Universidade Federal de Itajubá

Campus Itabira

ECAi21.2 – Laboratório de Robótica Móvel 17 de dezembro de 2022

Docente: Andŕe Chaves Magalhães

Discente:

- Gabriel Toffanetto França da Rocha – 2019003646

Exercício 7 –*Feedback Linearization*

Sumário

---

1 Introdução

A estratégia de controle *Feedback Linearization* utiliza de uma realimentação de estados para manter o controle de um robô sob a sua trajetória. Com o equacionamento matemático de uma curva, ou com a entrada de coordenadas periodicamente, é possível definir o caminho que o robô deve seguir, e garantir que o erro entre a posição do robô e a trajetória tenderão para 0 com o tempo tendendo para infinito.

$$\lim_{t\to\infty} e(t) = 0$$

2 Cálculo da trajetória

A curva utilizada para representar a trajetória do robô é uma Lemniscata de Bernoulli, com largura igual à 20 metros. A curva foi representada na forma parametrizada, variando os valores de x e y em função do tempo, como mostrado em (1) e (2).

$$x = \frac{\alpha \cos \frac{t}{\beta}}{1 + \sin^2 \frac{t}{\beta}} \tag{1}$$

$$y = \frac{\alpha \sin \frac{t}{\beta} \cos \frac{t}{\beta}}{1 + \sin^2 \frac{t}{\beta}} \tag{2}$$

Onde:

- α é a amplitude da curva;
- β é uma compressão do tempo;
- t é o tempo em segundos.

Dessa forma, pode-se realizar a evolução da trajetória no tempo, obtendo as coordenadas da curva para cada instante de amostragem. A constante α é utilizada para dar o tamanho da Lemniscata de Bernoulli, enquanto a constante β é utilizada para definir a velocidade com que a curva evoluí, que varia inversamente com a constante.

A implementação da curva é feita em código utilizando a função `now()` do ROS 2 para obter o tempo atual do sistema e aplicando à curva o tempo relativo ao início da execução do código, garantindo que a curva sempre irá iniciar da sua origem. A posição da curva é dada pela função `getLemniscatePoint()`, mostrada abaixo:

```
void FeedbackLinearizationControl::getLemniscatePoint(double &x, double &y){
    double t = (now().nanoseconds())*1e-9 - t_init;
    x = (A*cos(t/B))/(1+sin(t/B)*sin(t/B));
    y = (A*sin(t/B)*cos(t/B))/(1+sin(t/B)*sin(t/B));
}
```

3 *Feedback Linearization*

Uma vez feito o desenvolvimento do *Feedback Linearization*, obtém-se a matriz de realimentação de estados A'^{-1}, mostrada em (3), que é utilizada para montar a lei de controle mostrada em (4). Considerou-se um valor de $d = 0,2$.

$$A'^{-1} = \frac{1}{d}\begin{bmatrix} d\cos\theta & d\sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{d} & \frac{\cos\theta}{d} \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = A'^{-1}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{d} & \frac{\cos\theta}{d} \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{4}$$

O valor das entradas u_1 e u_2 são dados por (5) e (6), sendo o valor do ganho $K = 0,3$.

$$u_1 = \dot{x}_r + K(x - x_r) \tag{5}$$

$$u_2 = \dot{y}_r + K(y - y_r) \tag{6}$$

O valor da velocidade da trajetória, $\dot{x}_r$ e $\dot{y}_y$, é calculado a cada iteração do algoritmo, como mostrado em (7) e (8). O valor com sub-índice n é a posição atual da curva e o valor com sub-índice $n-1$ é o valor anterior, enquanto T_s é o período de amostragem, dado por 20 ms.

$$\dot{x}_r = \frac{x_n - x_{n-1}}{T_s} \tag{7}$$

$$\dot{y}_r = \frac{y_n - y_{n-1}}{T_s} \tag{8}$$

Dessa forma, da lei de controle (4), vem as equações para a velocidade linear, (9), e angular, (10).

$$v = \cos\theta u_1 + \sin\theta u_2 \tag{9}$$

$$\omega = -\frac{\sin\theta}{d}u_1 + \frac{\cos\theta}{d}u_2 \tag{10}$$

A função `getFeedbackLinearizationControl()` realiza a implementação do controle em código, como mostrado a seguir.

```
void FeedbackLinearizationControl::getFeedbackLinearizationControl(double &v
    , double &w, double &x, double &y){
    double x_dot_r = 0.5, y_dot_r = 0.5;

    if(x_old == 0 && y_old == 0){
        x_old = x;
        y_old = y;
    }
    else{
```

```
        x_dot_r = (x - x_old)/20e-3;
        y_dot_r = (y - y_old)/20e-3;
        x_old = x;
        y_old = y;
    }

    double u1 = x_dot_r + K*(x - robot_point.x);
    double u2 = y_dot_r + K*(y - robot_point.y);

    v = cos(robot_orientation.yaw)*u1 + sin(robot_orientation.yaw)*u2;
    w = -sin(robot_orientation.yaw)/D*u1 + cos(robot_orientation.yaw)/D*u2;
}
```