frequência de ~30 frames por segundo, enquanto a execução do filtro de partículas necessita de em média ~1 segundo para execução com 50 partículas, variando conforme situação e principalmente quantidade de partículas.

No próprio processo de geração do campo são determinadas as coordenadas das linhas que o compõe, bem como as do círculo central e a linha do meio do campo. Tais informações são relevantes posteriormente para avaliação das partículas de maneira eficiente e para o cálculo das regiões de maior probabilidade do robô para *sensor resetting*.

Inicialmente, a posição e orientação do robô no campo é gerada aleatoriamente obedecendo tendências previamente estabelecidas (a fim de atacar problemas de ambiguidade, ideia explorada posteriormente neste trabalho). A locomoção do robô é realizada através de comandos do teclado para deslocamentos de incremento fixos, definidos inicialmente.

4.2 Filtro de partículas

Dado uma execução k do filtro de partículas em que $pos_k = (x, y, \theta)$ e $\overline{pos}_k = (x, y, \theta)$ representam a localização real e estimada, respectivamente, do robô no simulador, $particula = (x, y, \theta)$ a posição de uma determinada partícula, $Particulas_k = \{particula_i \mid i = 1, \cdots, N\}$ o conjunto de amostras e $W_k = \{w_i \mid i = 1, \cdots, N\}$ seus respectivos pesos após a avaliação de suas leituras de sensor. A abstração do algoritmo do filtro de partículas empregado pode ser representada de forma abstrata pela Tabela 4.

Tabela 4 - Algoritmo Filtro de Partículas

1. $\Delta_{pos} = pos_k - pos_{k-1}$
2. $Particulas_k \leftarrow \boldsymbol{func\_mover\_particulas}(Particulas_{k-1}, \Delta_{pos})$
3. $sensores_{robo} \leftarrow \boldsymbol{func\_leitura\_sensores}(pos_k)$
4. $SE\ \boldsymbol{func\_pontos\_de\_referencia\_visiveis}(pos_k) == VERDADEIRO\ ENTAO$
5. $\quad Particulas_{SR}, W_{SR} \leftarrow \boldsymbol{func\_sensor\_resetting}(pos_k)$
6. $W_k \leftarrow \boldsymbol{func\_testar\_particulas}(Particulas_k, W_{k-1}, sensores_{robo})$

7. $\overline{pos}_k \leftarrow \boldsymbol{func\_estimar\_pos}(Particulas_k, W_k)$
8. $Particulas_k \leftarrow \boldsymbol{func\_reamostragem}(Particulas_k, W_k, Particulas_{SR}, W_{SR})$

Fonte: do autor

A partir disso, as funções destacadas podem ser descritas:

- Mover partículas ($func\_mover\_particulas$):
  Utiliza a variação na posição do robô em (x, y, θ) e propaga tal variação, aplicando um erro proporcional à distância percorrida, na posição de todas as amostras, a fim de reproduzir o comportamento do robô nas partículas.

- Leitura de sensores ($func\_leitura\_sensores$):
  A partir da posição do robô no campo, infere informações dos pontos de referência e calcula as medições de sensor utilizando o *point of view* (perspectiva, POV) do robô, processo que será detalhado na seção 4.2.1 deste trabalho.

- Pontos de referência visíveis ($func\_pontos\_de\_referencia\_visiveis$):
  Verifica se existem pontos de referência dentro do POV do robô, também detalhado na próxima seção.

- *Sensor resetting* ($func\_sensor\_resetting$):
  Havendo pontos de referência no campo de visão, calcula prováveis localizações no campo e seus respectivos pesos para reamostragem. A seção 4.2.3 deste documento aprofunda na metodologia adotada.

- Testar partículas ($func\_testar\_particulas$):
  De maneira análoga ao que foi feito para obter as medições de sensor do robô, obtém-se as medições das partículas e as compara com as do robô, a fim de determinar os pesos das amostras. A seção 4.2.2 traz detalhes de como é realizada as medições das partículas de maneira eficiente e do cálculo de seus pesos.

- Estimar posição ($func\_estimar\_pos$):
  Como os pesos das partículas representam o quanto suas leituras de sensores concordam com as do robô, a média das posições das partículas ponderada pelos seus respectivos pesos resulta numa aproximação da posição do robô, em determinada execução do filtro.

- Reamostragem ($func\_reamostragem$):

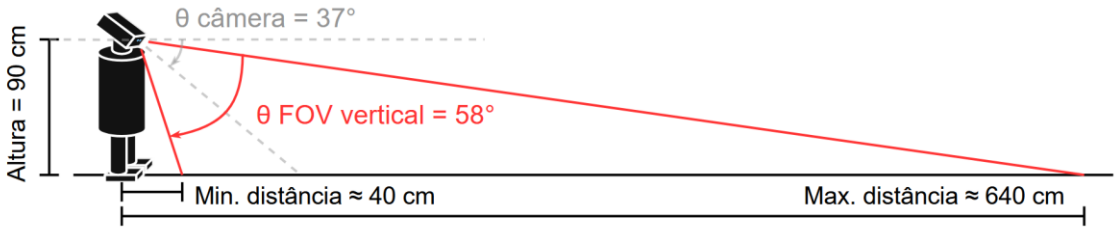
  A partir das informações obtidas na execução do filtro, realiza a reamostragem das partículas. A principal diferença do EMCL para o MCL convencional está no processo de reamostragem, conforme previamente citado. A maneira como a reamostragem do EMCL foi implementada nesse trabalho será descrita na seção 4.2.4.

As seções a seguir explicam com maior profundidade as etapas da solução implementada.

4.2.1 Perspectiva do robô

A fim de simular o campo de visão de um robô real, seu POV foi calculado a partir das medidas de FOV especificadas pela câmera base, uma altura hipotética de 90 centímetros (dentro dos parâmetros da RoboCup para a competição) e um ângulo de câmera fixo de 37°. Com isso, a distância máxima dentro do campo de visão do robô é de aproximadamente 640 centímetros e a mínima de 40 centímetros. A Figura 5 representa um esquema do alcance da perspectiva do robô simulado.

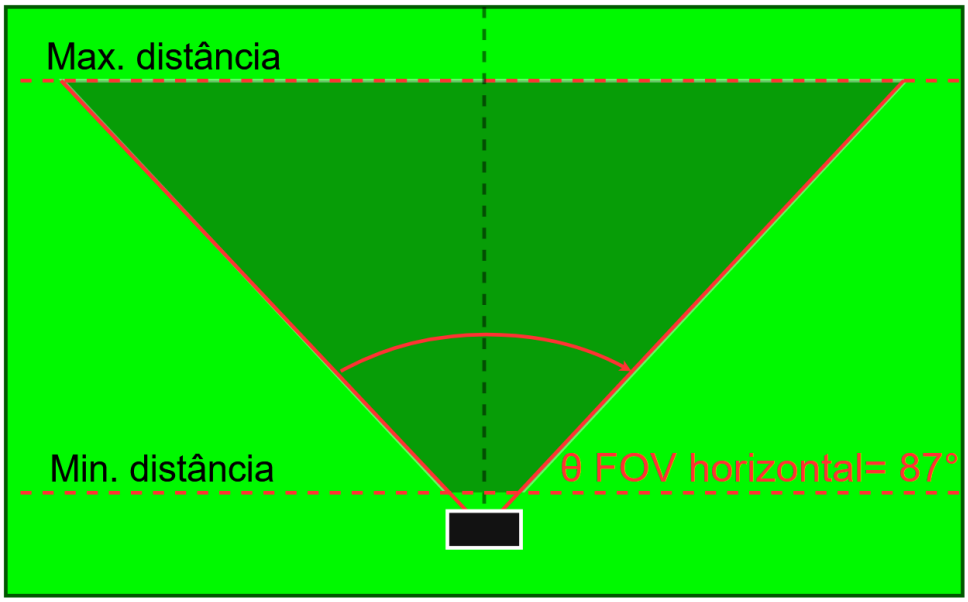
Figura 5 - Esboço do alcance do campo de visão do robô



Fonte: do autor

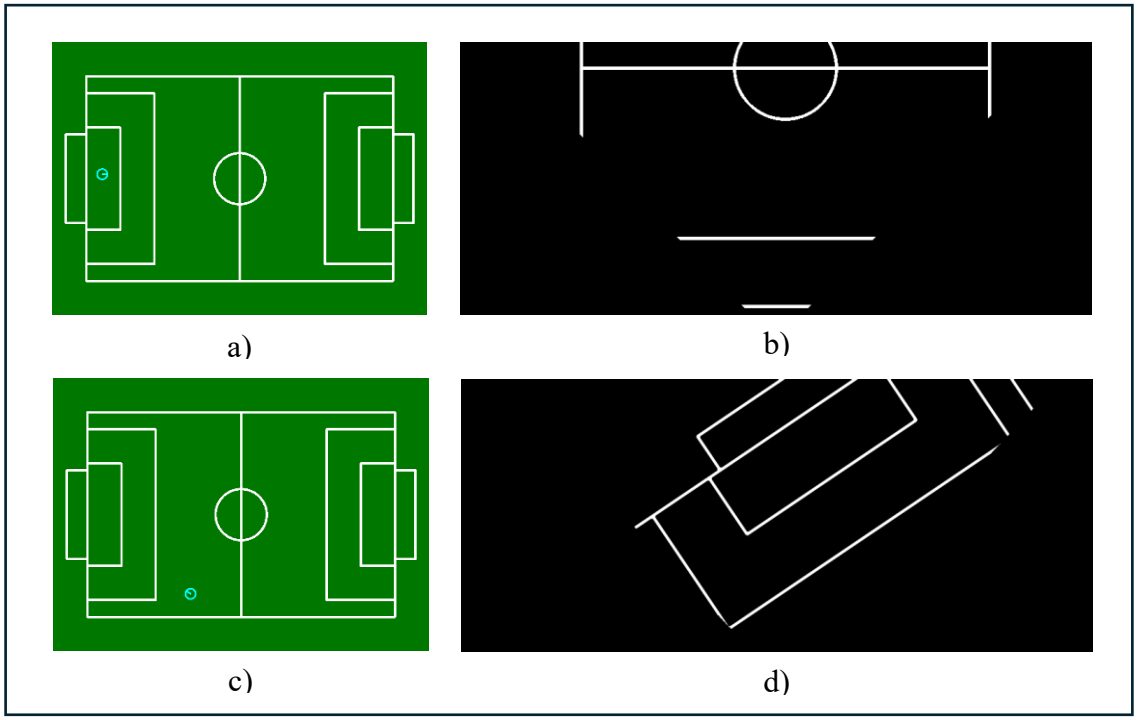
A partir dessas distâncias, e considerando a localização do robô no campo, é calculado o trapézio que representa, nesse trabalho, a projeção do campo de visão do robô, esquematizado na Figura 6 e exemplificado na Figura 7. Visando performance, a exibição do POV do robô na simulação é convertida para uma imagem binária (preta e branca).

Figura 6 - Esboço da região que compõe o campo de visão do robô



Fonte: do autor

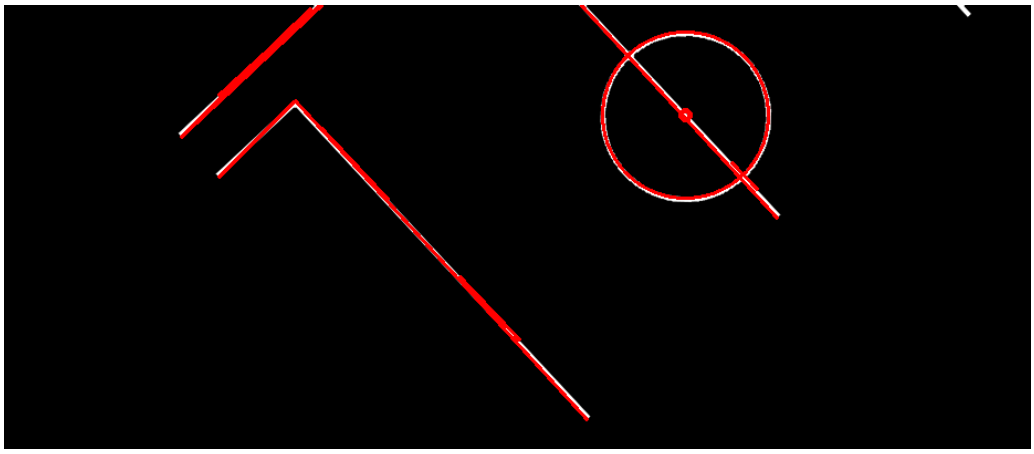
Figura 7 - Exemplos do recorte do POV do robô em b) e d) referentes às posições a) e c), respectivamente, no campo



Fonte: do autor

Utilizando a imagem que representa o campo de visão do robô e as funções *HoughLinesP* e *HoughCircles* da biblioteca OpenCV, juntamente com segmentação de imagem, obtém-se as linhas e, caso visível, o círculo central do campo. Essa etapa simula o robô real inferindo sobre as informações captadas por sua câmera. A Figura 8 exemplifica tal inferência.

Figura 8 - Linhas e círculo identificados no POV destacados (em vermelho)



Fonte: do autor

Com as informações das linhas, calcula-se as distâncias radiais do robô para a linha do campo mais próxima identificada. Os ângulos das $N_{sensores}$ direções para medição de distância são calculados pela equação (9), em que $\theta_{particula}$ é o ângulo da direção da partícula e $fov_{horizontal}$ o ângulo do FOV horizontal da câmera. Nota-se no caso da medição do robô, como sua perspectiva possui sempre a mesma orientação, $\theta_{particula} = \theta_{POV\ robo} = -90°$, obedecendo a orientação do OpenCV.

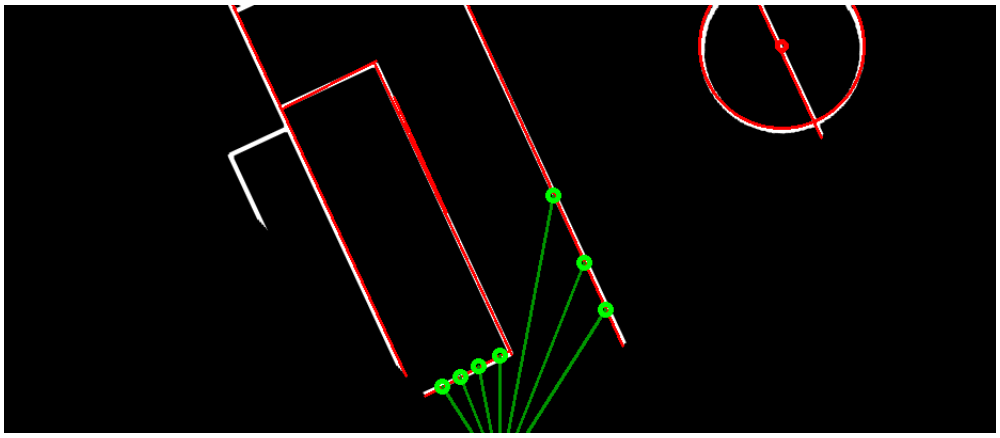
$$angulo_{sensor\ j} = \theta_{particula} - \frac{fov_{horizontal}}{2} + j * \frac{fov_{horizontal}}{N_{sensores} + 1} \quad (9)$$

$$em\ que\ j = \{1, \cdots, N_{sensores}\}$$

Devido ao método escolhida para o cálculo do peso das partículas, que será discutido na seção seguinte, a quantidade de medições impacta quase linearmente no tempo de execução do código, fazendo com que uma alta quantidade de leituras seja inviável. No entanto, observou-se que utilizar poucas direções resulta em uma conversão lenta e pouco reativa, reduzindo a exatidão das amostras. Portanto, neste trabalho foram utilizadas sete medições de distância visando equilíbrio entre performance e resultados. A Figura 9 exemplifica tais medições em

determinado momento. Quando não é encontrada uma linha em certa direção, um valor arbitrariamente alto será atribuído à medição, evitando indeterminações.

Figura 9 - Direções das medidas de distâncias destacadas (em verde)

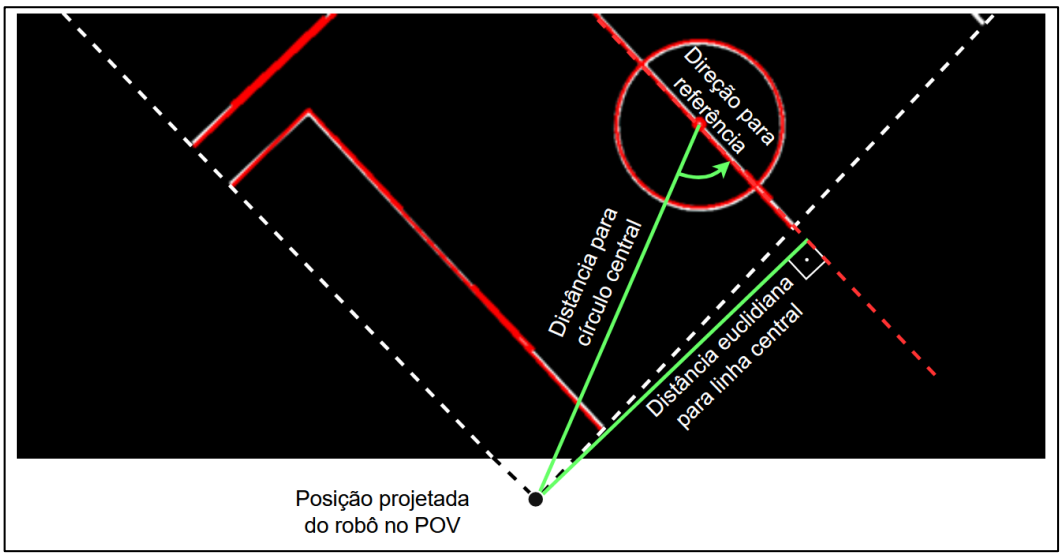


Fonte: do autor

Nesta etapa também são obtidas as informações dos pontos de referência do campo. O círculo central do campo e a linha que o cruza estão sendo considerados neste documento como pontos de referência. É possível estipular outros pontos de referência em um campo de futebol, porém para simplificação do problema e para avaliar situações em que não existem referências no POV do robô, foram escolhidos esses. Tal escolha se deu pela facilidade de identificação (tanto pela simplicidade da forma quanto pela disposição central no campo). Ressalta-se que a linha central somente é reconhecida como um ponto de referência quando identificada junto ao círculo central.

Quando o círculo é identificado calcula-se a distância até ele, a distância euclidiana para a linha central e a direção do ponto de referência, isto é, o ângulo do vetor que parte do robô e aponta para o ponto central do campo. A Figura 10 esboça tais medidas no POV representado na Figura 8. Essas informações serão utilizadas na etapa de *Sensor Resetting*.

Figura 10 - Representação das leituras relevantes dos pontos de referência



Fonte: do autor

4.2.2 Avaliação das partículas

A partir das linhas geradas na etapa de construção do campo, é possível extrair as informações sensoriais simuladas das partículas de maneira mais eficiente, sem depender de entender seu POV da mesma forma que é feita com o robô. Utilizando novamente a equação (9), porém agora aplicada a cada partícula, encontra-se as direções para as medições de distância comparando com as linhas pré-estabelecidas, economizando o processamento de imagem que seria necessário pela abordagem utilizada com o robô.

Após a medição dos sensores de uma determinada partícula, as distâncias encontradas são comparadas com as do robô de acordo com a equação (10), em que $medicao_j$ é a leitura de distância de índice j da partícula e $medicao_{j_{robo}}$ a leitura de distância de índice j do robô, resultando em uma pontuação (*score*) entre 0 e 1, com 0 representando uma medida de distância exatamente igual à do robô e 1 uma extremamente diferente. A partir dos 7 *scores* resultantes das 7 medições de distância utilizados, é calculado o peso w não normalizado da partícula com a equação (11), resultando também em um valor entre 0 e 1, porém com 1 agora representando uma partícula com métricas exatamente iguais às do robô.

$$score_j = \frac{\left|medicao_{j_{robo}} - medicao_j\right|}{medicao_{j_{robo}} + medicao_j}, em\ que\ j = \{1, \cdots, N_{sensores}\} \tag{10}$$

$$w = \prod_{j=1}^{N_{sensores}} \left(1 - score_j\right) \tag{11}$$

A média dos pesos não normalizados é calculada, para futura utilização na etapa de *uniformity validation*, e os mesmos são normalizados de modo que $\sum w_i = 1$ para os cálculos de *ESS*, entropia e posterior reamostragem.

4.2.3 *Sensor Resetting*

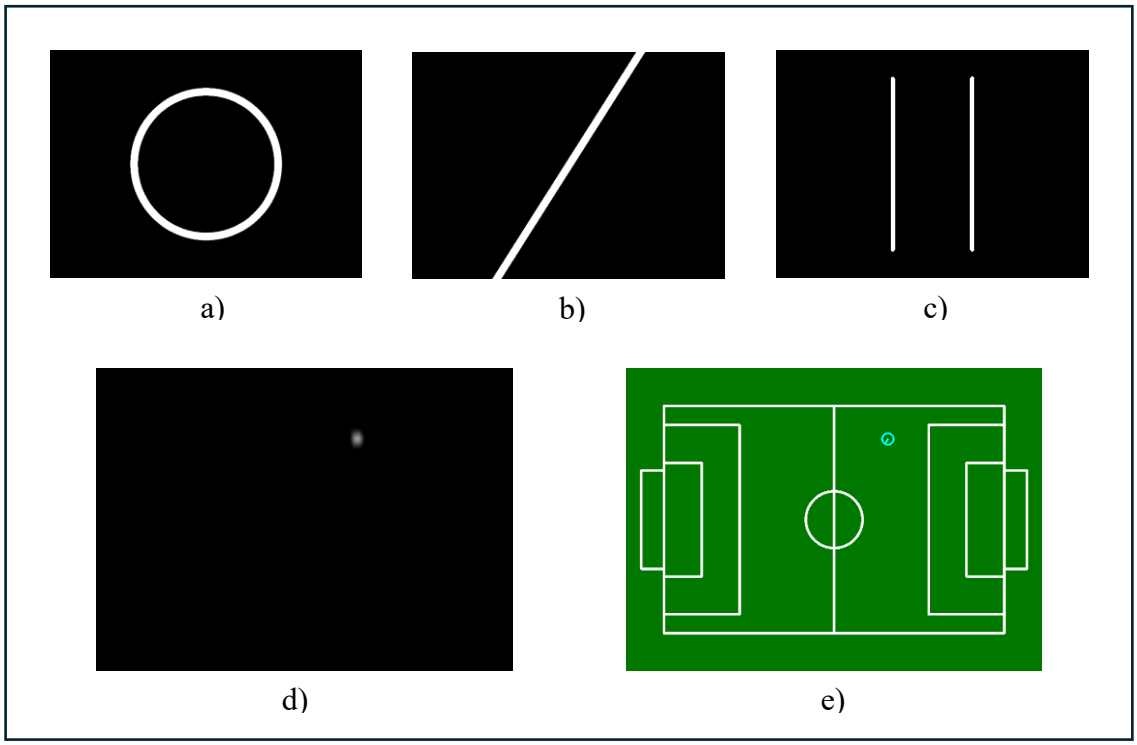
Com as informações dos pontos de referência, são inferidos locais com maior probabilidade de o robô estar. Tal processo consiste em combinar a distância até o centro do campo (ou epicentro do círculo central), a direção do vetor que parte do robô até o centro do campo e a distância para a linha central. O erro de leitura de distância da câmera simulada é, de acordo com as informações do fabricante, de 2% à 2 metros. Como existe na simulação um erro ocasionado pela detecção de linhas causado pela espessura das linhas do campo (5 cm ou 5 pixels), assume-se um erro conservador de 10% nos dados utilizados para *Sensor Resetting*.

Efetivamente, tal processo é realizado com operações de segmentação com Opencv.

- Cria-se uma máscara de um círculo centrado no epicentro do círculo central do campo com raio igual à distância para o mesmo;
- Cria-se uma máscara com uma reta que passa pelo centro do campo e direção equivalente à calculada para cruzar a localização do robô;
- Cria-se uma máscara com dois seguimentos de reta equidistantes da linha central, de acordo com a distância medida da mesma;
- Todas as máscaras possuem espessura proporcional ao erro calculado;
- As máscaras são combinadas e aplica-se um filtro de gaussiana para suavizar o resultado e obter-se as regiões de maior probabilidade.

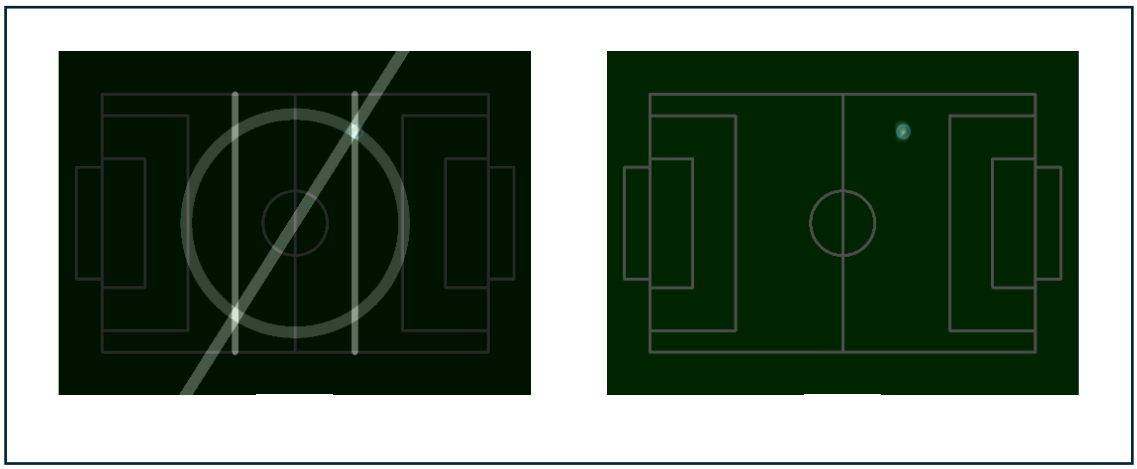
A fim de evitar ambiguidades, inicialmente e na etapa de estimativa da posição do robô, é determinada a tendência de sua posição no campo. A tendência indica se o robô foi localizado pela última vez (ou inicializado) nos quadrantes superiores, inferiores, à esquerda ou à direita, ou em uma combinação de quadrantes. Como padrão, foi definido que inicialmente o robô será inicializado no quadrante direito (sem tendência vertical definida). A Figura 11 exemplifica uma situação em que as informações dos pontos de referência geram as máscaras e o resultado da segmentação, enquanto a Figura 12 expõe a mesma situação, porém com as máscaras e o resultado sobrepostos ao campo com o robô.

Figura 11 - Em a), b) e c) as máscaras parciais, em d) sua junção e em e) a localização real do robô



Fonte: do autor

Figura 12 - As máscaras parciais da Figura 11 e a região de maior probabilidade sobrepostas ao campo, respectivamente à esquerda e à direita



Fonte: do autor

Com o resultado da segmentação e a função *findNonZero* do OpenCV é possível montar os conjuntos $Particulas_{SR}$, que consiste na localização dos pixels não nulos da máscara resultante (isto é, dos locais com alguma probabilidade de representar a localização real do

robô), e W_{SR}, que consiste nos pesos de tais partículas (equivalentes aos valores dos pixels não nulos, após normalização).

Havendo então pontos de referência no campo de visão do robô, passa-se a ter dois conjuntos de partículas para realização da reamostragem: o conjunto original que poderá ser utilizado para *importance resampling* e o conjunto de novas partículas baseadas em *sensor resetting*.

4.2.4 Reamostragem

Para a verificação de *over-confidence* é primeiro comparado o *ESS* com um limite, depois é feita a análise das entropias de acordo com a equação (7). O limite adotado para a comparação da quantidade efetiva de partículas é igual a 50% da quantidade de partículas adotadas. Quando o *ESS* for insuficiente, a quantidade de partículas a sofrerem *sensor resetting* é calculada pela equação (6). O valor de λ adotado foi de 0.4 tanto para validação com a equação (7) quanto para cálculo da quantidade de novas partículas com a equação (8). Já a etapa de *uniformity validation*, quando não é confirmada a *over-convergence*, resulta em *sensor resetting* ocorrendo em todo o conjunto de partículas quando o somatório dos pesos não normalizado foi inferior a 25% da quantidade de partículas. Quando superior, não ocorre *sensor resetting*. O limite para *ESS*, o valor de λ e o valor mínimo para o somatório de pesos foram obtidos através de experimentação e avaliação qualitativa dos resultados.

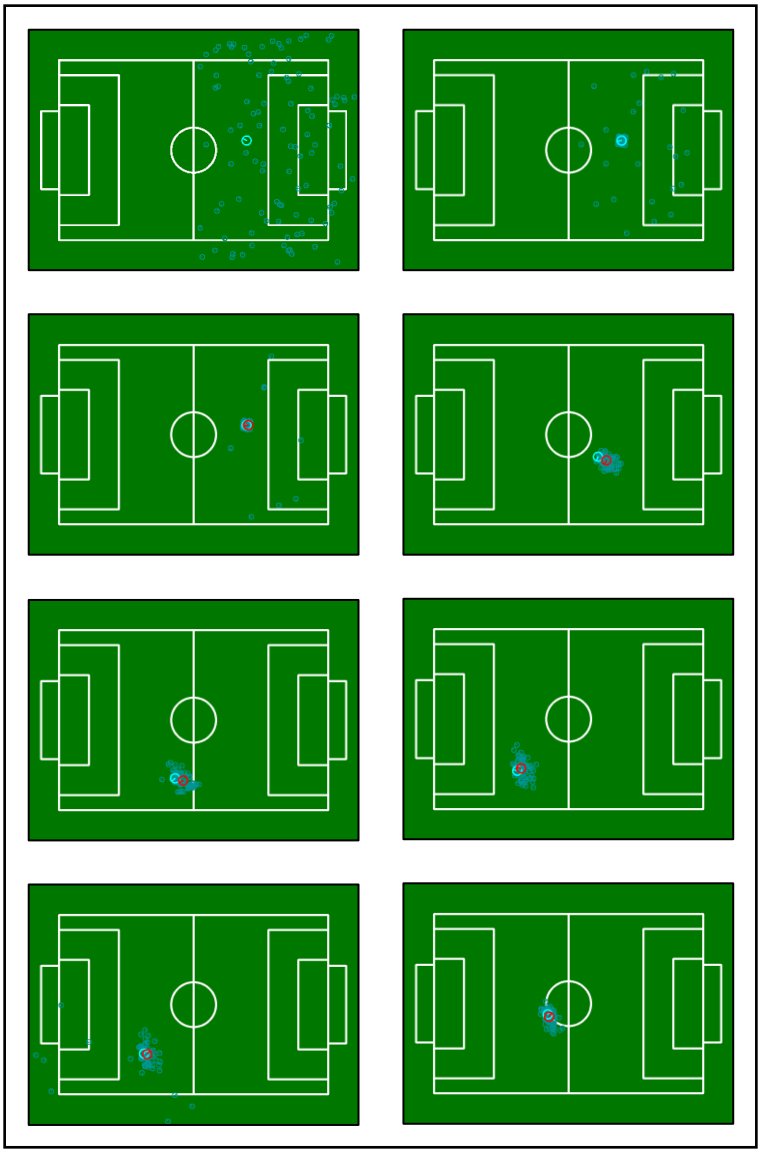
Quando *sensor resetting* não é possível, isto é, quando nenhum ponto de referência se encontra no campo de visão do robô, as mesmas verificações e cálculos são realizados resultando em uma quantidade de novas partículas a ser adicionadas. O algoritmo implementado, nessas situações, irá gerar novas partículas a partir de uma distribuição uniforme (obedecendo a tendência padrão ou estimada), porém com apenas 10% da quantidade de novas partículas calculadas. Assim, são adicionadas novas amostras ao conjunto, combatendo problemas de convergências ruins mesmo quando não é possível realizar *sensor resetting*, porém sem adicionar muita entropia ao sistema (como consequência da adição de muitas partículas aleatórias).

Ao restante das partículas do conjunto inicial, é realizado *importance resampling* com um algoritmo de *systematic resample* conforme descrito na seção 3.2.

5 Resultados

Ao se empregar todos os métodos descritos na última seção, foi possível obter um sistema de localização consistente. A Figura 13 exemplifica a execução do programa utilizando 100 partículas para melhor visualização, inicializando o robô (e, consequentemente, as partículas) no quadrante direito do campo. Ao longo das iterações (sequencialmente da esquerda para direita, de cima para baixo) é possível notar as partículas (em azul) convergindo para o local do robô (em ciano) e a estimativa da sua localização sendo gerada (em vermelho).

Figura 13 - Iterações do filtro de partículas



Fonte: do autor

A estimativa da localização do robô $\overline{pos}$ é calculada a partir da equação (12), utilizando a média das localizações das partículas ponderada por seus pesos. De maneira similar, o desvio σ das partículas é a raiz quadrada da variância σ^2 obtida com a equação (13). Como nesta etapa os pesos estão normalizados de modo que $\sum w_i = 1$, as equações podem ser simplificadas.

$$\overline{pos} = \frac{\sum_{i=1}^{N_{particulas}} (particula_i * w_i)}{\sum_{i=1}^{N_{particulas}} w_i} = \sum_{i=1}^{N_{particulas}} (particula_i * w_i) \tag{12}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N_{particulas}} \left((particula_i - \overline{pos})^2 * w_i\right)}{\sum_{i=1}^{N_{particulas}} w_i} = \sum_{i=1}^{N_{particulas}} \left((particula_i - \overline{pos})^2 * w_i\right) \tag{13}$$

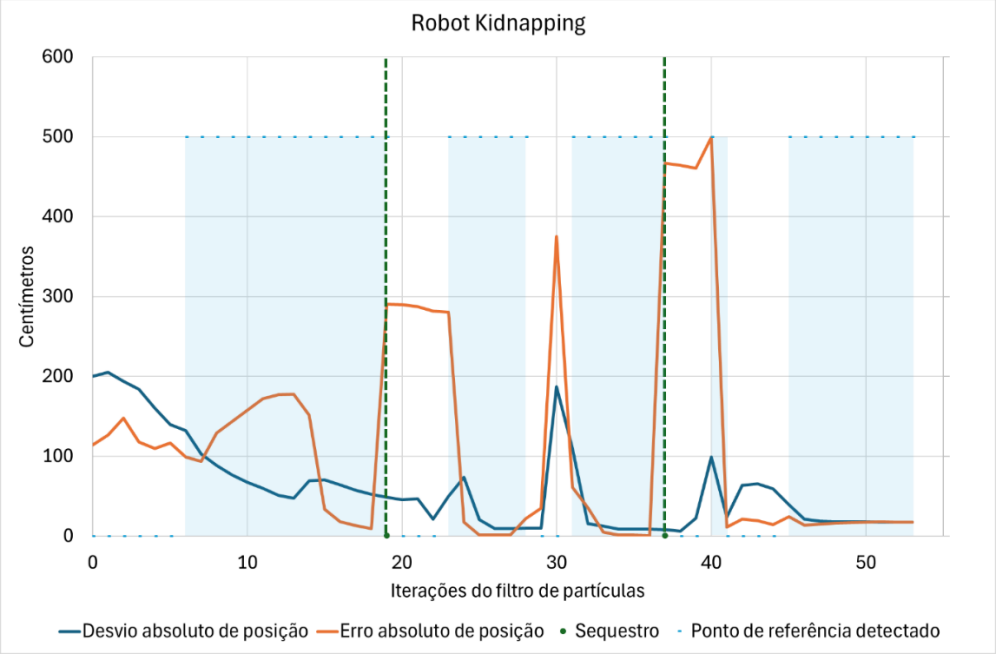
A fim de gerar resultados qualitativos, foi calculado o desvio absoluto de posição com a equação (14) e o erro absoluto de posição com a equação (15). Para esta etapa, está sendo desconsiderado o ângulo da direção. O desvio absoluto indica o quanto o sistema de partículas convergiu, enquanto o erro absoluto o quão longe da posição real do robô tal convergência se encontra. Deseja-se sempre minimizar as métricas propostas.

$$\sigma_{abs} = \sqrt{\sigma_x^2 + \sigma_y^2} \tag{14}$$

$$erro_{abs} = \sqrt{(pos_x - \overline{pos}_x)^2 + (pos_y - \overline{pos}_y)^2} \tag{15}$$

A Figura 14 apresenta um gráfico da performance do filtro de partículas utilizando 50 partículas quando submetido a duas situações de sequestro de robô. Nota-se os picos de erro absoluto resultantes do movimento não modelado súbito. Em azul claro estão destacados os momentos em que o robô era capaz de identificar os pontos de referência e, consequentemente, realizar *sensor resetting*, se necessário.

Figura 14 - Erro e desvio absolutos da posição após duas situações de sequestro



Fonte: do autor

Fica evidente, no teste analisado, a dependência dos pontos de referência para um bom desempenho do sistema. Nos momentos em que não se detecta as referências observa-se pouca melhoria do erro ou desvio, sendo possível observar iterações onde o as métricas pioraram.

No link a seguir está disponível um vídeo demonstrando o funcionamento do projeto apresentado: demonstração.

6 Conclusões e sugestão de trabalhos futuros

Conforme observado nas métricas quantitativas, o resultado do filtro de partículas implementado depende fortemente de possuir pontos de referência disponíveis para *sensor resetting*. Não havendo, a convergência se torna incerta e pouco confiável. Isso ocorre devido às possibilidades de reamostragem quando não se tem a referência: reaplicar *importance resampling* correndo o risco de concentrar as novas partículas em locais indevidos (*over-convergence*), ou introduzir novas partículas aleatórias, fazendo com que todas as métricas piorem.

Analisando a performance do algoritmo, com 50 partículas foi possível obter os resultados da Figura 12 e um tempo médio entre iterações de ~1 segundo. O sistema mostrou-se responsivo e apresentou rápida convergência para local adequado quando identifica pontos de referência. Uma vez localizado, o rastreio do robô era eficaz. Um tempo de execução de 1 segundo não apresenta impacto quando projetado para a situação real quando se toma como base a competição da RoboCup, visto que os robôs são lentos. Entretanto, o hardware utilizado para execução é extremamente superior ao disponível na competição, além da situação de competição apresentar diversos outros algoritmos executando simultaneamente. Considerando isso, para aplicar a solução em prática faz-se necessário uma melhoria de performance.

Para melhorias futuras, as etapas necessárias para a plena localização de um robô humanoide em cenário de competição são duas: implementação em três dimensões e melhorias do código do filtro de partículas, visando solucionar os problemas citados.

Com relação à migração para o espaço tridimensional, sugere-se abstrair informações para o bidimensional, de forma a simplificar a simulação das partículas e poupar performance computacional. As leituras realizadas pelo robô simulado neste trabalho foram a identificação (e localização relativa ao robô) dos pontos de referência, medições de distância e medidas de deslocamento. A fim de continuar utilizando uma simulação de partículas em duas dimensões, propõe-se a extração desses dados do robô real ou simulado tridimensionalmente para utilização no algoritmo.

Visando melhorias no código, a dependência dos pontos de referência só é um problema quando combinado com a escassez desses. Apesar da posição central das referências escolhidas, inúmeras posições do campo não as enxergam, resultando em situações difíceis para o filtro.

Com isso, acredita-se fortemente que a modelagem de mais pontos de referência conhecidos, como os gols, interseções específicas do campo ou marcas de pênalti, aumentarão a eficácia do filtro.

Python foi escolhido para desenvolvimento do código devido à baixa complexidade, métrica desejada conforme previamente citado. Porém, por se tratar de uma linguagem de alto nível, tal escolha impactou no tempo de execução do código. Sugere-se a utilização de uma linguagem mais eficaz na manipulação de códigos e matrizes extensas, como C++ ou MATLAB para atingir melhores resultados.

Apesar de diversos pontos de melhoria o principal objetivo de validar a abordagem escolhida e apresentar uma base para evolução foi atingido.

Referências

A. C. Almeida, S. R. J. Neto e R. A. C. Bianchi, **Comparing Vision-Based Monte-Carlo Localization Methods**, *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, João Pessoa, Brazil, 2018, pp. 437-442, doi: 10.1109/LARS/SBR/WRE.2018.00084.

Čapek, Karel (2001). **R.U.R.**. Traduzido por Paul Selver e Nigel Playfair. Dover Publications.

ROBOCUP. **Scope**. [*S. n.*], 2025. Disponível em: https://www.ieee-ras.org/robocup. Acesso em: 4 maio 2025.

Doucet, A. (1998). **On sequential simulation-based methods for Bayesian filtering, Cambridge University**, Department of Engineering, Cambridge, UK, Technical Report. CUED/FINFENG/ TR 310

Fox, Dieter & Burgard, Wolfram & Dellaert, Frank & Thrun, Sebastian. (1999). **Monte Carlo Localization: Efficient Position Estimation for Mobile Robots**. Proceedings of the National Conference on Artificial Intelligence. 343-349.

Gutmann, Steffen & Fox, Dieter. (2002). **An Experimental Comparison of Localization Methods Continued**. 10.1109/IRDS.2002.1041432.

HICKS, R. D., 1972. **Lives of Eminent Philosophers: Chapter 2. DIOGENES (404-323 B.C.)**. Disponível em: http://data.perseus.org/citations/urn:cts:greekLit:tlg0004.tlg001.perseus-eng1:6.2. Acesso em: 4 maio 2025.

Hol, Jeroen & Schön, Thomas & Gustafsson, Fredrik. (2006). **On Resampling Algorithms for Particle Filters**. IEEE Nonlinear Statistical Signal Processing Workshop. 79 - 82. 10.1109/NSSPW.2006.4378824.

INTEL REALSENSE. **Intel® RealSense™ Depth Camera D435**. [*S. n.*], 2025. Disponível em: https://www.intelrealsense.com/depth-camera-d435/. Acesso em: 4 maio 2025.

INTERNATIONAL FEDERATION OF ROBOTICS. **Standardization: Robot definitions at ISO**. [S. l.], 2025. Disponível em: https://ifr.org/standardisation. Acesso em: 4 maio 2025.

Lenser, S. & Veloso, M. (2000). **Sensor resetting localization for poorly modeled mobile robots**, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1225-1232.

Liu, J.; Chen, R. & Logvinenko, T. (2001). **A theoretical framework for sequential importance sampling and resampling**, In Sequential Monte Carlo in Practice, Springer-Verlag, New York.

OPENCV. **Image Processing (imgproc module)**. [*S. n.*], 2025. Disponível em: https://docs.opencv.org/3.4/d7/da8/tutorial_table_of_content_imgproc.html. Acesso em: 4 maio 2025.

ROBOCUP. **RoboCup Soccer Humanoid League Laws of the Game 2025**. [*S. l.*], 2025. Disponível em: https://humanoid.robocup.org/wp-content/uploads/RC-HL-2025-Rules.pdf. Acesso em: 4 maio 2025.

Rubin, D. 1988. **Using the SIR algorithm to simulate posterior distributions**. Bayesian Statistics 3. Oxford University Press.

Tanner, M. 1993. **Tools for Statistical Inference**. Springer.

Wen Shang e Dong Sun (2007). **A Visual Based Extended Monte Carlo Localization for Autonomous Mobile Robots**, Vision Systems: Applications, Goro Obinata and Ashish Dutta (Ed.), ISBN: 978-3-902613-01-1.

Apêndices

APÊNDICE A – Código Desenvolvido

O código desenvolvido está disponível em:

https://github.com/pedrohrpc/2d_monte_carlo_localization