CENTRO UNIVERSITÁRIO FEI

Laboratório 2: Robô caça bloco

CC7711 – Inteligência Artificial e Robótica

Vitor Acosta da Rosa
RA: 22.218.006-9

São Bernardo do Campo
2021

1 Descrição do algoritmo

Com o objetivo de tocar em todas as caixas pelo menos uma vez e identificar as caixas leves (com menor massa) e consequentemente movíveis, foi idealizado um algoritmo que auto-ajustasse a orientação do robô de acordo com a caixa mais próxima que ainda não foi tocada, e, a partir das premissas adotadas (seção 1.1 Considerações Iniciais), o robô seria capaz de tomar decisões sobre sua movimentação, tocar nas caixas e avaliar se são leves ou não, indicando através de um LED.

1.1 Considerações Iniciais

As considerações feitas para a construção do algoritmo foram:

I **A posição inicial de toda caixa é conhecida previamente:** Dessa forma, o algoritmo pode encontrar a caixa mais próxima através das coordenadas da caixa.

II **A posição das caixas é monitorada:** Assim, é possível avaliar a diferença (caso ocorra deslocamento) e categorizar a caixa como caixa leve.

III **A posição do robô é conhecida e atualizada:** A todo instante é conhecida a coordenada do robô em campo.

IV **A velocidade rotacional e tangencial do robô é conhecida:**

Essa velocidade indica o quanto o robô pode girar no lugar, assim, é possível rotacionar a frente do robô para a caixa correta. Esse valor é de $278.237 grau/seg$, calculado pela fórmula

$$V_{rotacional} = \frac{360 \times V_{tangencial}}{(\pi \times L_{Eixo})}$$

Sendo a $V_{tangencial} = V_{rotacional} \times R_{roda}$.

Com $R_{roda} = 0.0205m$ e $V_{rotacional} = 6.28 rad/s$.

Além disso, L_{eixo} é o comprimento do eixo do E-Puck, que é $0.052m$.

(Informações retiradas do site: https://cyberbotics.com/doc/guide/epuck).

V **Não existem obstáculos além das próprias caixas:** Não existem obstáculos no cenário que impossibilitem que o robô alcance a caixa alvo, como por exemplo uma caixa cercada por paredes.

1.2 Módulos extras

Para o correto comportamento do robô na tomada de decisão e deslocamento, foram alocados dois módulos extras ao E-Puck: o GPS e a bússola (*compass*). O objetivo dessa colocação foi corrigir imprecisões geradas pelo *supervisor* do nó do robô, sendo possível adquirir corretamente a direção norte do E-Puck e

seu posicionamento em campo.

Ambos módulos foram inseridos através do menu do E-Puck na seção "Turret Slot".

2 O Algoritmo e sua lógica

A ideia base para permitir que o robô se desloque pelo campo e vá em direção à todas as caixas possui três pilares:

I Conversão de coordenadas do webots para coordenadas cartesianas,

II Cálculo de caixa mais próxima, e

III Cálculo do ângulo à caixa mais próxima.

2.1 Conversão de coordenadas

Toda a estrutura de coordenadas do webots baseia-se em coordenadas dadas nos eixos X, Y e Z, como o robô não sofre alterações de altura (eixo Y), considera-se para esse problema somente os eixos X e Z.

O primeiro passo do algoritmo, é converter as coordenadas dadas em X e Z para coordenadas equivalentes cartesianas X e Y, a fim de facilitar o cálculo de ângulos de rotação para a caixa mais próxima (ideia abordada na seção 2.3. Cálculo do ângulo à caixa mais próxima).

A coordenada X do webots é a mesma que a coordenada X cartesiana: é horizontal e cresce da esquerda para a direita.

Já a coordenada Z do webots, para convertê-la ao equivalente Y cartesiano, basta negativar o valor. Isso porque o eixo Z é perpendicular ao eixo X e cresce de cima para baixo, enquanto o Y cartesiano cresce de baixo para cima.

Portanto, todas coordenadas (X e Z) adquiridas das caixas e do robô são convertidas sempre que for necessário calcular a caixa mais próxima e a rotação do robô.

2.1.1 Bússola e GPS

O GPS já fornece por padrão em seu método *wb_gps_get_values* um vetor de três valores que é compatível com a função de conversão abordada na seção anterior.

Já a bússola (*compass*) deve passar por um tratamento específico para a correta obtenção do norte relativo do robô, uma vez que o norte por padrão é apontando o eixo Y positivo. O tratamento necessário foi retirado do site https://cyberbotics.com/doc/reference/compass conforme a documentação.

2.2 Cálculo de caixa mais próxima

A partir do momento em que é conhecida a coordenada (X,Y) de todas as caixas dispostas, e também que o GPS rastreia a coordenada (X,Y) do robô a todo instante, o cálculo de caixa mais próxima baseia-se na simples distância entre dois pontos, dada por:

$$d = \sqrt{(caixa_x - robo_x)^2 + (caixa_y - robo_y)^2}$$

A busca de cálculo de caixa mais próxima acontece sempre que a variável de controle indica que o robô não possui uma caixa alvo. A busca também envolve percorrer todas as caixas e analisar individualmente. Ou seja, como existem nove caixas, é realizado um loop entre as coordenadas (X,Y) da $caixa_i$, $0 \leq i < 9$.

Vale ressaltar que é empregado um vetor que controla a visita às caixas, logo, toda busca de caixa mais próxima leva em consideração se a caixa mais próxima foi visitada (tocada) anteriormente.

2.3 Cálculo do ângulo à caixa mais próxima

Após selecionada a caixa de menor distância, é possível formar um triângulo para descobrir o ângulo que leva a frente do robô em direção a caixa correta. A figura 1 demonstra o esquema proposto.

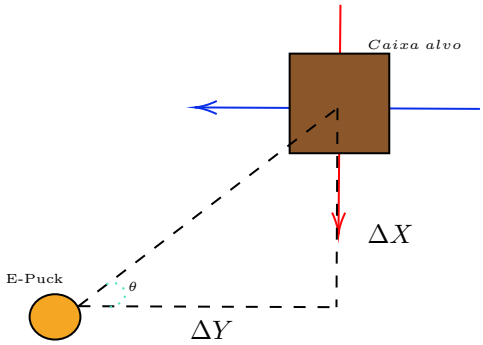

Figura 1: Triângulo formado pela caixa e robô

A partir dessa ideia, é possível calcular o ângulo θ através do arco tangente de $\frac{\Delta Y}{\Delta X}$.
Esse ângulo é calculado sempre que o robô necessita de uma nova caixa alvo.

É importante ressaltar que como as coordenadas do robô e das caixas estão em suas coordenadas cartesianas equivalentes, o direcionamento do robô deve ser convertido também, de forma que o ângulo θ seja calculado de maneira satisfatória. Nessa perspectiva, a sequência de cálculos é a seguinte:

I **Converter o direcionamento do robô para cartesiano:** Como no ambiente webots a direção que possui o ângulo 0 é no sentido do eixo Z negativo (ou de maneira equivalente o Y positivo em cartesiano), é necessário somar o direcionamento obtido com o valor 90, evidenciando que o ângulo 0 está deslocado (na visão cartesiana) em 90 graus.

II **Calcular o ângulo θ entre o robô e o destino:** utilizando a função de arco tangente, o ângulo theta entre o robô e a caixa mais próxima é descoberto.

III **Corrigir o direcionamento do robô:** Baseado no direcionamento convertido em coordenadas cartesianas (item I) e tendo o ângulo à caixa alvo (item II) é calculado o direcionamento (norte) do robô e retornado qual o ângulo o robô deve rodar a fim de alinhar-se a caixa alvo.

2.3.1 Procurando uma nova caixa

Por padrão, a simulação inicia já requirindo uma caixa alvo, através de uma variável de controle chamada *target*, quando essa variável está com o valor -1 quer dizer que o robô está apto para receber um novo alvo, qualquer valor ≥ 0 já é considerado que o robô possui uma caixa alvo.

É importante ressaltar que essa variável indica a caixa que é a caixa alvo, ou seja, através do valor nela armazenado é possível acessar a matriz que guarda as coordenadas de todas as caixas do cenário e encontrar a respectiva caixa.

2.3.2 Caixa válida ou inválida?

Em complemento a matriz de coordenadas (X,Y) das caixas, também é gerenciado um vetor que armazena também dois tipos de valor (zero e um) chamado *visited_boxes*.

Quando zero, quer dizer que o robô não visitou aquela caixa, e, portanto, ela pode ser considerada ao buscar uma nova caixa alvo. Quando o índice tem valor um, o robô não pode mais considerar tal caixa como alvo pois já visitou ela uma vez e deve procurar uma nova, mesmo que a distância entre a nova caixa seja maior.

Os índices são respectivos a caixa alvo que é selecionada, ou seja, caso a caixa alvo seja a 3, o vetor *visited_boxes* recebe o valor 1 no índice 3 quando o robô encostar na caixa.

2.3.3 Colisão e nova candidata: Comportamento do robô

O robô, após executados todos os passos citados nas sessões anteriores fará a escolha e o movimento, na seguinte ordem:

I Calcula a caixa mais próxima, levando em consideração o *status* (visitado ou não) daquela caixa.

II Calcula o ângulo que o robô deve girar (no lugar) para alinhar com a caixa.

III Roda o robô até que o ângulo seja pelo menos 1 grau menor/maior que o ângulo alvo.

Note que a consideração de 1 grau menor ou maior quando feita a comparação entre ângulo do robô e ângulo alvo é realizada por conta do ruído que pode existir.

IV Após rodar suficientemente até alinhar a frente do robô à caixa, o robô anda para frente.

V Quando for detectada a colisão (esquerda, direita ou frente, através dos sensores de proximidade ps6, ps7; ps0, ps1; e ps7, ps0 respectivamente) E a distância entre o robô e o centro da caixa for menor que o limite pré-definido 0.35m, o robô realiza um giro e se afasta da caixa.

VI A variável *target* é definida como -1, para que seja procurada uma nova caixa alvo.

VII O algoritmo retorna ao item I.

3 Simulação

O tempo decorrido desde o começo da simulação até o ponto em que todas as caixas foram tocadas pelo menos uma vez foi de 1 minuto, 06 segundos.
A simulação pode ser vista em: https://youtu.be/nD6PT2wQ0FY
E o código fonte pode ser encontrado em: https://github.com/VitorAcosta/Robotics