TP1 Planejamento de Movimento de Robôs: Curve Following, Attractive/Repulsive Potential Fields, Tangent Bug e Wave-Front

Eudes e João Paulo

***Abstract*— The goal of this work is to apply trajectory planning and navigation algorithms in simulations of mobile robots. The implementation of the algorithms and its results will be discussed. The algorithms contemplated in this work are: Curve Following, Attractive/Repulsive Potential Fields, Tangent Bug and Wave-Front.**

I. INTRODUÇÃO

O planejamento de movimento de robôs tem o objetivo de, dada uma tarefa de alto nível, i.e. "mova-se para a coordenada (5m,5m)", o robô deve ser capaz de compilar as especificações em um conjunto de ações de baixo nível para atendê-la. Basicamente, é preciso que um caminho seja determinado de forma a priori ou online para que colisões sejam evitadas e a tarefa desejada seja realizada.

Neste trabalho, a computação de caminhos e navegação realizados pelos algoritmos de Curve Following, Attractive/Repulsive Potential Fields, Tangent Bug e Wave-Front são implementados com auxílio do pacote ROS e avaliados em simulação no STAGE.

Este artigo encontra-se dividido da seguinte forma: na seção II, a configuração do robô é explicada; a seção III tratará do algoritmo utilizado para fazer com que o robô siga uma curva desejada; na seção IV, a estratégia de utilizar campos atrativos e repulsivos para movimentar o robô é explicada e demonstrada; a seção V tratará do algoritmo de Tangent Bug; a seção VII tratará do algoritmo de Wave-Front; e, por fim, a seção VIII fechará com uma breve conclusão sobre o trabalho desenvolvido.

O projeto com os códigos das implementações realizadas pode ser encontrado no github, através do link: TP1_github

II. CONFIGURAÇÃO DO ROBÔ

O robô utilizado para simulação no STAGE dos algoritmos implementados possui a seguinte configuração no de .world:

```
define erratic position
(
    size [ 1.000 1.000 0.250 ]
    origin [ -0.050 0.000 0.000 0.000 ]
    drive "diff"
    topurg
    (
        pose [ 0.050 0.000 0.000 0.000 ]
    )
)
```

Em que, o componente topurg é um sensor definido por:

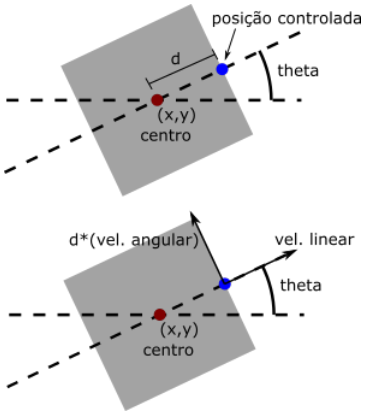


Fig. 1. Posição controlada

```
define topurg ranger
(
    sensor(
        range [ 0.000 10.000 ]
        fov 360
        samples 2000
    )

    color "black"
    size [ 0.050 0.050 0.100 ]
)
```

Ou seja, é definido um robô movel quadrado de tamanho $1m$ com atuação diferencial e que possui um sensor laser para medição de distância com range $0 - 10m$ e cobertura de 360 graus com 2000 amostras. Algumas variações são utiilzadas desses valores em alguns algoritmos, mas, em geral, a configuração é a mesma em todos.

A. Controle de posição

Todos algoritmos apresentados neste trabalho tem o objetivo de controlar a posição do robô. Ou seja, o controle de orientação não é incluída.

Para controlar a posição o robô, uma simplificação interessante é considerar um ponto específico deste para ser controlado. Neste trabalho foi escolhido o ponto central da frente do robô coforme mostrado na Figura 1.

A posição (x, y) do centro do robô é mensurada por um odômetro. Portanto a posição (x_f, y_f) a ser controlada é obtida por:

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} x + d\cos\theta \\ y + d\sin\theta \end{bmatrix} \quad (1)$$

Para o controle do robô, primeiramente é atribuído um vetor 2D da direção que o robô deve se mover (u_x, u_y), a qual pode se definida ao ponderar o erro da posição do robô em relação ao objetivo ou pelo uso do algoritmo de gradiente descendente com uma função potencial conforme será visto mais a frente. Utilizando linearização por realimentação de estados estáticas, as velocidades são definidas por:

$$v_{lin} = u_x \cos\theta + u_y \sin\theta \tag{2}$$

$$v_{ang} = \frac{-u_x \sin\theta + u_y \cos\theta}{d} \tag{3}$$

III. CURVE FOLLOWING

Para o algoritmo de Curve Following, primeiramente é necessário que uma curva paramétrica seja definida. A curva selecionada neste trabalho foi a curva de Cornoid, dada pelas seguintes equações:

$$x = x_c + a\cos t(1 - 2\sin^2 t) \tag{4}$$

$$y = y_c + a\sin t(1 + 2\cos^2 t) \tag{5}$$

em que $a > 0$, $t \in R$ e (x_c, y_c) é o centro da curva.

Para que o robô seja capaz de seguir a curva, o ponto alvo irá variar de acordo com as equações paramétricas definidas. Para que a curva seja seguida com êxito, é importante que, além da velocidade definida pela ponderação do erro de posição em relação ao ponto alvo, exista uma componente do sinal de controle com a derivada da variação desses pontos na curva. Desta forma, a componente da velocidade dada pela derivada da curva é dada por:

$$\begin{bmatrix} V_{ref_x} \\ V_{ref_y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x_f}{\partial t} \\ \frac{\partial y_f}{\partial t} \end{bmatrix} = \begin{bmatrix} -a\sin t(1 - 2\sin^2 t + 4\cos^2 t) \\ a\cos t(1 + 2\cos^2 t - 4\sin^2 t) \end{bmatrix} \tag{6}$$

E o sinal de controle resultante, somado com a ponderação do erro de posição $(\Delta x, \Delta y)$, é dado por:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} K_d V_{ref_x} + K\Delta x \\ K_d V_{ref_y} + K\Delta y \end{bmatrix} \tag{7}$$

A. Resultados

O algoritmo de Curve following foi aplicado com os parâmetros $a = 10$, $(x_c, y_c) = (0, 0)$, $K_d = 1$ e $K = 5$. O resultado é demonstrado na Figura 2.

É possível perceber que o resultado exibe o fenômeno de "Chattering" que é uma vibração indesejada que impede que o robô siga a curva suavemente. É possível ajustar os ganhos dos componentes do sinal de controle para reduzir esse efeito.

Ao tentar fazer este ajuste, com $K_d = 0.1$ e $K = 0.5$ (Figura 3), foi obtido uma movimentação mais suave do robô, mas este apresentou dificuldade em convergir a curva.

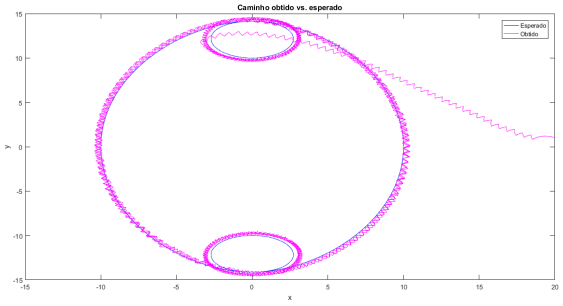


Fig. 2. Curve Following

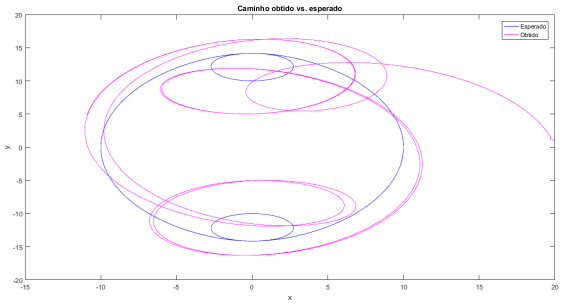


Fig. 3. Curve Following

IV. ATTRACTIVE/REPULSIVE POTENTIAL FIELDS

O algoritmo que utiliza campos potenciais, atrativo e repulsivo, lança mão de uma pesquisa online de um caminho a ser seguido, em que o robô é atraído pelo objetivo e repelido por obstáculos.

Definidas funções de potencial, é desejado que o robô caminhe de na direção que as minimize. Para isso é utilizado o algoritmo de gradiente descendente. O gradiente é o vetor que aponta para a direção que maximiza uma função escalar, portanto, para obter a direção de minimização, inverte-se o gradiente. O sinal de controle para seguir o caminho do inverso do gradiente é dado por:

$$u = -\alpha\vec{\nabla}U(q(t)) \tag{8}$$

Com potenciais de atração e repulsão, o gradiente resultante é dado por:

$$\vec{\nabla}U(q(t)) = \vec{\nabla}U_{att}(q(t)) + \vec{\nabla}U_{rep}(q(t)) \tag{9}$$

A. Potencial de Atração

O potencial de atração pode ser calculado analiticamente dado conhecimento do objetivo do robô, basta escolher uma função que cresça com a distância $(d(q, q_f))$ do robô (q) ao objetivo (q_f).

Para implementação desenvolvida, foram escolhida uma função potencial de atração composta por uma parte cônica e outra quadrática. A função potencial cônica é utilizada até o robô chegar a uma determinada distância d^* do objetivo. A partir dessa distância, a função potencial quadrática é

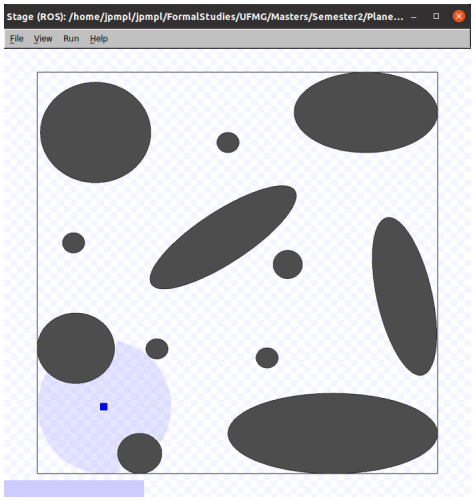


Fig. 4. Cenário Attractive/Repulsive potentials

utilizada. Desta forma evita-se que velocidades extremas sejam obtidas em pontos distantes e que ao aproximar do objetivo a velocidade do robô varie mais suavemente.

O gradiente utilizado é dado por:

$$\vec{\nabla}U_{att}(q) = \begin{cases} \zeta(q - q_f), & d(q, q_f) \leq d^* \\ \frac{d^*\zeta(q-q_f)}{d(q,q_f)}, & d(q, q_f) > d^* \end{cases} \quad (10)$$

B. Potencial Repulsivo

É considerado que o robô não tem conhecimento do ambiente. Portanto, o potencial Repulsivo é calculado a partir dos dados de um sensor de distância com um alcance finito.

A função potencial repulsiva deve crescer com a aproximação de um obstáculo. Escolhendo uma função potencial como a seguinte:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{D(q)} - \frac{1}{Q^*}), & D(q) \leq Q^* \\ 0, & D(q) > Q^* \end{cases} \quad (11)$$

O gradiente é dado por:

$$\vec{\nabla}U_{rep_i}(q) = \begin{cases} \eta(\frac{1}{d_i(q)} - \frac{1}{Q^*})\frac{1}{d_i^2(q)}, & d_i(q) \leq Q^* \\ 0, & d_i(q) > Q^* \end{cases} \quad (12)$$

em que $d_i(q)$ são mínimos locais identificados pelo sensor de distância.

Como podem existir múltiplos obstáculos, o gradiente é calculado para cada um dos mínimos locais detectados e sua resultante é somada:

$$\vec{\nabla}U_{rep}(q) = \sum_{i=1}^{n} \vec{\nabla}U_{rep_i}(q) \quad (13)$$

C. Resultados

O algoritmo implementado foi testado no ambiente mostrado pela Figura 4.

Para $\zeta = 5$, $\eta = 100$ e $q_f = (20, 15)$, o resultado é apresentado na Figura 5, que mostra que o robô chega muito próximo do objetivo definido.

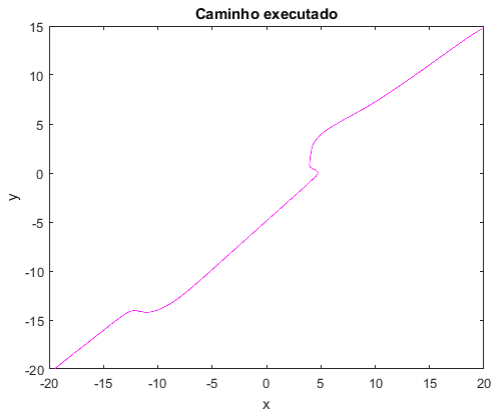


Fig. 5. Caminho

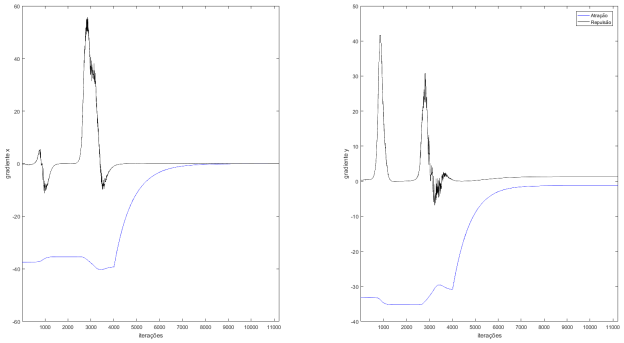


Fig. 6. Gradientes

Os gradientes atrativo e repulsivo neste caso são apresentados na Figura 6.É possível perceber que embora o robô chegue próximo ao ponto objetivo, o potencial atrativo não alcança 0, o que significaria que o robô chegou exatamente no alvp;

Foi também realizado um experiemento com os argumentos $\zeta = 5$, $\eta = 1000$ e $q_f = (20, 15)$, cujo caminho realizado consta na Figura fig:ar$_caminho_2$.

Neste caso fica mais evidente o fato que que o algoritmo que utiliza potenciais está sujeito a encontrar um mínimo local e não conseguir mais progredir em direção ao objetivo. A evolução do gradiente na Figura 8 demostra esse fenômeno.

V. TANGENT BUG

O algoritmo de Tangent Bug é uma evolução dos algoritmos de Bug 1 e Bug 2. Neste algoritmo, utiliza-se das informações de um sensor de distância com alcance finito para encontrar obstáculos e determinar um caminho mais otimizado em comparação com seus predecessores.

Este algoritmo opera em dois modos, o de "motion-to-goal" e o de "boundary following" conforme explicado no Capítulo 2, Seção 2.2 de [1].

VI. Detalhes de Implementação

Salientando alguns pontos da implementação realizada:

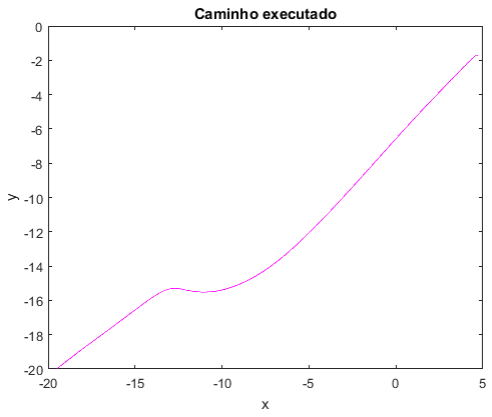


Fig. 7. Caminho

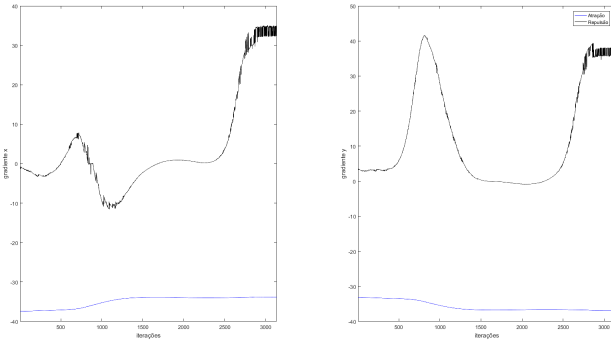


Fig. 8. Caminho

- O cálculo dos intervalos de continuidade foram realizados pela identificação dos pontos de transição em um threshold definido 0.01 (arbitrariamente escolhido) a menos que o range máximo do sensor. Figura 9.
- O possível evento de um ponto objetivo do "motion-to-goal" ter o caminho ao objetivo final obstruído por outro obstáculo a frente não foi considerada. Figura 10.
- Devido a dificuldade de encontrar um ponto a uma distância d^* dos limites dos intervalos de continuidade na direção normal em relação a superfície do obstáculo, foi atribuido um modo "normal" no "motion-to-goal" de forma que se o robô detectar que entrou no threshold definido de aproximação do obstáculo, ele é repelido pela normal. Figura 11.
- O chaveamento para seguimento de borda está, num primeiro momento, levando o robô à distância limite do threshold do obstáculo. Figura 12.

A. *Resultados*

Foram realizados quatro experimentos para demonstração dos resultados. Os 3 primeiros com solução factível e o último sem solução.

Para os 3 primeiros experimentos, o cenário utilizado foi o da Figura 13.

Para o último experimento, o cenário utilizado foi o da figura 14.

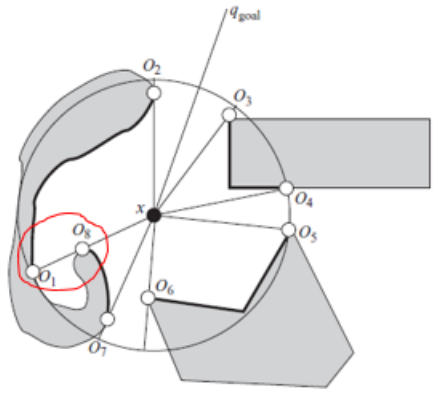


Fig. 9. Caso não considerado destacado em vermelho

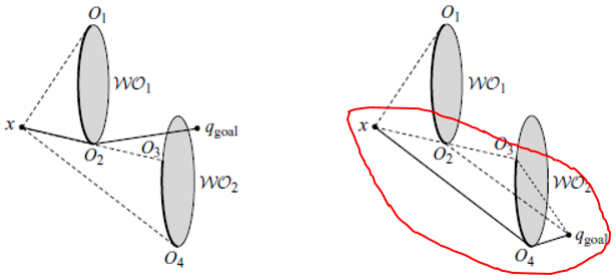


Fig. 10. Caso não considerado destacado em vermelho

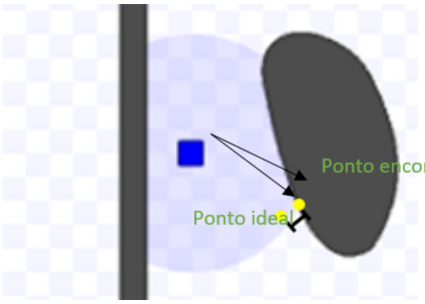


Fig. 11. Dificuldade em encontrar ponto objetivo que evitasse colisão

Fig. 12. Caminho em direção ao threshold do obstáculo ao chavear para "boundary-following"

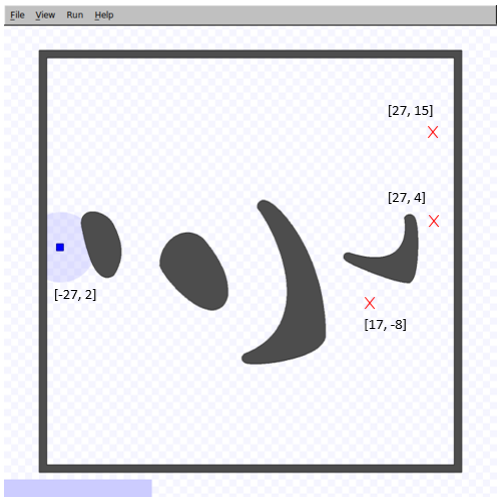


Fig. 13. Cenário dos experimentos 1, 2 e 3

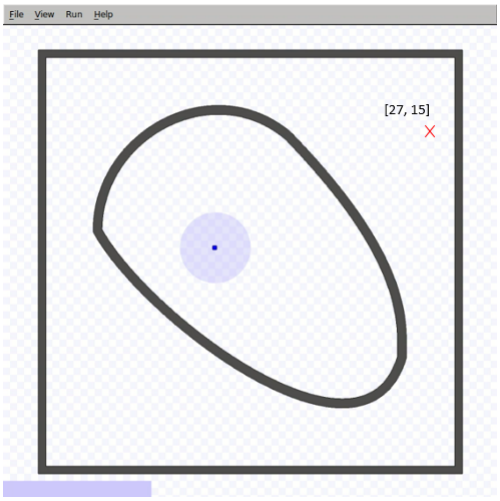


Fig. 14. Cenário do experimento 4

1) Experimento 1: Neste experimento o objetivo definido foi $q_f = (27, 4)$. O caminho resultante e os modos em que o robô se encontra em cada ponto são mostrados na Figura 15.

2) Experimento 2: Neste experimento o objetivo definido foi $q_f = (27, 15)$. O caminho resultante e os modos em que o robô se encontra em cada ponto são mostrados na Figura 16.

3) Experimento 3: Neste experimento o objetivo definido foi $q_f = (17, -8)$. O caminho resultante e os modos em que o robô se encontra em cada ponto são mostrados na Figura 17.

4) Experimento 4: Neste último experimento foi testada a capacidade do robô identificar se não há solução para a tarefa desejada. O objetivo definido foi $q_f = (27, 15)$. O caminho resultante e os modos em que o robô se encontra em cada ponto são mostrados na Figura 18.

VII. WAVE FRONT

Em anexo.

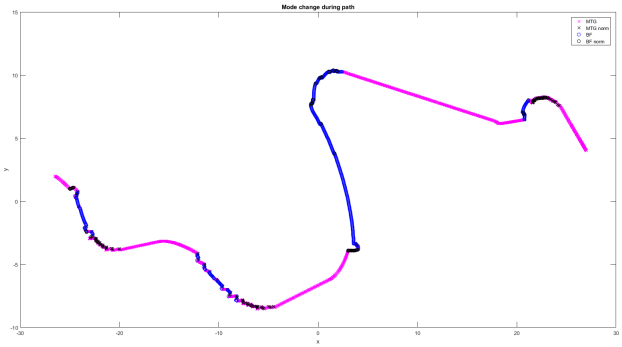


Fig. 15. Caminho realizado e modos do robô para $q_f = (27, 4)$

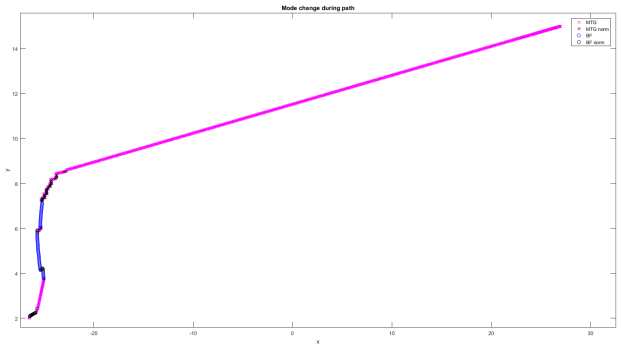


Fig. 16. Caminho realizado e modos do robô para $q_f = (27, 15)$

VIII. CONCLUSÃO

Todos algoritmos apresentados conseguem realizar a tarefa de alcançar um ponto objetivo (ou informar que não existe solução), com algumas exceções em casos especiais. Ao implementá-los, é interessante perceber que, embora na teoria os algoritmos sejam bem elaborados e seu funcionamento bem descrito, existe a dificuldade em transcrevê-los para códigos robustos. Essa constatação é importante, pois traz a reflexão de que, se existe a dificuldade em simulação, muitos outros problemas provavelmente surgem ao trazer os algoritmos implementados para uma aplicação em um robô real.

REFERENCES

[1] S. H. G. K. W. B. L. K. Howie Choset, Kevin Lynch and S. Thrun, *Principles of Robot Motion*, 2005.

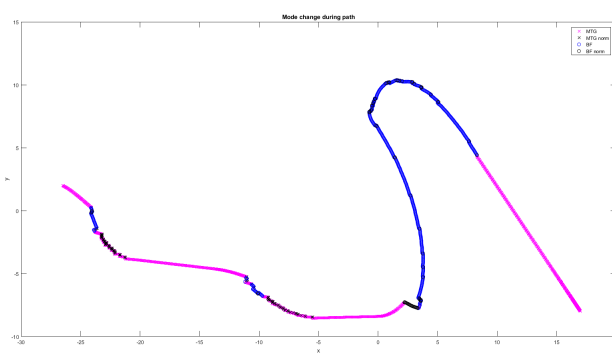


Fig. 17. Caminho realizado e modos do robô para $q_f = (17, -8)$

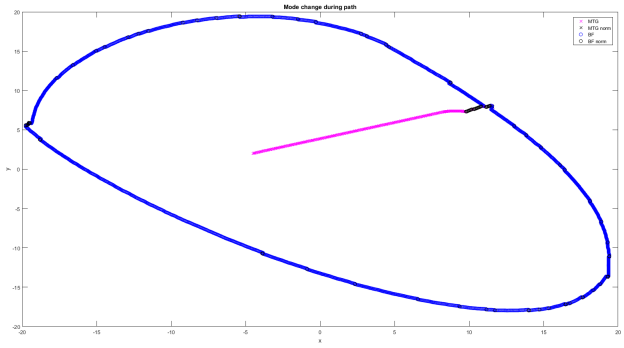


Fig. 18. Caminho realizado e modos do robô para $q_f = (27, 15)$. Caso sem solução.

Algoritmo wafefront

Para este algoritmo criamos um cenário representado por uma imagem em formato pgm, de 600x600 pixels. Também usamos a escala de 1 pixel para 1 metro.

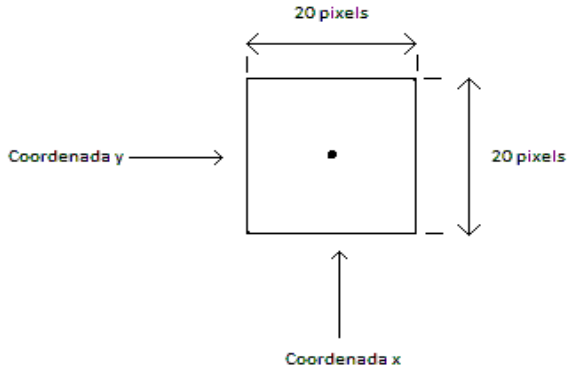
O simulador foi o Stage e também usamos o pycharm como plataforma IDE para escrita do nosso código.

Em nossos testes posicionamos o alvo na coordenada (x,y)=(250, 250) e ponto de partida do nosso robô na coordenada (x,y)=(250, -250).

No python criamos um objeto imagem que nos deu uma matriz 600x600, onde cada ponto da matriz representava um pixel da imagem, como vocês podem ver na figura abaixo:

```
obj_img: None = cv2.imread("/home/eudes/catkin_ws/src/trab_questao4/worlds/cenario.pgm")
```

Após a obtenção de um objeto que representasse o cenário dividimos o cenário em células obtendo uma matriz 30x30. Abaixo uma figura para entendimento destas células criadas.



Cada célula desta nova matriz possui 4 atributos, sendo eles:

- 1: coord_x
- 2: coord_y
- 3: obstáculo
- 4: distancia para o alvo

Nosso algoritmo calculo e deu valores a todos os atributos de cada célula. Assim coord_x e coord_y indicam a posição do centro da célula., em pixels.

O parâmetro obstaculo indica a existência de obstaculo ou não. Quando o mesmo assume o valor “0”, isto nos indica que a célula está livre. Caso assuma o valor “1” indica a presença de obstáculo naquele espaço.

O parâmetro “distancia_para_alvo” indica o peso da célula até chegar o alvo. Quanto maior este valor, maior a distância para se atingir o alvo. Assim o alvo recebe peso=1 e às células laterais são adicionadas 1 unidade a medida que se afasta do alvo, como se vê na figura abaixo. Esse processo se repete até chegar ao fim do mapa ou a um obstáculo.

Então criamos uma matriz de 100 linhas e duas colunas que indicará o caminho de células por onde o robô deverá passar até atingir o alvo. A primeira coluna representa o índice y da célula para onde o robô deve se movimentar. A segunda coluna representa o índice x.

As equações para usadas para determinação das velocidades linear e angular são como:

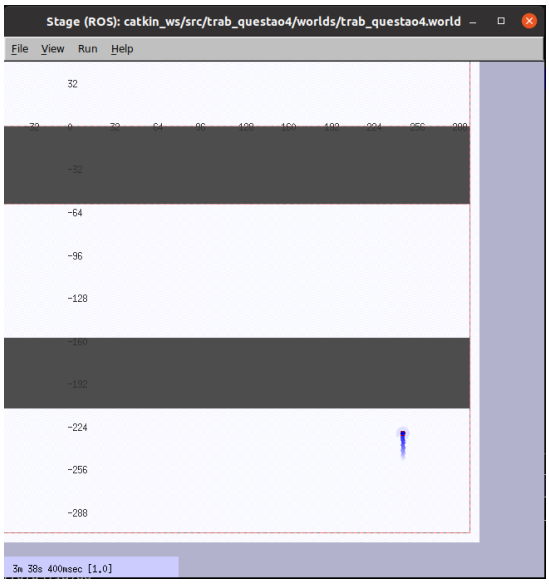
$$\begin{pmatrix} V \\ \omega \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\frac{\sin(\theta)}{d} & \frac{\cos(\theta)}{d} \end{pmatrix} \times \begin{pmatrix} U_x \\ U_y \end{pmatrix}$$

$$\begin{pmatrix} U_x \\ U_y \end{pmatrix} = U = V_{ref} + KP$$

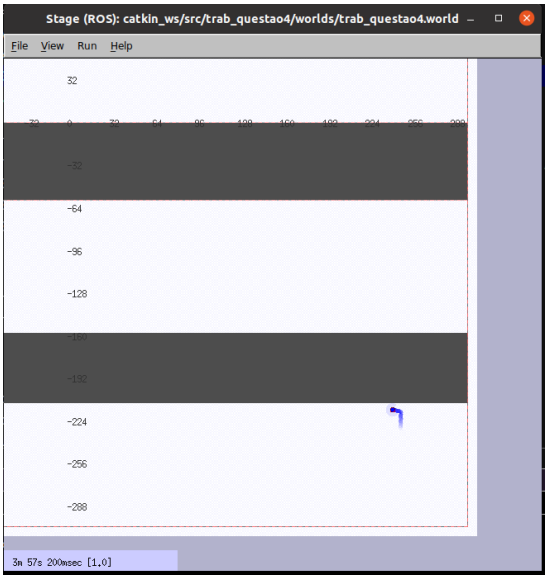
$$V_{ref} = \begin{pmatrix} V_{refx} \\ V_{refy} \end{pmatrix}, P = \begin{pmatrix} \Delta X \\ \Delta Y \end{pmatrix}$$

Desta forma passamos para a simulação obtendo os seguintes resultados.

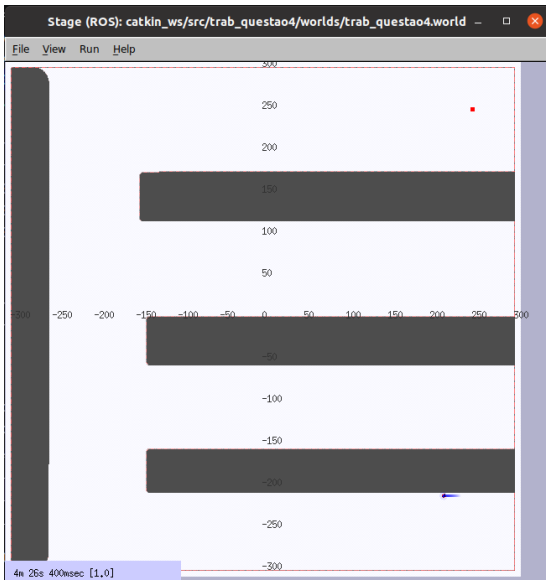
1 – Partida do robô:



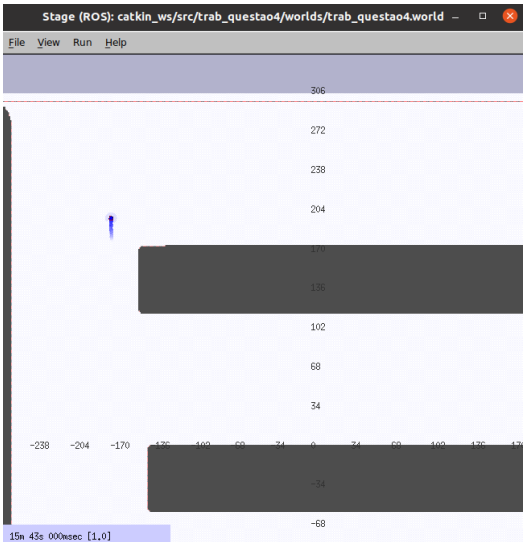
2 – Desviando do primeiro obstáculo:



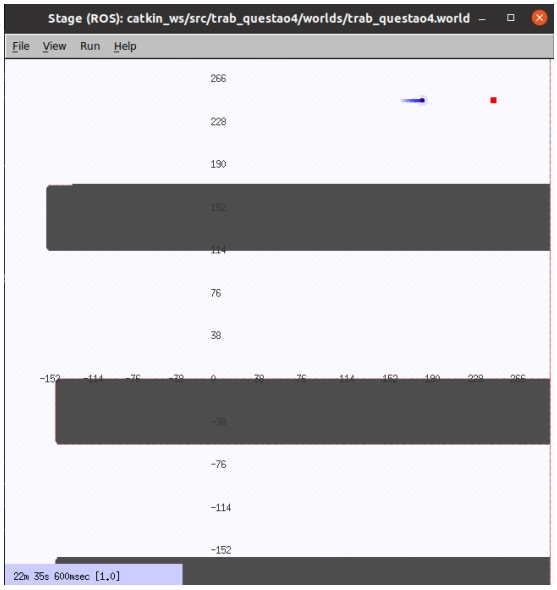
3 – Contornando o obstáculo:

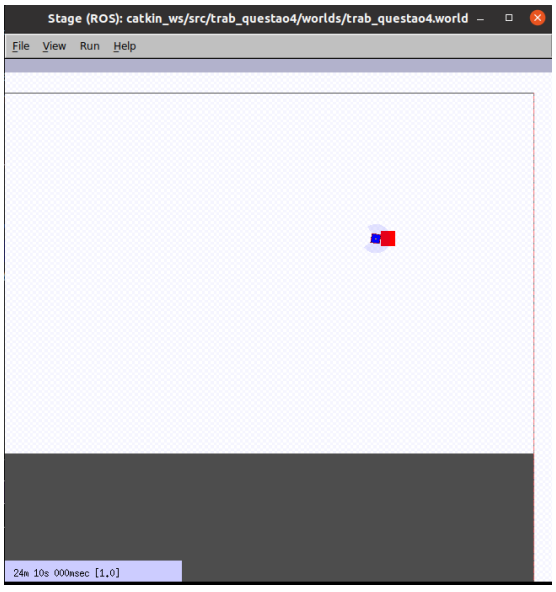


4 – Contornando novo obstáculo:



5 – Chegando ao alvo





Conclusão: Para melhorar o programa podemos implementar o ponto de partida do robô como entrada do usuário.