Instituto Tecnológico de Aeronáutica – ITA

Sistemas de Controle Contínuos e Discretos – CMC-12

Laboratório 2 – Projeto de Sistema de Controle para Robô Seguidor de Linha

Professor: Marcos Ricardo Omena de Albuquerque Maximo

1 de abril de 2022

Observação: por questões de compatibilidade, este laboratório deve ser feito utilizando Simulink R2021b.

1 Introdução

O objetivo desse laboratório é familiarizar o aluno com projeto, simulação e validação de uma malha de controle com realimentação de velocidade. Em específico, utilizar-se-á um sistema de controle para fazer um robô diferencial seguir uma linha até parar num determinado ponto nessa linha. Com isso, espera-se exemplificar também heurísticas necessárias para fazer um sistema de controle funcionar na prática.

Conforme visto em sala, considere um robô como o apresentado na Figura 1. Assumindo controle sobre as velocidades linear e angular de um robô, o sistema dinâmico associado (modelo uniciclo) é

$$\begin{cases} \dot{x} = v\cos\psi, \\ \dot{y} = v\sin\psi, \\ \dot{\psi} = \omega, \end{cases} \tag{1}$$

em que x e y são coordenadas cartesianas e ψ é a orientação. Para controlar o sistema, é necessário ter informações para realimentação (malha fechada). No futebol de robôs, usa-se uma câmera acima do campo e algoritmos de visão computacional para determinar x, y e ψ. Como representado na Figura 1, os robôs possuem padrões coloridos que facilitam a detecção por visão computacional. Num carro autônomo, estados como esses são determinados tipicamente através da fusão de vários sensores, como GPS, LIDAR e câmeras embarcadas.

Esse sistema é não-linear e MIMO, de modo que não é possível controlá-lo nessa forma usando as técnicas estudadas até então. Porém, pode-se usar uma estratégia conhecida, de fazer o robô seguir uma linha.

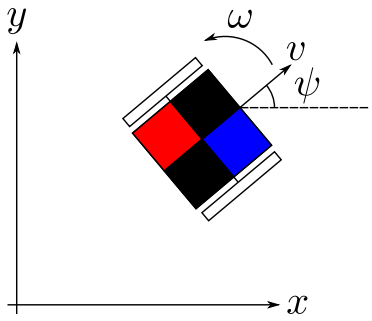

Figura 1: Modelo uniciclo para robô diferencial.

Considere um robô seguidor de linha como mostrado na Figura 2, para o qual pode-se escrever

$$\begin{cases} \dot{y} = v \sin \psi, \\ \dot{\psi} = \omega, \end{cases} \tag{2}$$

em que y é o desvio em relação à linha, v é a velocidade linear do robô e ψ é o ângulo em relação à linha. Considerando

$$\sin \psi \approx \psi, \tag{3}$$

obtém-se

$$\begin{cases} \dot{h} = v\psi, \\ \dot{\psi} = \omega, \end{cases} \tag{4}$$

que é um sistema linear. Com isso, pode-se usar duas malhas aninhadas (posição e ângulo) para fazer o robô seguir a linha:

$$\begin{cases} \psi_r = K_y \left(y_r - y\right), \\ \omega = K_\psi \left(\psi_r - \psi\right), \end{cases} \tag{5}$$

em que ψ_r é uma referência de ângulo calculada pela malha externa de posição, K_y é um ganho de posição e K_ψ é um ganho de velocidade (relacionado ao controle do ângulo). Desse modo, a dinâmica do sistema em malha fechada fica

$$\dddot{y} + K_\psi \ddot{y} + K_y K_\psi v \dot{y} = K_y K_\psi v y_r, \;\; \Rightarrow G_l(s) = \frac{K_y K_\psi v}{s^2 + K_\psi s + K_y K_\psi v}, \tag{6}$$

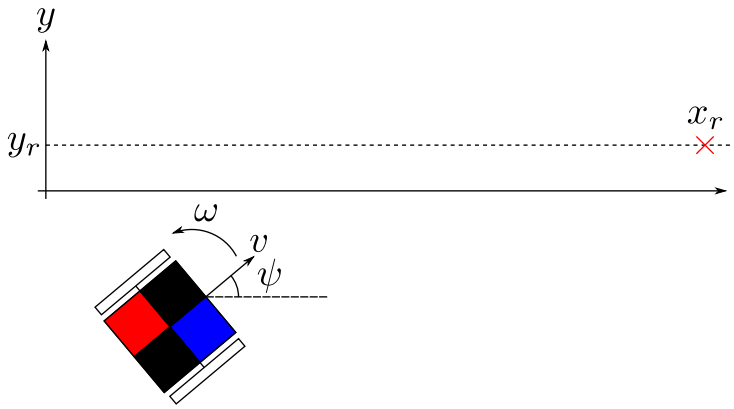

Figura 2: Robô seguidor de linha.

em que $G_l(s)$ é a função de transferência do sistema em malha fechada, aqui chamado de **controlador lateral**. Portanto, para que a dinâmica em malha fechada tenha frequência natural ω_n e fator de amortecimento ξ, deve-se escolher os ganhos como

$$K_y = \frac{\omega_n}{2\xi v}, \tag{7}$$

$$K_\psi = 2\xi\omega_n. \tag{8}$$

A estratégia de controle explica até então faz com que o robô se mantenha seguindo a linha indefinidamente. Porém, em muitas aplicações é interessante fazer com que o robô pare em algum ponto da linha. Por exemplo, no caso do futebol de robôs, isso é usado na estratégia do goleiro: o robô segue a linha do gol até a posição que ele espera que a bola a cruze, de modo a bloquear bolas lançadas pelo adversário.

Para permitir que o robô pare em um ponto, usa-se um **controlador frontal** que roda em paralelo com o controlador lateral. Considere a dinâmica frontal como

$$\dot{x} = v\cos\psi, \tag{9}$$

logo para ângulo pequenos, pode-se adotar $\cos\psi = 1$, de modo que a dinâmica torna-se

$$\dot{x} = v. \tag{10}$$

Como uma dinâmica de primeira ordem, uma lei de controle do tipo P é suficiente. Desse modo, a dinâmica em malha fechada é dada por

$$\begin{cases} v = K_x\left(x_r - x\right), \\ \dot{x} = v. \end{cases} \Rightarrow \dot{x} + K_x x = K_x x_r \Rightarrow G_f(s) = \frac{K_x}{s + K_x}. \tag{11}$$

em que K_x é um ganho proporcional. Assim, para que essa dinâmica tenha constante de tempo τ, deve-se usar

$$K_x = \frac{1}{\tau}. \tag{12}$$

Perceba que o controlador frontal altera v, que sua vez faz parte da dinâmica em malha fechada do controlador lateral, conforme mostra (6). Portanto, para manter os valores de ω_n e ξ desejados na malha lateral, deve-se "escalonar" o ganho K_y de acordo com v, i.e. deve-se recalcular K_y segundo (7) a cada instante. Além disso, outras heurísticas são necessárias para que a estratégia de controle proposta funcione na prática, as quais serão discutidas adiante neste roteiro.

2 Tarefas

2.1 Projeto do Controlador Frontal

Inicialmente, implementar-se-á a simulação do controlador frontal, por ser mais simples. Para facilitar, foram fornecidos *script script* `inicializarFrontal.m` e o modelo de simulação Simulink `frontal.slx`.

O *script* `inicializarFrontal.m` inicializa variáveis necessárias para a simulação e já está implementado, porém recomenda-se analisar o código para entender as variáveis

envolvidas. Perceba que foi escolhida uma constante de tempo $\tau = 0,15\ s$ para a dinâmica em malha fechada. Você deve implementar a simulação usando Simulink no arquivo `frontal.slx`, que deve ser entregue. No arquivo entregue como *template*, já há os blocos necessários, os quais inclusive já estão corretamente configurados. Quanto a isso, sugere-se que o aluno analise as propriedades de cada bloco (dois cliques em cima do bloco) para entender melhor como as variáveis definidas em `inicializarFrontal.m` se relacionam com o modelo Simulink.

Verifique se o resultado da simulação condiz com o previsto pela teoria e discuta no seu relatório os resultados. Inclua no seu relatório um gráfico $t \times x$ para embasar suas conclusões.

2.2 Projeto do Controlador Lateral

Nessa seção, deve-se implementar o controlador lateral. Para isso, foram fornecidos os arquivos:

- `inicializarLateral.m`: inicializa variáveis necessárias para a simulação. *Script* já implementado. Sugere-se lê-lo para entender quais são as variáveis relevantes e quais valores estão sendo utilizados.
- `lateral_linear.slx`: modelo de simulação Simulink para dinâmica lateral linearizada. Blocos fornecidos, mas é necessário montar o diagrama.
- `lateral_naolinear.slx`: modelo de simulação Simulink para dinâmica lateral não-linear (original). Blocos fornecidos, mas é necessário montar o diagrama.
- `avaliarLinearidadeLateral.m`: realiza simulações com diferentes condições iniciais e traça gráficos para comparar o modelo linear com o não-linear. *Script* já implementado.

Implemente os diagramas `lateral_linear.slx` e `lateral_naolinear.slx`. Aqui, deve-se lembrar que a aproximação feita em (3) para linearizar o sistema tem um domínio de validade associado à magnitude de ψ. Embora a aproximação tenha bom resultado apenas para ângulos pequenos, para o sistema de controle em questão, ela não gera grandes problemas para ângulo maiores (i.e. a dinâmica passa a não ser linear, porém o sistema de controle ainda funciona), a não ser que aconteça $|\psi| > \pi/2$. Para entender, lembre-se que

$$|\psi| > \frac{\pi}{2} \Rightarrow \cos\psi < 0, \tag{13}$$

logo, por (9), o robô move-se frontalmente no sentido contrário ao desejado. Na prática, quando há esse problema, o robô move-se em círculos e nunca converge para a linha. Isso acontece quando o robô está mais distante da linha, pois a malha externa comanda uma referência de ângulo ψ_r alta, possivelmente levando a $|\psi| > \pi/2$, de acordo com (5).

Uma solução heurística muito simples para solucionar esse problema é saturar o valor absoluto do ângulo de referência ψ_r até um máximo de $\psi_{r,max}$ para impor $|\psi| < \pi/2$. Devido à própria dinâmica da malha fechada e a erros de medida, é possível ψ passar do valor de ψ_r em algum momento, então é sensato manter uma margem de segurança. No caso, o valor usado em `inicializarLateral.m` é $\psi_{r,max} = 80°$. O limite de referência

de ângulo ser implementado apenas no modelo não-linear, pois saturação representa uma não-linearidade no modelo.

No relatório, não há necessidade de mostrar gráficos das simulações isoladamente para comprovar o funcionamento dos seus modelos Simulink. Basta incluir os gráficos gerados pelo `avaliarLinearidadeLateral.m`. Comente os resultados obtidos.

2.3 Projeto do Controlador Completo

Nessa seção, você usará os conhecimentos das seções anteriores para implementar a estratégia completa de controle do seguidor de linha. Para isso, foram fornecidos os seguintes arquivos:

- `inicializarFollowline.m`: inicializa variáveis necessárias para a simulação. *Script* já implementado. Sugere-se lê-lo para entender quais são as variáveis relevantes e quais valores estão sendo utilizados.
- `followline.slx`: modelo de simulação Simulink para dinâmica completa do robô diferencial. Blocos e diagrama fornecidos. Você deve implementar apenas o código do controlador na *MATLAB Function* `controller`.
- `avaliarFollowline.m`: avalia o funcionamento do seguidor de linha para diferentes linhas. *Script* já implementado.
- `animacaoFollowline.m`: mostra uma animação do robô seguidor de linha e gera um vídeo.

Embora a implementação do controlador completo fosse possível em Simulink, pede-se implementação usando *MATLAB Function* para que o aluno tenha experiência de como implementar uma lei de controle em código. Na verdade, para implementações de funções mais complexas, como é o caso desse controlador, é comum o uso de *MATLAB Function* no Simulink ao invés de blocos Simulink.

Além das heurísticas explicadas anteriormente, deve-se levar em conta na sua implementação:

- No escalonamento (adaptação) proposto para o ganho Ky a partir de v, que é apresentado em (7), v aparece no denominador, logo um pequeno valor de v faz com que K_y seja muito grande. Para evitar que isso se torne um problema, um truque simples envolve apenas limitar o valor de v usado no escalonamento por um valor mínimo v_{min}, de modo que

$$K_y = \begin{cases} \dfrac{K'_y}{v}, & |v| > v_{min}, \\ \dfrac{K'_y}{v_{min}\text{sign}(v)}, & |v| \leq v_{min}, \end{cases} \tag{14}$$

  em que $K'_y = \omega_n/(2\xi)$ representa a parte constante de K_y e $\text{sign}(\cdot)$ retorna o sinal do argumento. Implemente isso no seu controlador.

- Na dinâmica do robô implementada no `followline.slx`, considera-se os limites de velocidade que as rodas do robô conseguem desempenhar. Logo, para evitar comandar velocidades superiores às que o robô consegue executar, deve-se limitar $|v|$ superiormente por um limite v_{max}. Assim, após o cálculo de v através de (9), faça

$$v = \min\left(v_{max}, \max\left(-v_{max}, v\right)\right), \tag{15}$$

em que $\min(a, b)$ e $\max(a, b)$ retornam o mínimo e o máximo entre seus argumentos, respectivamente. Lembre-se de fazer essa saturação antes do escalonamento de K_y.

Sugere-se primeiramente testar sua implementação em alguns casos isolados, como no caso já configurado em `inicializarFollowline.m`. Também recomenda-se verificar a partir dos dados de simulação se os limites $\psi_{r,max}$ e v_{max} estão sendo respeitados pelo seu controlador. Quando estiver seguro da implementação, use `avaliarFollowline.m` para avaliar sua implementação. Esse *script* realiza simulações com diferentes retas de referência. Além disso, realiza uma bateria de simulações em que o robô deve se mover para frente até chegar ao ponto e outra em que ele deve se mover para trás. Inclua os gráficos no seu relatório gerados pelo `avaliarFollowline.m` e comente os resultados obtidos.

Finalmente, execute uma simulação com os parâmetros de `inicializarFollowline.m` e então execute `animacaoFollowline(out)` para gerar uma animação do robô seguidor de linha. Inclua o vídeo gerado `followline.avi` na sua entrega. O interessante do vídeo é que o tempo decorre em 1x, de modo que é possível perceber o quão rápido o robô é.

3 Discussões Extras (Apenas Leitura)

3.1 Uso de Heurísticas em Controle

Perceba que além de controle possuir uma teoria matemática sólida, baseada em teoria de sistemas dinâmicos lineares, a aplicação de controle em sistemas reais geralmente envolve o uso de heurísticas, como as que foram discutidas neste laboratório. Nesse sentido, um engenheiro de controle competente tipicamente é capaz de mesclar bem teoria e prática. Destaca-se ainda que a recomendação é sempre procurar soluções baseadas em teoria matemática formal e recorrer às heurísticas apenas para questões não cobertas pela teoria. Para mitigar o risco introduzido pelas heurísticas e aproximações de modelo, costuma-se validar o projeto através de diversas simulações não-lineares e testes com o sistema real. Além disso, técnicas de controle mais avançadas permitem resolver muitas das limitações do controle clássico, de modo que se obtém soluções mais elegantes e com mais garantias formais.

3.2 Simulação da Dinâmica do Robô

Além de usar as equações do modelo uniciclo (1) para simular a dinâmica do robô, o modelo de simulação completo considera que as rodas possuem limites de velocidade angular nas rodas do robô. Perceba que na prática, num robô diferencial (com duas rodas), como o mostrado na Figura 1, controla-se ω_r e ω_l, que são as velocidades angulares das rodas direita e esquerda, respectivamente, e não diretamente v e ω. Entretanto, é

mais conveniente trabalhar com v e ω do que com ω_r e ω_l, por isso usa-se o modelo uniciclo no controlador.

Pela geometria da Figura 3, para converter de v e ω para ω_r e ω_l, faz-se

$$\begin{cases} v_r = \omega\left(R + \dfrac{l}{2}\right) = \omega_r r \\ v = \omega R \\ v_l = \omega\left(R - \dfrac{l}{2}\right) = \omega_l r \end{cases} \Rightarrow \begin{cases} \omega_r = \left(v + \dfrac{\omega l}{2}\right)\dfrac{1}{r} \\ \omega_l = \left(v - \dfrac{\omega l}{2}\right)\dfrac{1}{r} \end{cases}, \tag{16}$$

em que v_r e v_l são as velocidades lineares das rodas direita e esquerda, respectivamente. Já a conversão contrária é obtida através de

$$\begin{cases} v = \left(\dfrac{\omega_r + \omega_l}{2}\right) r, \\ \omega = \left(\dfrac{\omega_r - \omega_l}{l}\right) r. \end{cases} \tag{17}$$

Para simular a dinâmica do robô, o simulador completo converte os comandos de velocidade para velocidades das rodas usando (16), satura as velocidades angulares das rodas e então converte de volta para velocidades linear e angular através de (17) para uso no modelo uniciclo.

4 Instruções

- A primeira etapa do processo de correção consistirá em submeter as funções implementadas a vários casos de teste de forma automatizada. Assim, os cabeçalhos das funções devem ser seguidos **rigorosamente**. Arquivos `.m` com os nomes destas funções e os cabeçalhos já implementados foram fornecidos juntamente com este roteiro. Dê preferência a implementar seu laboratório a partir destes arquivos `.m` fornecidos para evitar erros.
- A entrega da solução desse laboratório consiste de arquivos de código (MATLAB e Simulink) e de um relatório (em `.pdf`), que devem ser submetidos no Google Classroom.

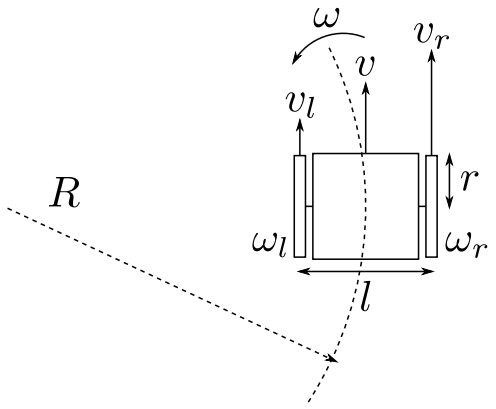

Figura 3: Robô diferencial realizando uma curva.

- Compacte todos os arquivos a serem submetidos em um único `.zip` (use obrigatoriamente `.zip`, e **não** outra tecnologia de compactação de arquivos) e anexe esse `.zip` no Google Classroom. Para o `.zip`, use o padrão de nome `<login_ga>_labX.zip`. Por exemplo, se seu login é `marcos.maximo` e você está entregando o laboratório 1, o nome do arquivo deve ser `marcos.maximo_lab1.zip`. **Não** crie subpastas, deixe todos os arquivos na "raiz" do `.zip`.

- O relatório deve ser sucinto, preocupe-se apenas em incluir discussões e entregáveis solicitados no roteiro. Pede-se apenas um cuidado mínimo na elaboração do relatório: responder adequadamente as perguntas, incluir figuras diretamente no relatório (ao invés de deixar como arquivos separados), figuras de boa qualidade, colocar nomes nos eixos dos gráficos, colocar legenda para diferenciar curvas num mesmo gráfico etc.

- **Não** é permitido o uso de funções ou comandos prontos do MATLAB que realizem toda a funcionalidade atribuída a uma certa função cuja implementação foi solicitada. Entretanto, o uso destas funções para verificação das implementações realizadas é encorajado. Em caso de dúvida, consulte o professor.

- A criação de *scripts* e funções auxiliares no MATLAB para execução de experimentos e geração de gráficos é fortemente recomendada para facilitar seu trabalho. Porém, não há necessidade de entregar código auxiliares.

- **Não** há necessidade de copiar e colar o código no seu relatório, dado que você também submeterá os arquivos de código. Também **não** há necessidade de explicar sua implementação no relatório, a **não** ser que o roteiro tenha solicitado explicitamente. Porém, organizar e comentar o código é muito salutar, pois ele será o foco da correção.

5 Dicas

- Nos modelos de simulação entregues, todos os blocos e os parâmetros de simulação já estão adequadamente configurados.

- Além dos blocos de Simulink vistos anteriormente, há os seguintes blocos novos nesse laboratório:

  (a) `Step`: simula uma função degrau.

  (b) `Subsystem`: cria um bloco de Simulink composto (subsistema) a partir de blocos mais simples.

  (c) `MATLAB Function`: permite programar uma função de MATLAB para ser usada no Simulink.

  (d) `Saturation`: satura o sinal de entrada para que permaneça entre um limite inferior e um superior.

  (e) `Product`: saída é produto das entradas.

(f) `Trigonometric Function`: realiza uma operação trigonométrica (seno, cosseno etc.).

- Para usar a saída do bloco `To Workspace` no formato `Structure with Time`:

  ```
  plot(out.x.time, out.x.signals.values)
  ```

- Caso queira chamar o Simulink dentro do MATLAB, use `out = sim('arquivo.slx')`. A saída do Simulink ficará na variável `out` nesse caso.