Universidade Federal de Itajubá

Campus Itabira

ECAi21.2 – Laboratório de Robótica Móvel 30 de novembro de 2022

Docente: Andŕe Chaves Magalhães

Discente:

– Gabriel Toffanetto França da Rocha – 2019003646

Exercício 6 – Campos Potênciais

Sumário

1 Introdução

A navegação de um robô por um ambiente com obstáculos de forma reativa pode ser problemática, uma vez que se a decisão de desvio tomada for aleatória, não pode-se garantir que o robô chegará ao destino final em tempo de execução. Desta forma, a técnica de campos potenciais pode ser utilizada para, utilizando o posicionamento do robô, dado por GPS ou odometria, e um sensor LiDAR para a detecção de obstáculo, navegar do ponto inicial até o ponto final com base em um vetor de força resultante, que é calculado com base nos campos criados pelo destino e pelos obstáculos. O alvo cria um campo potencial atrativo, que faz com que apareça uma força no robô que aponta para o alvo. Já os obstáculos geram um campo repulsivo, que provoca uma força no robô na direção contrária à do obstáculo. Ao somar todas essas forças, se obtém uma força resultante que pode ser utilizada como modelo para o vetor velocidade do robô, permitindo que o robô chegue até o seu destino sem bater em nenhum obstáculo.

Ressalta-se que essa estratégia está submetida à mínimos locais, uma vez que podem ocorrer situações onde as forças repulsivas vão cancelar a força atrativa, criando um mínimo local onde o robô ficará preso, considerando que chegou à posição de destino. Porém, com o uso do simulador Gazebo, onde são simulados ruídos na medição dos sensores e a inércia do robô, esses mínimos locais podem ser evitados naturalmente, uma vez que dificilmente a força resultante será completamente nula, e também devido à inercia do robô, que faz com que ao passar pelo mínimo local, o robô avance o ponto onde ele ocorre, saindo do mínimo local devido à dinâmica do sistema.

O ponto de destino para a navegação do robô é setado por meio de duas constantes, `TARGET_X` e `TARGET_Y`, localizadas na seção de constantes do código fonte.

2 Campos de atração

Para atração do robô por meio do ponto alvo, $\mathbf{q}_{goal}$, é utilizado um campo vetorial parabólico, enunciado em (1).

$$U_{att}(\mathbf{q}) = \frac{k_{att}}{2} \left(\rho_{goal}(\mathbf{q})\right)^2 \tag{1}$$

Sendo:

– k_{att} o ganho do campo;

– $\rho_{goal}(\mathbf{q})$ a distância euclidiana do robô até o alvo.

Obtendo a força de atração, $F_{att}(\mathbf{q})$, em (2).

$$\vec{F}_{att}(\mathbf{q}) = k_{att} \left(\mathbf{q}_{goal} - \mathbf{q}\right) \tag{2}$$

A força de atração é obtida via código por meio da função `getFatt()`, que recebe como entrada a posição do alvo e tem como saída o vetor da força de atração, passado por referência.

```
void PotentialFieldControl::getFatt(Ponto goal, Ponto &Fatt){
  Fatt.x = (goal.x - robot_point.x)*K_ATT;
  Fatt.y = (goal.y - robot_point.y)*K_ATT;
}
```

3 Localização dos obstáculos

Para o cálculo das forças de repulsão sobre o robô, é necessário obter a posição dos obstáculos que estão em volta do mesmo. Os obstáculos são detectados por meio de um sensor LiDAR, onde são despejados vários raios *laser*, em ângulos conhecidos e é medido a distância do sensor até o obstáculo para cada raio.

Desta forma, a posição de cada obstáculo $\mathbf{q}_{obs,i}$ é dada por meio do *range* medido pelo LiDAR, r_i, e pelo ângulo desse raio, θ_i. No sistema de coordenadas do LiDAR, a posição pode ser obtida por meio de relações trigonométricas, como mostrado em (3).

$$\mathbf{q}_{obs,i_{LiDAR}} = \begin{bmatrix} r_i \cos\theta_i \\ r_i \sin\theta_i \end{bmatrix} \tag{3}$$

Para obter a posição do obstáculo em relação ao sistema de coordenadas globais, aplica-se a matriz de rotação $\mathbf{R}(\theta)$, e a translação em relação à posição do LiDAR em relação ao robô e a translação do robô em relação à origem do sistema de coordenadas globais, como mostrado em (4).

$$\mathbf{q}_{obs,i} = \begin{bmatrix} \cos\theta_R & -\sin\theta_R \\ \sin\theta_R & \cos\theta_R \end{bmatrix} \begin{bmatrix} r_i \cos\theta_i \\ r_i \sin\theta_i \end{bmatrix} + \mathbf{q} + \mathbf{q}_{LiDAR} \tag{4}$$

Sendo:

- $\mathbf{q}$ a posição do robô;
- $\mathbf{q}_{LiDAR}$ a posição do LiDAR em relação ao robô;
- θ_R o ângulo do robô em relação ao eixo z (*Yaw*).

A implementação em código é feita por meio da função `getPose()`, que recebe como entrada o *range* e o ângulo, e tem como saída por referência o ponto do obstáculo.

```
void PotentialFieldControl::getPose(double range, double theta, Ponto &
    obstacle){
     double xr = range*cos(theta);
     double yr = range*sin(theta);
     obstacle.x = (xr*cos(robot_orientation.yaw) - yr*sin(robot_orientation.
    yaw)) + robot_point.x + LIDAR_X;
     obstacle.y = (xr*sin(robot_orientation.yaw) + yr*cos(robot_orientation.
    yaw)) + robot_point.y + LIDAR_Y;
}
```

4 Campos de repulsão

$$U_{rep,i}(\mathbf{q}) = \begin{cases} \frac{k_{rep,i}}{\gamma}\left(\frac{1}{\rho_i(\mathbf{q})} - \frac{1}{\rho_{0,i}}\right)^{\gamma} & \text{, se } \rho(\mathbf{q}) \leq \rho_0 \\ 0 & \text{, se } \rho(\mathbf{q}) > \rho_0 \end{cases} \tag{5}$$

$$\vec{F}_{rep,i}(\mathbf{q}) = \begin{cases} \frac{k_{rep,i}}{\rho_i^2(\mathbf{q})}\left(\frac{1}{\rho_i(\mathbf{q})} - \frac{1}{\rho_{0,i}}\right)^{\gamma-1}\frac{q-q_{obs,i}}{\rho_i(\mathbf{q})} & \text{, se } \rho(\mathbf{q}) \leq \rho_0 \\ 0 & \text{, se } \rho(\mathbf{q}) > \rho_0 \end{cases} \tag{6}$$

- $\mathbf{q}$ a posição do robô;
- $\mathbf{q}_{obs,i}$ a posição do obstáculo;
- $\rho_i(\mathbf{q})$ a distância euclidiana entre o robô e o obstáculo;
- $\rho_{0,i}$ a menor distância de influência do obstáculo;
- i o número do raio que detectou o obstáculo.

Considerando $\gamma = 2$, têm-se a seguinte implementação em código, por meio da função `getFr()`.

```
void PotentialFieldControl::getFr(Ponto obstacle, Ponto &Fr){
  double p_i = distPoints(obstacle,robot_point);
  Fr.x = K_REP*1/(p_i*p_i*p_i)*(1/(p_i) - 1/(P0))*(robot_point.x - obstacle.
   x);
  Fr.y = K_REP*1/(p_i*p_i*p_i)*(1/(p_i) - 1/(P0))*(robot_point.y - obstacle.
   y);
}
```

5 Vetor de força resultante

Todos os vetores de força são armazenados em um `vector`, e somados por meio da função `getFres()`, matematicamente mostrado em (7). Com isso, obtêm-se o vetor resultante que guia o vetor de velocidade do robô. A partir do vetor resultante, pode-se obter o erro angular entre a direção do robô e a do vetor, obtendo a velocidade angular proporcional a esse erro, e também a velocidade linear, proporcional à magnitude do vetor.

$$\vec{F}_{res}(\mathbf{q}) = \vec{F}_{att}(\mathbf{q}) + \sum_{i=0}^{N} \vec{F}_{rep,i}(\mathbf{q}) \tag{7}$$

A implementação em código é dada pela função `getFres()`.

```
void PotentialFieldControl::getFres(std::vector<Ponto> F, Ponto &Fres){
  Fres.x = 0;
  Fres.y = 0;
  for(auto Fi : F){
    Fres.x += Fi.x;
```

```
    Fres.y += Fi.y;
  }
}
```

6 Leis de controle

Para o controle de velocidade do robô, é utilizado o erro de ângulo entre a direção do robô e o vetor resultante, e o módulo do vetor resultante. O ângulo do vetor resultante é dado por meio da função `getYaw()`, e o módulo por meio da função `getMag()`. A velocidade linear é calculada também com uma parcela dependente do erro angular, possibilitando uma melhor manobra em situações onde o erro angular é grande, onde o robô diferencial irá tender a parar de se locomover linearmente e irá rotacionar sobre o menor raio possível, em seu torno do seu próprio eixo.

```
double getYaw(Ponto robot, Ponto point){
    Ponto u = {point.x - robot.x, point.y - robot.y, point.z - robot.z};
    Ponto x = {1,0,0};
    return (modulo(u)==0) ? 0 : copysign(acos(prodInter(x,u)/modulo(u)),u.y)
   ;
}
double getMag(Ponto point){
  return sqrt(point.x*point.x + point.y*point.y);
}
```

Desta forma, as velocidades são dadas por:

$$v = \begin{cases} ||\vec{F}_{res}|| \cdot K_L - \left|\left(\angle\vec{F}_{res} - \theta_R\right) \cdot K_{Ld}\right| & \text{, se } 0 < v \leq v_{max} \\ v_{max} & \text{, se } v > v_{max} \\ 0 & \text{, se } v \leq 0 \end{cases} \tag{8}$$

$$\omega = \begin{cases} \left(\angle\vec{F}_{res} - \theta_R\right) \cdot K_A & \text{, se } \omega \leq \omega_{max} \\ \omega_{max} & \text{, se } \omega > \omega_{max} \end{cases} \tag{9}$$

A implementação em código dessas leis de controle é dada por:

```
linear_speed = Fres_mag*Kp_L - abs(angle_error*Kp_Ld);
angular_speed = angle_error*Kp_A;

// Satuadores
if(linear_speed > LINEAR_SPEED_MAX)
    linear_speed = LINEAR_SPEED_MAX;
else if(linear_speed < 0)
    linear_speed = 0;
if(abs(angular_speed) > ANGULAR_SPEED_MAX)
    angular_speed = copysign(ANGULAR_SPEED_MAX,angular_speed);
}
```