ECAI21.2 - Robótica Móvel (Prática) - Turma 01

Prof. André Chaves Magalhães
Dezembro/2023
Universidade Federal de Itajubá - *Campus* de Itabira

RELATÓRIO - PROJETO 4

Emilly Lara Deiró[*], Rodrigo[*], Saulo Labiapari[*], Vinicius de Sousa David[*], Wilgner Luís Pereira Barreto[*]

[*] *Universidade Federal de Itajubá - Campus de Itabira*
Rua Irmã Ivone Drumond, 200 - Distrito Industrial II - 35903-087
Itabira, Minas Gerais, Brasil

E-mails: emillydeiro@unifei.edu.br, @unifei.edu.br, saulolabiapari@unifei.edu.br, d2019005631@unifei.edu.br, wilgner.v9@gmail.com

Resumo— Este trabalho propõe o desenvolvimento de um controlador para um robô de acionamento diferencial, com o objetivo de permitir que o robô siga continuamente uma curva em forma de oito, denominada Lemniscata de Bernoulli. Os parâmetros da curva, como tamanho, inclinação e posição inicial, serão configuráveis como entradas do programa. O controlador será implementado e testado no ambiente de simulação StageROS, demonstrando a capacidade do robô de seguir a curva com precisão, adaptando-se às variações nos parâmetros da curva. Este projeto envolve conceitos de robótica, controle e programação, com potenciais aplicações em robótica autônoma e navegação em ambientes complexos.

Palavras-chave— Controlador, Robô, Lemniscata de Bernoulli, StageROS

1 Introdução

A presente pesquisa tem como propósito a implementação de um controlador destinado a um robô móvel com acionamento diferencial, com a finalidade de possibilitar sua capacidade de convergir e circular indefinidamente em uma curva de formato oito, especificamente a Lemniscata de Bernoulli. O referido controle será executado por meio do emprego das técnicas de feedback linearization e do ambiente de visualização 3D denominado RViz no contexto do ROS (Sistema Operacional de Robôs). Além disso, é relevante destacar que os parâmetros da curva em forma de oito servirão como entradas para o programa, conferindo flexibilidade ao sistema. A demonstração prática deste trabalho será conduzida no ambiente StageROS.

A técnica de feedback linearization é amplamente utilizada no controle de sistemas não lineares. Essas técnicas são aplicadas a sistemas de controle não lineares descritos da seguinte forma:

$$\frac{dx}{dt} = f(x) + g(x)u \tag{1}$$

$$y = h(x) \tag{2}$$

Nesse contexto, onde "x"representa o estado, "u"denota a entrada e "y"é a saída do sistema, a abordagem consiste em transformar o sistema de controle não linear em um equivalente linear, por meio de uma transformação de variáveis apropriada e uma entrada de controle adequada. De modo específico, busca-se encontrar uma transformação de coordenadas e uma entrada de controle que permitam que a dinâmica do estado "x"nas novas coordenadas adote a forma de um sistema de controle linear, que seja controlável.

Ademais, outro conceito importante é o ambiente de visualização tridimensional denominado RViz é uma ferramenta incorporada ao Robot Operating System (ROS), que desempenha um papel essencial no domínio da robótica. O RViz viabiliza a visualização em tempo real de dados provenientes de sensores, além de possibilitar a representação em 3D do modelo do robô e do ambiente circundante. Dessa forma, ele se torna uma peça fundamental na análise, depuração e desenvolvimento de sistemas de controle de robôs, permitindo a visualização de informações cruciais para o entendimento do comportamento do robô e sua interação com o ambiente.

2 Objetivos

Este estudo tem como objetivo principal implementar um controlador para um robô móvel com acionamento diferencial, permitindo que o robô siga uma trajetória em forma de oito (Lemniscata de Bernoulli) de forma contínua. A implementação será baseada nas técnicas de feedback linearization e será visualizada em tempo real no ambiente RViz, integrado ao ROS. A demonstração prática ocorrerá no ambiente StageROS.

```
// Initialize ROS node
ros::init(argc, argv, "lemniscate_trajectory");
ros::NodeHandle nh;

// Get parameters from command line or launch file
// Get parameters from command line or launch file
ros::param::get("~a", a); // half-width of lemniscate
ros::param::get("~Ts", Ts); // angle parameter increment of lemniscate

ros::Rate loop_rate(int(1/ (Ts)));
t=0;

// Create a publisher object for the Twist message topic
ros::Publisher twist_pub = nh.advertise<geometry_msgs::Twist>("desired_twist", 1000);
```

Figura 1: Declaração de variáveis

```
while (ros::ok()) {
    // Calculate desired position and orientation of robot on lemniscate using parametric equations
    double t_new = t + Ts;

    double x_d = a * sqrt(2) * cos(t_new) / (sin(t_new) * sin(t_new) + 1);
    double y_d = a * sqrt(2) * cos(t_new) * sin(t_new) / (sin(t_new) * sin(t_new) + 1);
    double theta_d = atan2(y_d, x_d);

    double v_x_d = (a * sqrt(2) * cos(t) / (sin(t) * sin(t) + 1));
    double v_y_d = (a * sqrt(2) * cos(t) * sin(t) / (sin(t) * sin(t) + 1));


    geometry_msgs::Twist desired_twist;
    desired_twist.linear.x = x_d;
    desired_twist.linear.y = y_d;
    desired_twist.angular.z = theta_d;
    desired_twist.angular.x = v_x_d;
    desired_twist.angular.y = v_y_d;


    twist_pub.publish(desired_twist);
```

Figura 2: Calculo da Lemniscata de Bernoulli

3 Ferramentas

Para a realização deste projeto foram utilizados o Sistema Operacional de Robôs (ROS) e o ambiente de simulação StageROS. O ROS é um framework de software livre amplamente utilizado em robótica, que fornece uma plataforma para desenvolvimento, integração e execução de software em robôs. Ele oferece uma ampla variedade de bibliotecas, ferramentas e pacotes para programação de robôs, incluindo controle, percepção, planejamento e comunicação. O ROS foi utilizado neste projeto para implementar o controlador do robô e para visualizar os dados em tempo real no ambiente de simulação.

O StageROS é um ambiente de simulação 2D para robôs móveis que permite a criação de cenários virtuais para testar e validar algoritmos de controle e navegação. Ele é baseado no simulador Stage, que é um simulador de robôs de código aberto e multiplataforma. O StageROS foi utilizado neste projeto para simular o robô de acionamento diferencial e para testar o controlador desenvolvido pelos alunos. Ele permitiu que os alunos avaliassem o desempenho do controlador em diferentes cenários e ajustassem os parâmetros do controlador para melhorar a precisão e a estabilidade do robô.

4 Desenvolvimento

Durante o desenvolvimento deste projeto, foram realizadas duas etapas principais: a criação da trajetória e a implementação do controlador do robô.

4.1 Trajetória

a criação da trajetória em forma de oito, denominada Lemniscata de Bernoulli, que seria seguida pelo robô. Para isso, foram definidos os parâmetros da curva, como tamanho, angulo e o incremento do angulo atual, que seriam configuráveis como entradas do programa, no *Launch File.*

É preciso fazer a criação do nó de comunicação da trajetória e o mesmo foi feito usando os padrões já usados na sala de aula. O nome do nó da trajetória é **lemniscatetrajectory.**

A trajetória foi criada com o objetivo de testar a capacidade do robô de seguir uma trajetória complexa e adaptar-se às variações nos parâmetros da curva.

O código em questão representa um loop principal em ROS que calcula e publica as posições desejadas e comandos de velocidade para um robô em uma trajetória em forma de lemniscata. Utilizando equações paramétricas, o código calcula a posição e orientação desejadas do robô na trajetória, além das velocidades lineares desejadas. Os valores calculados são então encapsulados em uma mensagem geometrymsgs::Twist e publicados no tópico twistpub. O loop assegura que esses cálculos e publicações ocorram continuamente, enquanto o código lida com as comunicações e eventos do ROS, garantindo a execução contínua e adequada do comportamento de controle do robô na trajetória especificada.

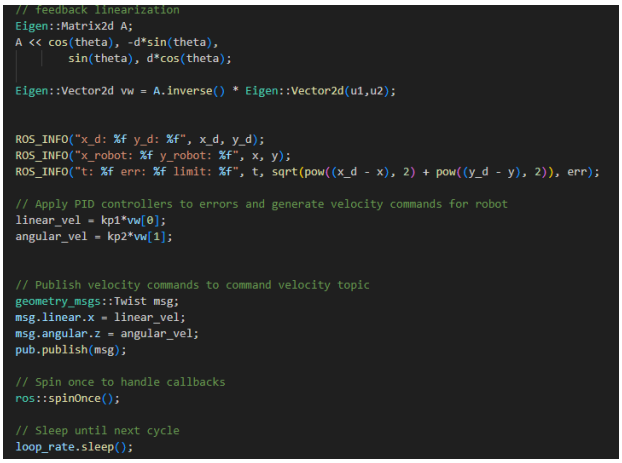
A lemniscata de Bernoulli é uma curva matemática descrita pela equação polar $r^2 = a^2 * cos(2\Theta)$, onde ”a”é um parâmetro. As equações paramétricas no código calculam a posição desejada (xd, yd) e a orientação desejada (thetad) do robô ao longo da lemniscata. Além disso, as velocidades lineares desejadas (vxd, vyd) são calculadas para garantir que o robô siga corretamente a trajetória. Os valores calculados são encapsulados em uma mensagem geometrymsgs::Twist e publicados no tópico apropriado para controlar o movimento do robô. O loop principal assegura que esses cálculos e publicações ocorram continuamente, permitindo que o robô siga a trajetória desejada de forma autônoma.

4.2 Controle do Robô

O loop principal inicia com o cálculo das posições desejadas (xd, yd) e a orientação desejada (thetad) do robô na lemniscata, utilizando equações paramétricas. Esses cálculos são essenciais para determinar a trajetória desejada que o robô deve seguir.

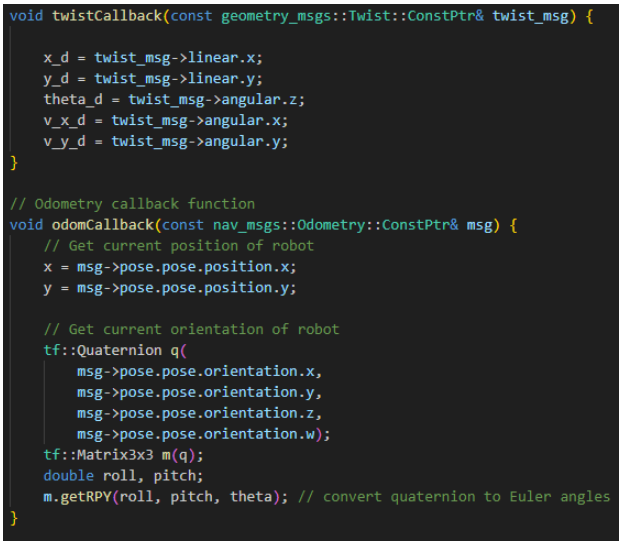
O loop principal inicia com o cálculo das posições desejadas (xd, yd) e a orientação desejada (thetad) do robô na lemniscata, utilizando equações paramétricas. Esses cálculos são essenciais para determinar a trajetória desejada que o robô deve seguir. Verifica-se se a velocidade total calculada é maior que um valor máximo (Vmax). Em caso afirmativo, as velocidades lineares e angulares são ajustadas para respeitar esse limite, garantindo que o robô não exceda as velocidades máximas permitidas.

Verifica-se se a velocidade total calculada é maior



```
// feedback linearization
Eigen::Matrix2d A;
A << cos(theta), -d*sin(theta),
     sin(theta), d*cos(theta);

Eigen::Vector2d vw = A.inverse() * Eigen::Vector2d(u1,u2);


ROS_INFO("x_d: %f y_d: %f", x_d, y_d);
ROS_INFO("x_robot: %f y_robot: %f", x, y);
ROS_INFO("t: %f err: %f limit: %f", t, sqrt(pow((x_d - x), 2) + pow((y_d - y), 2)), err);

// Apply PID controllers to errors and generate velocity commands for robot
linear_vel = kp1*vw[0];
angular_vel = kp2*vw[1];


// Publish velocity commands to command velocity topic
geometry_msgs::Twist msg;
msg.linear.x = linear_vel;
msg.angular.z = angular_vel;
pub.publish(msg);

// Spin once to handle callbacks
ros::spinOnce();

// Sleep until next cycle
loop_rate.sleep();
```

Figura 3: FeedBack Linearization



```
void twistCallback(const geometry_msgs::Twist::ConstPtr& twist_msg) {

    x_d = twist_msg->linear.x;
    y_d = twist_msg->linear.y;
    theta_d = twist_msg->angular.z;
    v_x_d = twist_msg->angular.x;
    v_y_d = twist_msg->angular.y;
}

// Odometry callback function
void odomCallback(const nav_msgs::Odometry::ConstPtr& msg) {
    // Get current position of robot
    x = msg->pose.pose.position.x;
    y = msg->pose.pose.position.y;

    // Get current orientation of robot
    tf::Quaternion q(
        msg->pose.pose.orientation.x,
        msg->pose.pose.orientation.y,
        msg->pose.pose.orientation.z,
        msg->pose.pose.orientation.w);
    tf::Matrix3x3 m(q);
    double roll, pitch;
    m.getRPY(roll, pitch, theta); // convert quaternion to Euler angles
}
```

Figura 4: Funções de callbacks

que um valor máximo (Vmax). Em caso afirmativo, as velocidades lineares e angulares são ajustadas para respeitar esse limite, garantindo que o robô não exceda as velocidades máximas permitidas. Os controladores PID são aplicados aos erros calculados, e os comandos de velocidade resultantes são gerados para o robô. Esses comandos de velocidade são então publicados no tópico de velocidade, desencadeando o movimento do robô ao longo da trajetória desejada.

Durante cada iteração do loop, as callbacks são processadas para atualizar as informações de odometria e comandos de velocidade. Após o processamento das callbacks, o loop aguarda até o próximo ciclo, permitindo que o código responda a eventos externos de forma assíncrona.

5 Resultados e análises

Após a implementação do loop principal do código de controle de trajetória para o robô em uma lemniscata de Bernoulli, foram realizados testes e análises para avaliar o desempenho e a eficácia do algoritmo. Durante os testes iniciais, observou-se que, embora o código possa não executar corretamente na primeira tentativa, após algumas iterações, o mesmo funciona conforme o esperado

6 Conclusões

Com base nos testes realizados e nas análises efetuadas, foi possível constatar que o código de controle de trajetória demonstrou ser funcional e eficaz e que será bem útil para os demais trabalhos seguintes, desde que as considerações acima sejam levadas em conta e que o código seja executado em um ambiente devidamente configurado e preparado para a execução de tarefas de controle de robôs em ROS.

Referências