Universidade Federal do Espírito Santo

ELE08563 Robótica Móvel
Tarefas Simuladas

Alaf do Nascimento Santos
2017100781

2020/2 Earte

Conteúdo

1 Introdução

A robótica está presente em nossa vida por toda parte e, com o avanço tecnológico, os robôs tendem a substituir humanos em diferentes tarefas. Dentre tais tarefas, podemos citar casos de periculosidade envolvendo gases tóxicos ou radioatividade, bem como grande esforço físico ou trabalhos repetitivos. Apesar da sua necessidade e importância, a robótica ainda é bastante limitada, pois, por ser uma área que depende de diversas outras, esta passa por diversos problemas, muitos deles relacionados a mecânica, computação, telecomunicações ou controle; por exemplo, quando pensamos na robótica móvel, surgem novos problemas como a navegação autônoma de robôs.

Nesse contexto, utilizando-se de formações multi-robôs podemos executar determinadas tarefas que, se feitas com um único robô isolado, seriam mais difíceis ou até mesmo impossíveis. Essas formações multi-robôs podem ser do tipo homogênea ou heterogênea. No primeiro tipo, temos dois ou mais robôs iguais trabalhando em conjunto, enquanto que no segundo caso temos robôs diferentes, como por exemplo uma formação com um robô terrestre e um robô aéreo. Para cada objetivo a ser alcançado utilizando-se da robótica móvel, existe uma formação e um controle mais adequado a serem aplicados.

2 Objetivo

Este trabalho trata do desenvolvimento de códigos de simulações em MatLab para controle de duas formações homogêneas, começando com uma tarefa de posicionamento de dois robôs terrestres Pioneer 3-DX (uniciclo) e, por último, uma formação com dois robôs aéreos Parrot Bebop 2 (quadrimotor).

3 Estrutura do Programa

Os códigos gerados a partir da desenvolvimento deste trabalho, feito em MatLab, estão em anexo. Também encontramos os respectivos arquivos no repositório "mobile-robotics"disponível no meu GitHub. Para implementação das simulações das tarefas propostas, foi pensado em um programa estruturado em chamada de funções baseado na escolha do usuário. Aqui temos um programa principal, o qual é responsável pela chamada das simulações de tarefas a partir de entradas do teclado em uma interface gráfica. Em outras palavras, neste trabalho foi desenvolvido uma função 'main.m' a qual possui uma interface com o usuário para que seja possível decidir qual ou quais simulações executar e se pretenda visualizar a animação da mesma ou não, sendo possível se obter somente as respostas gráficas de saídas estáticas, caso assim desejado.

3.1 Função Principal

A função principal (main) trata da chamada das tarefas individualmente. Esta função toma a decisão sobre quais tarefas chamar, baseada na entrada do teclado que o usuário fará. A implementação da mesma está disponível nos anexos deste trabalho. Nesta etapa

inicial, o programa começa plotando na tela uma pequena janela com caixas de entrada de texto, nas quais o usuário deve entrar com 'S' ou 's' para as tarefas que deseja simular, qualquer outra entrada será interpretada como 'n' ou 'N' e não será executada a respectiva simulação. A entrada de teclado é armazenada e em seguida interpretada em estruturas condicionais do tipo "if", onde são chamadas as funções de tarefas (task1 e task2).

Em seguida é feito o armazenamento dos dados coletados na simulação em vetores que são retornados por cada função de tarefa e, em seguida, estes foram armazenados em arquivos próprios para abertura em MATLAB para gerar gráficos para análise dos resultados. Vale lembrar que as funções task1 e task2 são as implementações solicitadas na especificação deste trabalho e cada uma será tratada individualmente em seções contendo detalhes sobre sua implementação numérica, além da discussão das respostas obtidas como resultados das simulações.

4 Tarefa 1

A implementação da primeira tarefa se baseia na formação com dois robôs Pioneer 3-DX (uniciclo) ilustrada na Figura 1. Nesta etapa foi tratada da formação de dois robôs móveis em linha considerando que a posição da formação é definida como sendo a posição do robô 1 (robô mais a esquerda), portanto a posição (x_f,y_f) é a própria posição do veículo 1 da formação.

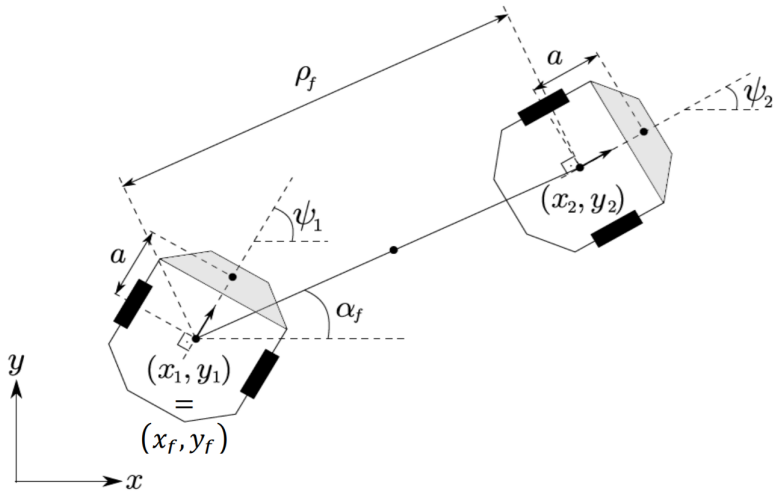

Figura 1: Formação Tarefa 1.

Foi projetado um controlador, para mover a formação, baseado no paradigma de estrutura virtual, usando as transformações do espaço dos robôs para o espaço da formação e vice-versa, com base no esquema multicamadas de controle. Considerou-se que as variáveis de formação são a distância entre os robôs, o ângulo de tal reta com o eixo horizontal e a posição do robô 1, ou seja, as variáveis (x_f,y_f,ρ_f,α_f) mostradas na Figura 1. Como o movimento da formação foi simulado, todas as variáveis de interesse estão disponíveis no código, o que equivale a assumir que todos os veículos da formação se comunicam com o computador onde o código estiver sendo executado.

A seguir temos uma descrição das equações que regem as transformações do paradigma utilizado. Foi obtida a matriz Jacobiana correspondente à transformação das variações $\dot{\mathbf{q}}_\mathbf{r}$ da formação para as velocidades $\dot{\mathbf{x}}_\mathbf{r}$ dos robôs. Por fim, foi usado um compensador dinâmico em cada um dos robôs.

4.1 Implementação Numérica

A implementação numérica começa basicamente com definições de variáveis de formação e inicialização de vetores necessários para modelar-se a estrutura multicamadas, representada na Figura 2.

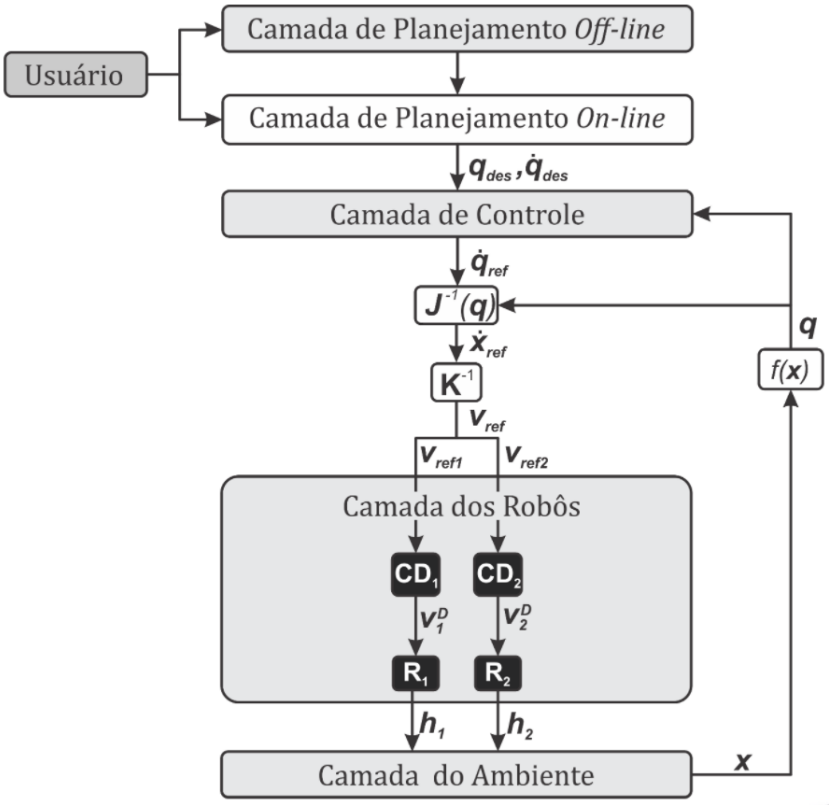

Figura 2: Esquema de multicamadas de controle.

Seguiu-se exatamente a sequência de camadas representadas no esquema de multicamadas de controle apresentado. Para tal, foram necessárias definições dadas na literatura, tais como a relação das variáveis de formação com as variáveis de posição de cada um dos robôs e vice-versa. A seguir temos a função $f(\mathbf{x})$ que transforma as posições dos veículos nas variáveis de formação, onde $\mathbf{x} = [x_1 \quad y_1 \quad x_2 \quad y_2]^T$.

$$f(x) = \mathbf{q} = \left[x_1 \quad y_1 \quad \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \right]^T$$

O caminho inverso, como de se esperar, é feito pela função $f^{-1}(\mathbf{q})$. Esta função transforma as variáveis da formação nas posições dos veículos. Sendo $\mathbf{q} = [x_f \quad y_f \quad \rho_f \quad \alpha_f]^T$, então $f^{-1}(\mathbf{q})$ é dada por:

$$f^{-1}(\mathbf{q}) = \mathbf{x} = [x_f \quad y_f \quad x_f + \rho_f \cdot \cos(\alpha_f) \quad y_f + \rho_f \cdot \sin(\alpha_f)]^T$$

Tendo-se calculado $\mathbf{q}$, é possível então calcular o erro referente ao seu valor atual em relação à posição desejada (alvo), para então calcular o $\dot{\mathbf{q}}_\mathbf{r}$ que será usada para determinar como cada robô irá se mover. Essa definição é dada por uma transformação do espaço da formação $\mathbf{q}$ para o espaço dos robôs $\mathbf{x}$. Tal transformação é dada pelo Jacobiano a ser apresentado. O trecho de código a seguir mostra exatamente como esta relação foi implementada na simulação:

```
q = [x1(counter+1) y1(counter+1) sqrt((x2(counter+1)
    - x1(counter+1))^2 + (y2(counter+1) - y1(counter+1))^2)
    atan((y2(counter+1) - y1(counter+1))/(x2(counter+1) - x1(counter+1)))]';
q_erro = q_des - q;
q_ref = L*tanh(inv(L)*k*q_erro);
```

No código acima, a variável counter é um contador de iterações e é utilizado na definição de tempo total da tarefa. Em seguida, após termos $\dot{\mathbf{q}}_\mathbf{r}$ definido, podemos passá-lo pela matriz Jacobiana inversa dada por:

$$J^{-1}(\mathbf{q}) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & \cos(\alpha_f) & -\rho_f \cdot \sin(\alpha_f) \\ 0 & 0 & \sin(\alpha_f) & \rho_f \cdot \cos(\alpha_f) \end{bmatrix}$$

Seguindo o esquema da Figura 2, foi necessário fazer a transformação em $J^{-1}(q)$ para então obter-se $\dot{\mathbf{x}}_\mathbf{r}$ que, em seguida, é transformado pela matriz K^{-1} que é dada por:

$$K^{-1}(\mathbf{q}) = \begin{bmatrix} \cos(\psi_1) & \sin(\psi_1) & 0 & 0 \\ -\sin(\psi_1)/\alpha_1 & \cos(\psi_1)/\alpha_1 & 0 & 0 \\ 0 & 0 & \cos(\psi_2) & \sin(\psi_2) \\ 0 & 0 & -\sin(\psi_2)/\alpha_2 & \cos(\psi_2)/\alpha_2 \end{bmatrix}$$

Esta matriz é na verdade um bloco diagonal que contém a cinemática inversa dos robôs e, como nossa formação é homogênea, ambas são iguais, distinguindo-se somente pelo fato da diferença de posição e postura de cada robô e, portando, obtendo-se variáveis de ψ_1 e ψ_2, sabendo que $\alpha_1 = \alpha_2 = \text{cte}$, podemos calcular o vetor $\mathbf{v}_\mathbf{r}$ que contém as velocidades lineares e angulares referência de cada robô. A seguir temos o código implementado que executa essas etapas citadas:

```
J_inv = [1 0 1 0;
         0 1 0 1;
         0 0 cos(a_f) (-rho_f*sin(a_f));
         0 0 sin(a_f) (rho_f*cos(a_f))]; %jacobiano

x_ref = J_inv*q_ref;

K_inv = [cos(psi1) sin(psi1) 0 0;
         (-sin(psi1)/a1) (cos(psi1)/a1) 0 0;
         0 0 cos(psi2) sin(psi2);
         0 0 (-sin(psi2)/a2) (cos(psi2)/a2)];

vr = K_inv*x_ref; %contem as velocidades lineares e angulares refs
```

Em resumo, a partir daqui entramos na camada dos robôs e nela calcula-se a dinâmica e cinemática para cada robô no intuito de se obter os vetores $\mathbf{h_1}$ e $\mathbf{h_2}$. Trabalha-se com

um compensador dinâmico e todo o processo passa por um loop de iterações baseadas na norma do vetor de erros em relação a posição desejada. Como o erro nunca será nulo, trabalha-se com uma região de satisfação, onde para uma norma igual à 0.1 já é suficiente para parar a simulação. Como o controlador proposto é diretamente proporcional ao erro da formação, quanto mais distante do objetivo, maior será o sinal de controle e, quanto mais a formação se aproxima da posição desejada, o erro tende a diminuir, e assim o sinal de controle também. Isso é importante pois caso contrário poderíamos passar da posição desejada, devido à altas velocidades. O código completo está disponível nos anexos deste trabalho (Anexo A.2 Task 1).

4.2 Respostas do Sistema

Simulou-se o sistema de controle apresentado anteriormente e, com o intuito de validar o mesmo, foi executada a tarefa de posicionamento com poses iniciais dadas por $(x_1; y_1; \psi_1) = (0\ m; 0\ m; 0°)$ e $(x_2; y_2; \psi_2) = (1\ m; -2\ m; 0°)$. Buscou-se a posição final para formação de $(x_f; y_f; \rho_f; \alpha_f) = (2\ m; 3\ m; 2\ m; 0°)$. Para tal simulação, gerou-se vetores contendo valores de interesse para resposta do sistema e com eles foram obtidos os gráficos de trajetória percorrida pelos robôs, velocidades lineares e angulares, bem como os erros referentes a posição da formação no tempo.

Vale destacar que considerou-se o período de amostragem como sendo de 100 ms e, para o controlador projetado, a tarefa foi realizada em 41,8 segundos. Isso se dá também devido ao critério de parada das iterações selecionado, que neste caso foi considerada uma norma 0.1 do vetor de erro nas variáveis desejadas de formação final. Se desejássemos um critério ainda mais preciso, o tempo de execução da tarefa seria ainda maior, pois o erro tende a zero com o passar do tempo, porém esse tempo tenderá ao infinito para erro nulo.

Na Figura 7 tem-se o caminho percorrido pelos robôs até atingir o destino. Nela é possível ver que a formação realmente se posicionou corretamente dentro da margem de erro considerada aceitável.

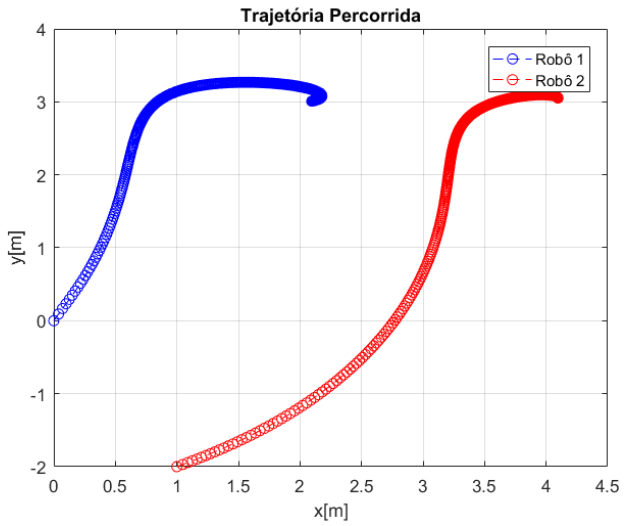

Figura 3: Trajetória dos robôs.

Sobre o erro relacionado as variáveis desejadas na formação final, temos que a Figura 8 pode ser confirmado que os erros da formação convergem assintoticamente para zero.

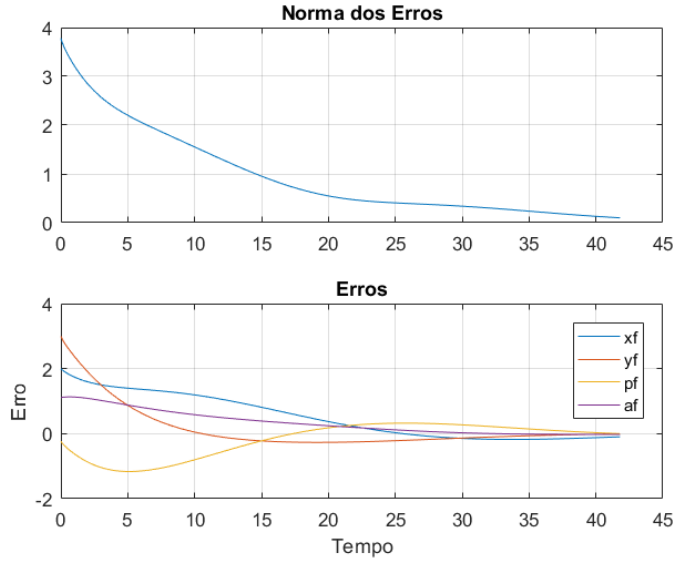

Figura 4: Erros em relação aos valores desejados.

Levantou-se também o gráfico das velocidades angulares e lineares de cada robô ao longo do tempo de trajeto. Essas curvas são dadas na Figura 9 onde é possível notar que as velocidades iniciais no geral são maiores que as outras desenvolvidas, pois no início o erro é alto e o sinal enviado pelo controlador é proporcional ao erro de formação. As velocidades seguem tendendo a zero a medida que os robôs se aproximam da posição desejada e, portanto, os sinais esperados foram alcançados com sucesso ao longo do percurso.

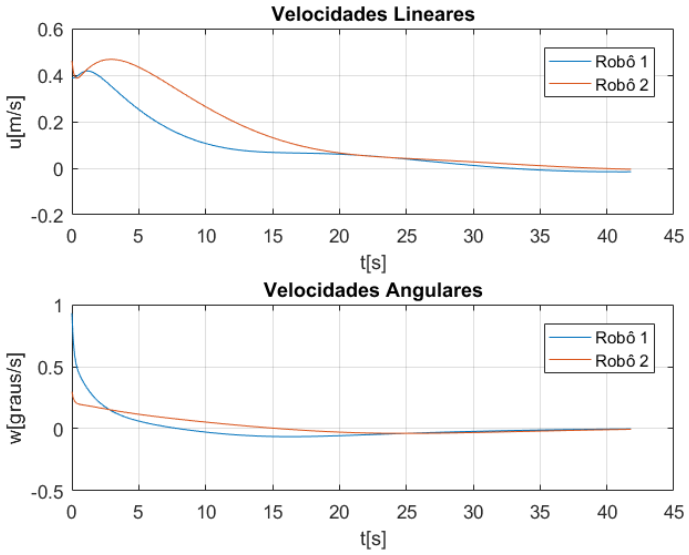

Figura 5: Velocidades lineares e angulares dos robôs.

5 Tarefa 2

A implementação da segunda tarefa se baseia na formação com dois robôs aéreos Parrot Bebop 2 (quadrimotor) ilustrada na Figura 6. Nesta etapa foi tratada da formação de dois robôs móveis em linha considerando que a posição da formação é definida como sendo a posição do robô 1 (robô mais a esquerda), portanto a posição (x_f,y_f) é a própria posição do veículo 1 da formação.

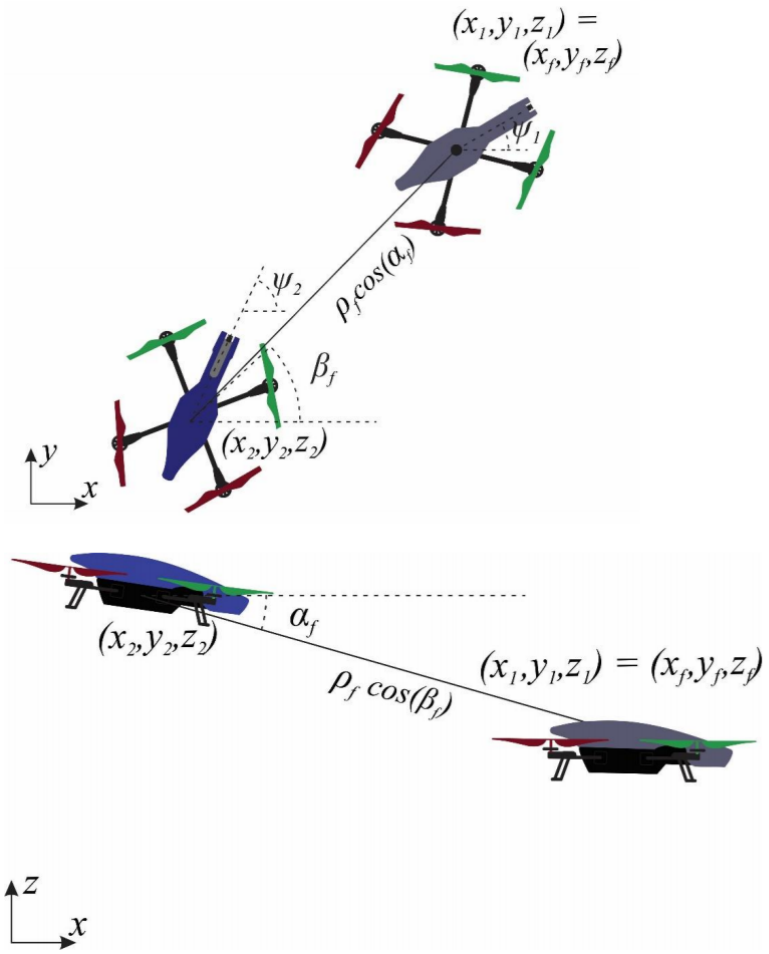

Figura 6: Formação Tarefa 2.

Foi projetado um controlador, para mover a formação, baseado no paradigma de estrutura virtual, usando as transformações do espaço dos robôs para o espaço da formação e vice-versa.

Considerou-se que as variáveis de formação são a distância entre os robôs, o ângulo de tal reta com o plano horizontal (XY), ou seja, α_f, o ângulo da projeção de tal reta no plano XY com o eixo X, ou seja, β_f, e a posição do robô 1, ou seja, as variáveis (x_f,y_f,z_f,ρ_f,α_f,β_f) mostradas na Figura 6. Como o movimento da formação foi simulado, todas as variáveis de interesse estão disponíveis no código, o que equivale a assumir que todos os veículos da formação se comunicam com o computador onde o código estiver sendo

executado. A seguir temos uma descrição das equações que regem as transformações da abordagem utilizada. Foi obtida a matriz Jacobiana correspondente à transformação de variações temporais do espaço da formação para o espaço dos robôs.

5.1 Implementação Numérica

Analogamente a implementação feita na tarefa 1, aqui temos a mesma estrutura de controle baseada em camadas. Por fins de simplicidade, etapas iguais não serão reexplicadas. Como no caso anterior a implementação numérica começa basicamente com definições de variáveis de formação e inicialização de vetores necessários para modelar-se a estrutura multicamadas.

Aqui temos algumas diferenças, pois estamos trabalhando com um sistema tridimensional e sem levarmos em consideração as orientações dos drones. Além disso, não trabalharemos com compensador dinâmico.

Nossa função $f(x)$ que transforma as posições dos robôs nas variáveis de formação fica da seguinte forma:

$$f(\mathbf{x}) = \mathbf{q} = \begin{bmatrix} x_f = x_1 \\ y_f = y_1 \\ z_f = z_1 \\ \rho_f = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \\ \alpha_f = \tan^{-1}\left(\dfrac{z_2 - z_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}\right) \\ \beta_f = \tan^{-1}\left(\dfrac{y_2 - y_1}{x_2 - x_1}\right) \end{bmatrix}$$

O vetor das variáveis de formação fica da seguinte forma neste modelo 3D:

$$\mathbf{q} = [x_f \quad y_f \quad z_f \quad \rho_f \quad \alpha_f \quad \beta_f]^T$$

A função $f^{-1}(q)$ que transforma as variáveis da formação nas posições dos robôs será dada por:

$$f^{-1}(\mathbf{q}) = \mathbf{x} = [x_f \quad y_f \quad z_f \quad x_f + \rho_f \cdot \cos(\alpha_f) \cdot \cos(\beta_f) \quad y_f + \rho_f \cdot \sin(\alpha_f) \cdot \cos(\beta_f) \quad z_f + \rho_f \cdot \sin(\beta_f)]^T$$

A matriz Jacobiana inversa é dada por:

$$J^{-1}(\mathbf{q}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & \cos(\alpha_f) \cdot \cos\beta_f & -\rho_f \cdot \sin(\alpha_f) \cdot \cos(\beta_f) & -\rho \cdot \cos(\alpha_f) \cdot \sin\beta_f \\ 0 & 1 & 0 & \sin(\alpha_f) \cdot \cos(\beta_f) & \rho_f \cdot \cos(\alpha_f) \cdot \cos(\beta_f) & -\rho_f \cdot \sin(\alpha_f) \cdot \sin\beta_f \\ 0 & 0 & 1 & \sin(\beta_f) & 0 & \rho_f \cdot \cos(\beta_f) \end{bmatrix}$$

A matriz K^{-1}, nosso bloco diagonal que contém a cinemática inversa dos robôs, fica da seguinte forma:

$$\mathbf{K^{-1}} = \begin{bmatrix} \mathbf{K_1^{-1}} & \mathbf{0_{4x4}} \\ \mathbf{0_{4x4}} & \mathbf{K_2^{-1}} \end{bmatrix} = \begin{bmatrix} \cos(\psi_1) & \text{sen}(\psi_1) & 0 & 0 & & & & \\ -\text{sen}(\psi_1) & \cos(\psi_1) & 0 & 0 & & \mathbf{0_{4x4}} & & \\ 0 & 0 & 1 & 0 & & & & \\ 0 & 0 & 0 & 1 & & & & \\ & & & & \cos(\psi_2) & \text{sen}(\psi_2) & 0 & 0 \\ & \mathbf{0_{4x4}} & & & -\text{sen}(\psi_2) & \cos(\psi_2) & 0 & 0 \\ & & & & 0 & 0 & 1 & 0 \\ & & & & 0 & 0 & 0 & 1 \end{bmatrix}$$

Diferente do caso explorado na tarefa 1, aqui não iremos usar os blocos de compensação dinâmica (CD1 e CD2). O processo passa por um loop de iterações baseadas na norma do vetor de erros em relação a posição desejada. Como o erro nunca será nulo, trabalha-se com uma região de satisfação, onde para uma norma igual à 0.1 já é suficiente para parar a simulação. Como o controlador proposto é diretamente proporcional ao erro da formação, quanto mais distante do objetivo, maior será o sinal de controle e, quanto mais a formação se aproxima da posição desejada, o erro tende a diminuir, e assim o sinal de controle também. Isso é importante pois caso contrário poderíamos passar da posição desejada, devido à altas velocidades. O código completo está disponível nos anexos deste trabalho (Anexo A.2 Task 2).

5.2 Respostas do Sistema

Simulou-se o sistema de controle apresentado anteriormente e, com o intuito de validar o mesmo, foi executada a tarefa de posicionamento com poses iniciais dadas por $(x_1; y_1; z_1\ \psi_1) = (0\ m; 0\ m; 0.75\ m; 0°)$ e $(x_2; y_2; \psi_2) = (-2\ m; 0\ m; 0.75\ m; 0°)$. Buscou-se a posição final para formação de $(x_f; y_f; z_f; \rho_f; \alpha_f; \beta_f) = (2\ m; 1\ m; 3\ m; 2\ m; 0°; 0°)$. Para tal simulação, gerou-se vetores contendo valores de interesse para resposta do sistema e com eles foram obtidos os gráficos de trajetória percorrida pelos robôs, velocidades lineares e angulares, bem como os erros referentes a posição da formação no tempo.

Vale destacar que considerou-se o período de amostragem como sendo de 200 ms e, para o controlador projetado, a tarefa foi realizada em 6,8 segundos. Isso se dá também devido ao critério de parada das iterações selecionado, que neste caso foi considerada uma norma 0.1 do vetor de erro nas variáveis desejadas de formação final. Se desejássemos um critério ainda mais preciso, o tempo de execução da tarefa seria ainda maior, pois o erro tende a zero com o passar do tempo, porém esse tempo tenderá ao infinito para erro nulo.

Na Figura 7 tem-se o caminho percorrido pelos robôs até atingir o destino. Nela é possível ver que a formação realmente se posicionou corretamente dentro da margem de erro considerada aceitável.

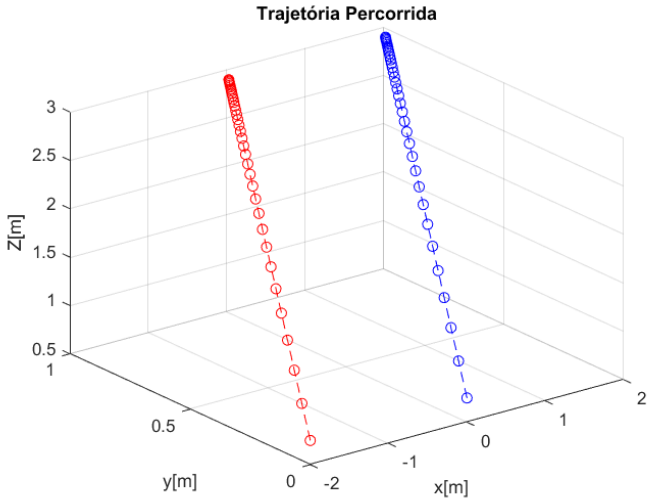

Figura 7: Trajetória dos robôs.

Sobre o erro relacionado as variáveis desejadas na formação final, temos que a Figura 8 pode ser confirmado que os erros da formação convergem assintoticamente para zero.

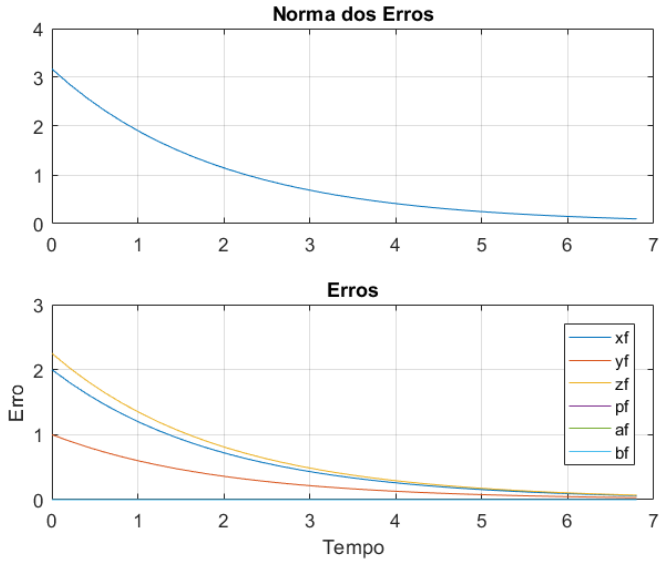

Figura 8: Erros em relação aos valores desejados.

Levantou-se também o gráfico das velocidades lineares de cada robô ao longo do tempo de trajeto. Essas curvas são dadas na Figura 9 onde é possível notar que as velocidades iniciais no geral são maiores que as outras desenvolvidas, pois no início o erro é alto e o sinal enviado pelo controlador é proporcional ao erro de formação. As velocidades seguem tendendo a zero a medida que os robôs se aproximam da posição desejada e, portanto, os sinais esperados foram alcançados com sucesso ao longo do percurso.

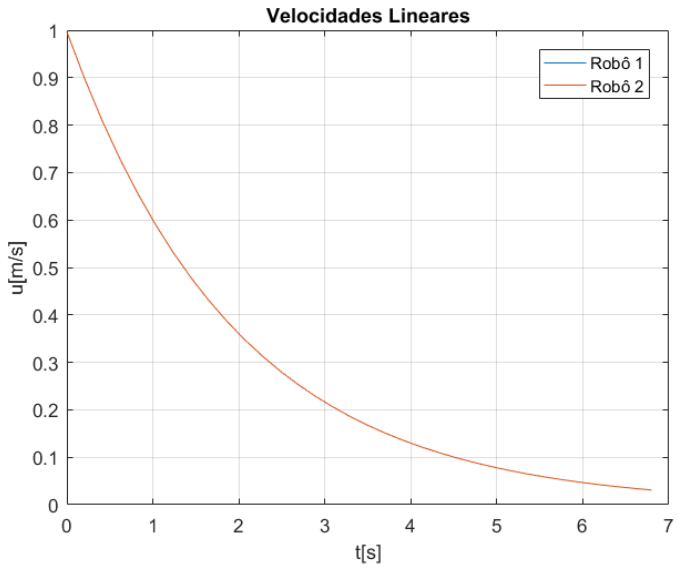

Figura 9: Velocidades lineares dos robôs.

6 Conclusões

Neste trabalho, assumiu-se que existe comunicação entre os robôs e um computador externo, onde o controle é executado. Desta forma, considerou-se que o computador recebe informação sobre a posição e orientação dos robôs e então envia os comandos de velocidade aos mesmos. Os resultados obtidos foram satisfatórios e comprovaram a eficiência do controle em camadas baseado em estrutura virtual. É de valia saber que na segunda formação, o caso de robôs aéreos, temos um sistema 3D que pode ser usado para o caso 2D também. Isso pode ser feito a partir da desconsideração do eixo z, bem como do ângulo vertical. Contudo, esta abordagem de robôs terrestres feita por meio de um modelo de simulação para robôs aéreos não é interessante, visto que a simulação 2D direta possui vantagens computacionais por trabalhar com menos variáveis e equações.

Referências Bibliográficas

BRANDÃO, Alexandre Santos et SARCINELLI-FILHO, Mário, *On the guidance of multiple uav using a centralized formation control scheme and delaunay triangulation*, Journal of Intelligent Robotic Systems, 2016, vol. 84, no 1, p. 397-413.

PESSIN, Gustavo. *Estratégias inteligentes aplicadas em robôs móveis autônomos e em coordenação de grupos de robôs*. 2013. Thèse de doctorat. Universidade de São Paulo.

RABELO, M. F. S., *Controle de Formações Homogêneas e Heterogêneas para Robôs Móveis*.

Sarcinelli-Filho, M. Robótica Móvel [material de sala de aula]. *ELE0863 Conjunto 11 - Controle de Formacao em Linha de Dois Robos*, Universidade Federal do Espírito Santo, Vitória, Brasil.

Sarcinelli-Filho, M. Robótica Móvel [material de sala de aula]. *ELE08563 Conjunto 3 - Robos Moveis a Rodas Tipo Uniciclo Seguimento de Trajetoria*, Universidade Federal do Espírito Santo, Vitória, Brasil.

A Código MatLab

A.1 Main Function

```
%Trabalho de Robótica Móvel (Simulação)
%Aluno: Alaf do Nascimento Santos
clear all, close all, clc;

%% INTERFACE COM USUARIO
%titulos das caixas de texto editável
prompt = {'Gostaria de simular a tarefa 1 com animação? S/N',
          'Gostaria de simular a tarefa 2? S/N'};
dims = [1 50]; %dimensão da caixa de texto
%variavel que guarda as entradas
entrada = inputdlg(prompt,'Escolha de Saídas',dims);
R1 = double(entrada{1});
R2 = double(entrada{2});

time = zeros(1,4);
norm_erro_vec = zeros(1,4);
xf_erro_vec = zeros(1,4);
yf_erro_vec = zeros(1,4);
veloc_lin_vec1 = zeros(1,4);
veloc_lin_vec2 = zeros(1,4);
veloc_ang_vec1 = zeros(1,4);
veloc_ang_vec2 = zeros(1,4);

animado1 = false;
animado2 = false;

if(R1 == 83 || R1 == 115) %S = 83, N = 78, s = 115, n = 110
    prompt = {'Quer a animação da tarefa 1? S/N'};
    dims = [1 50];
    %variavel que guarda as entradas
    entrada = inputdlg(prompt,'Escolha de Saídas',dims);
    Rx = double(entrada{1});
    %se animado = true, roda animação. Se animado = fase, roda sem animação.
    if(Rx == 83 || Rx == 115)
        animado1 = true;
    end
end

if(R2 == 83 || R2 == 115) %S = 83, N = 78, s = 115, n = 110
    prompt = {'Quer a animação da tarefa 2? S/N'};
```

```
    dims = [1 50];
    %variavel que guarda as entradas
    entrada = inputdlg(prompt,'Escolha de Saídas',dims);
    Rx = double(entrada{1});
    %se animado = true, roda animação. Se animado = fase, roda sem animação.
    if(Rx == 83 || Rx == 115)
        animado2 = true;
    end
end

if(R1 == 83 || R1 == 115) %S = 83, N = 78, s = 115, n = 110

    [x1,x2,y1,y2,time,norm_erro_vec,xf_erro_vec,yf_erro_vec,
    rho_f_erro_vec,a_f_erro_vec,veloc_lin_vec1,veloc_lin_vec2,
    veloc_ang_vec1,veloc_ang_vec2] = task1(animado1)

    tempo_total = time(end);

    fprintf('Tempo de Percurso: %i\n', tempo_total);
    fprintf('Posição Inicial - Robô 1: (%i,%i)\n', x1(1), y1(1));
    fprintf('Posição final - Robô 1: (%i,%i)\n', x1(end), y1(end));
    fprintf('Posição Inicial - Robô 2: (%i,%i)\n', x2(1), y2(1));
    fprintf('Posição final - Robô 2: (%i,%i)\n', x2(end), y2(end));

    figure('Name','Tarefa 1');
    plot(x1,y1,'b--o');
    hold on;
    plot(x2,y2, 'r--o');
    title('Trajetória Percorrida')
    legend('Robô 1','Robô 2');
    xlabel('x[m]')
    ylabel('y[m]')
    grid on

    figure('Name','Tarefa 1');
    subplot(2,1,1);
    plot(time,norm_erro_vec)
    grid on
    title('Norma dos Erros')
    subplot(2,1,2);
    plot(time,xf_erro_vec)
    hold on
    plot(time,yf_erro_vec)
    plot(time,rho_f_erro_vec)
```

```
    plot(time,a_f_erro_vec)
    title('Erros')
    legend('xf','yf','pf','af');
    xlabel('Tempo')
    ylabel('Erro')
    grid on

    figure('Name','Tarefa 1');
    subplot(2,1,1);
    plot(time,veloc_lin_vec1)
    hold on
    plot(time,veloc_lin_vec2)
    title('Velocidades Lineares');
    legend('Robô 1','Robô 2');
    xlabel('t[s]')
    ylabel('u[m/s]')
    grid on

    subplot(2,1,2);
    plot(time,veloc_ang_vec1)
    hold on
    plot(time,veloc_ang_vec2)
    title('Velocidades Angulares');
    legend('Robô 1','Robô 2');
    xlabel('t[s]')
    ylabel('w[graus/s]')
    grid on
end

if(R2 == 83 || R2 == 115) %S = 83, N = 78, s = 115, n = 110
    [x1,x2,y1,y2,z1,z2,time,norm_erro_vec,xf_erro_vec,
    yf_erro_vec,zf_erro_vec,rho_f_erro_vec,a_f_erro_vec,
    b_f_erro_vec,veloc_lin_vec1,veloc_lin_vec2] = task2(animado2);

    tempo_total = time(end);

    fprintf('Tempo de Percurso: %i\n', tempo_total);
    fprintf('Posição Inicial - Robô 1: (%i,%i,%i)\n', x1(1), y1(1),z1(1));
    fprintf('Posição final - Robô 1: (%i,%i,%i)\n', x1(end), y1(end),z1(end));
    fprintf('Posição Inicial - Robô 2: (%i,%i,%i)\n', x2(1), y2(1),z2(1));
    fprintf('Posição final - Robô 2: (%i,%i,%i)\n', x2(end), y2(end),z2(end));

    figure('Name','Tarefa 2');
    plot3(x1,y1,z1,'b--o');
```

```
    title('Trajetória Percorrida')
    xlabel('x[m]');
    ylabel('y[m]');
    zlabel('Z[m]')
    grid on
    hold on
    plot3(x2,y2,z2,'r--o');

    figure('Name','Tarefa 2');
    subplot(2,1,1);
    plot(time,norm_erro_vec)
    grid on
    title('Norma dos Erros')
    subplot(2,1,2);
    plot(time,xf_erro_vec)
    hold on
    plot(time,yf_erro_vec)
    plot(time,zf_erro_vec)
    plot(time,rho_f_erro_vec)
    plot(time,a_f_erro_vec)
    plot(time,b_f_erro_vec)
    title('Erros')
    legend('xf','yf','zf','pf','af','bf');
    xlabel('Tempo')
    ylabel('Erro')
    grid on

    figure('Name','Tarefa 2');
    plot(time,veloc_lin_vec1)
    hold on
    plot(time,veloc_lin_vec2)
    title('Velocidades Lineares');
    legend('Robô 1','Robô 2');
    xlabel('t[s]')
    ylabel('u[m/s]')
    grid on
end
```

A.2 Task 1

```
function [x1,x2,y1,y2,time,norm_erro_vec,xf_erro_vec, yf_erro_vec,
rho_f_erro_vec,a_f_erro_vec,veloc_lin_vec1,veloc_lin_vec2,
veloc_ang_vec1,veloc_ang_vec2] = task1(animado)
%TASK 1
```

```
%Formação com dois robôs Pioneer 3-DX (uniciclo).

%%%%%%%%%%%% Camada de planejamento e inicialização de variáveis %%%%%%%%%%%%
%variáveis de formação
xf = 2;
yf = 3;
rho_f = 2;
a_f = 0;

x1 = 0;
y1 = 0;
psi1 = 0;
a1 = 1;

x2 = 1;
y2 = -2;
psi2 = 0;
a2 = 1;

norm_erro_vec = [0 0 0];
xf_erro_vec = [0 0 0];
yf_erro_vec = [0 0 0];
rho_f_erro_vec = [0 0 0];
a_f_erro_vec = [0 0 0];

veloc_lin_vec1 = zeros(1,4);
veloc_ang_vec1 = zeros(1,4);
veloc_lin_vec2 = zeros(1,4);
veloc_ang_vec2 = zeros(1,4);

v = zeros(1,4);
vd = zeros(1,4);
vd_anterior = zeros(1,4);
vdp = zeros(1,4);

theta = [0.2604 0.2509 -0.000499 0.9965 0.00263 1.0768];

Ts = 100e-3; %O período de amostragem considerado deve ser de 100 ms

lu = 1; lw = 1;
ku = 0.2; kw = 0.2;
v1 = [100 100 100 100];
v2 = [0.15 0.15 0.15 0.15];
L = diag(v1);
```

```
k = diag(v2);

q_des = [xf yf rho_f a_f]'; %não varia no tempo, pois é tarefa de posicionamento

%%%%%%%%%%%%%%%%%%%% Camada de Controle %%%%%%%%%%%%%%%%%%%%
q_erro = q_des; %inicialização do erro
counter = 0;
%quanto menor a norma, mais proximo do ponto desejado e maior o tempo de percurso
while abs(norm(q_erro)) > 0.1
    %vai ser a realimentação
    q = [x1(counter+1) y1(counter+1)
        sqrt((x2(counter+1) - x1(counter+1))^2 + (y2(counter+1) - y1(counter+1))^2)
        atan((y2(counter+1) - y1(counter+1))/(x2(counter+1) - x1(counter+1)))]';
    q_erro = q_des - q; %deve decair para zero, mas nunca chegar a zero
    q_ref = L*tanh(inv(L)*k*q_erro);

    rho_f = q(3);
    a_f = q(4);
    J_inv = [1 0 1 0;
             0 1 0 1;
             0 0 cos(a_f) (-rho_f*sin(a_f));
             0 0 sin(a_f) (rho_f*cos(a_f))]; %jacobiano

    x_ref = J_inv*q_ref;

    K_inv = [cos(psi1) sin(psi1) 0 0;
             (-sin(psi1)/a1) (cos(psi1)/a1) 0 0;
             0 0 cos(psi2) sin(psi2);
             0 0 (-sin(psi2)/a2) (cos(psi2)/a2)];

    vr = K_inv*x_ref; %contem as velocidades lineares e angulares refs


%%%%%%%%%%%%%%%%%%%% Camada dos Robôs %%%%%%%%%%%%%%%%%%%%
%a partir daqui, preciso passar vref1 e vref2 pelo compensador dinâmico,

    %------------Calculando a dinamica para cada robô--------------
    vd = vr';
vdp = (vr' - vd_anterior)/Ts;
vd_anterior = vd;

    vd1 = [vd(1) vd(2)]'; vdp1 = [vdp(1) vdp(2)]'; v1 = [v(1) v(2)]';
    vd2 = [vd(3) vd(4)]'; vdp2 = [vdp(3) vdp(4)]'; v2 = [v(3) v(4)]';
```

```
v_erro1 = vd1 - v1;
F1 = [theta(4) 0;
          0 (theta(6) + (theta(5) - theta(3))*vd1(1))];
C1 = [0 -theta(3)*vd1(2);
          theta(3)*vd1(2) 0];
H1 = [theta(1) 0;
          0 theta(2)];
T1 = [lu 0;0 lw]*[tanh(ku*v_erro1(1)/lu);tanh(kw*v_erro1(2))];

    vr1 = T1 + H1*vdp1 + C1*vd1 + F1*vd1;


    v_erro2 = vd2 - v2;
F2 = [theta(4) 0;
          0 (theta(6) + (theta(5) - theta(3))*vd2(1))];
C2 = [0 -theta(3)*vd2(2);
          theta(3)*vd2(2) 0];
H2 = [theta(1) 0;
          0 theta(2)];
T2 = [lu 0;0 lw]*[tanh(ku*v_erro2(1)/lu);tanh(kw*v_erro2(2))];

    vr2 = T2 + H2*vdp2 + C2*vd2 + F2*vd2;

    u1 = v(1); ur1 = vr1(1); w1 = v(2); wr1 = vr1(2);
    u2 = v(3); ur2 = vr2(1); w2 = v(4); wr2 = vr2(2);

    %---------------Robô 1-----------
    vp1 = [1/theta(1) 0;
           0 1/theta(2)];

    vp1 = vp1*[ur1;
               wr1];

    vp1 = vp1 + [theta(3)/theta(1)*w1^2 - theta(4)/theta(1)*u1;
                 -theta(5)/theta(2)*u1*w1 - theta(6)/theta(2)*w1];
    %---------------Robô 2-----------
    vp2 = [1/theta(1) 0;
           0 1/theta(2)];

    vp2 = vp2*[ur2;
               wr2];

    vp2 = vp2 + [theta(3)/theta(1)*w2^2 - theta(4)/theta(1)*u2;
                 -theta(5)/theta(2)*u2*w2 - theta(6)/theta(2)*w2];
```

```
    %------------Calculando a cinematica para cada robô--------------

    %---------------Robô 1-----------
    v1 = vp1*Ts + v(1:2)';
    u1 = v1(1); w1 = v1(2);
    h1 = [u1*cos(psi1)-a1*w1*sin(psi1);
           u1*sin(psi1)+a1*w1*cos(psi1)];

   %---------------Robô 2-----------
    v2 = vp2*Ts + v(3:4)';
    u2 = v2(1); w2 = v2(2);
    h2 = [u2*cos(psi2)-a2*w2*sin(psi2);
           u2*sin(psi2)+a2*w2*cos(psi2)];

%%%%%%%%%%%%%%%%%%%% Camada do Ambiente %%%%%%%%%%%%%%%%%%%%
    %atualizar as posições dos robôs
    %aqui usar Ts aqui para multiplicar nas velocidades e ver as novas
    %coordenadas (x,y,psi) de cada robo
psi1 = psi1 + w1*Ts;
psi2 = psi2 + w2*Ts;
x1(counter+2) = h1(1)*Ts + x1(counter+1);
y1(counter+2) = h1(2)*Ts + y1(counter+1);
x2(counter+2) = h2(1)*Ts + x2(counter+1);
y2(counter+2) = h2(2)*Ts + y2(counter+1);


    if (animado)
        plot(x1(counter+1),y1(counter+1),'b--o');
        grid on
        hold on;
        plot(x2(counter+1),y2(counter+1), 'r--o');
        title('Posicionamento de Robôs Uniciclo')
        legend('Robô 1','Robô 2');
        xlabel('x[m]')
        ylabel('y[m]')
        axis([0 4.5 -3 5]);
        pause(0.00001); %uma pausa pequena só para ver o movimento "fluido"
        hold off;
    end

    v = [v1' v2'];
    counter = counter + 1; %contador de iterações
```

```
    norm_erro_vec(counter) = abs(norm(q_erro));
    xf_erro_vec(counter) = q_erro(1);
    yf_erro_vec(counter) = q_erro(2);
    rho_f_erro_vec(counter) = q_erro(3);
    a_f_erro_vec(counter) = q_erro(4);

    veloc_lin_vec1(counter) = u1;
    veloc_ang_vec1(counter) = w1;
    veloc_lin_vec2(counter) = u2;
    veloc_ang_vec2(counter) = w2;

end

%calculando tempo de percurso
tempo_total = counter*Ts;
time = linspace(0,tempo_total,counter);
end
```

A.3 Task 2

```
function [x1,x2,y1,y2,z1,z2,time,norm_erro_vec,xf_erro_vec,
yf_erro_vec,zf_erro_vec,rho_f_erro_vec,a_f_erro_vec,
b_f_erro_vec,veloc_lin_vec1,veloc_lin_vec2] = task2(animado)
%TASK 2
%Formação com dois robôs aéreos, do tipo quadrimotor.

%inicialização das variáveis
x1 = 0;
y1 = 0;
z1 = 0.75;
psi1 = 0;

x2 = -2;
y2 = 0;
z2 = 0.75;
psi2 = 0;

xf = 2;
yf = 1;
zf = 3;
pf = 2;
af = 0;
bf = 0;
```

```
k1 = 0.8417;
k2 = 0.18227;
k3 = 0.8354;
k4 = 0.17095;
k5 = 3.966;
k6 = 4.001;
k7 = 9.8524;
k8 = 4.7295;

Ku = diag([k1 k3 k5 k7]);
Kv = diag([k2 k4 k6 k8]);

Ts = 200e-3; %periodo de amostragem dado

v1 = 10*[1 1 1 1 1 1];
v2 = 10*[2 2 2 2 2 2];
L1 = diag(v1);
L2 = diag(v2);

q_des = [xf yf zf pf af bf]';
v = zeros(1,8);

%não varia no tempo, pois é tarefa de posicionamento
q_des = [xf yf zf pf af bf]';
q_erro = q_des; %inicialização do erro
q_erro_stop = [10 10 10];
counter = 0;

norm_erro_vec = [0 0 0];
xf_erro_vec = [0 0 0];
yf_erro_vec = [0 0 0];
zf_erro_vec = [0 0 0];
rho_f_erro_vec = [0 0 0];
a_f_erro_vec = [0 0 0];
b_f_erro_vec = [0 0 0];

veloc_lin_vec1 = zeros(1,4);
veloc_lin_vec2 = zeros(1,4);

while norm(abs(q_erro_stop)) > 0.1 %mudar esse criterio dps pra igual o da task1
q = [x1(counter+1) y1(counter+1) z1(counter+1)
sqrt((x2(counter+1)-x1(counter+1))^2 + (y2(counter+1)-y1(counter+1))^2
+ (z2(counter+1)-z1(counter+1))^2)
atan((z2(counter+1)-z1(counter+1))/sqrt((x2(counter+1)-x1(counter+1))^2 +
```

```
(y2(counter+1)-y1(counter+1))^2))
atan((y2(counter+1)-y1(counter+1))/(x2(counter+1)-x1(counter+1)))]';
    q_erro = q_des - q; %deve decair para zero, mas nunca chegar a zero
    q_ref = L1*tanh(inv(L2)*q_erro);

    q_erro_stop = q_erro(1:3);

    rho_f = q(4);
    a_f = q(5);
    b_f = q(6);

J_inv = [1 0 0 0 0 0;
             0 1 0 0 0 0;
             0 0 1 0 0 0;
             1 0 0 cos(af)*cos(bf) -pf*sin(af)*cos(bf) -pf*cos(af)*sin(bf);
             0 1 0 sin(af)*cos(bf) pf*cos(af)*cos(bf) -pf*sin(af)*sin(bf);
             0 0 1 sin(bf) 0 pf*cos(bf)]; %jacobiano

x_ref = J_inv*q_ref;
    x_ref = [x_ref(1:3)' 0 x_ref(4:6)' 0]';

    K_inv = [cos(psi1) sin(psi1) 0 0 0 0 0 0;
            -sin(psi1) cos(psi1) 0 0 0 0 0 0;
             0 0 1 0 0 0 0 0;
             0 0 0 1 0 0 0 0;
             0 0 0 0 cos(psi2) sin(psi2) 0 0;
             0 0 0 0 -sin(psi2) cos(psi2) 0 0;
             0 0 0 0 0 0 1 0;
             0 0 0 0 0 0 0 1];

    v_ref = K_inv*x_ref
v1 = v_ref(1:4);
v2 = v_ref(5:8);

   %M é função de psi1 e psi2, porém estes serão sempre zero e portanto gerarão M igu
    M = eye(4);

    h1 = M*v1;
    h2 = M*v2;

x1(counter+2) = h1(1)*Ts + x1(counter+1);
y1(counter+2) = h1(2)*Ts + y1(counter+1);
z1(counter+2) = h1(3)*Ts + z1(counter+1);
x2(counter+2) = h2(1)*Ts + x2(counter+1);
```

```
y2(counter+2) = h2(2)*Ts + y2(counter+1);
z2(counter+2) = h2(3)*Ts + z2(counter+1);

    if(animado)
        plot3(x1(counter+1),y1(counter+1),z1(counter+1),'b--*');
        xlabel('x[m]');
        ylabel('y[m]');
        zlabel('Z[m]');
        grid on
        hold on
        plot3(x1,y1,z1,'b');
        axis([-3 6 0 3 0 6]);
        plot3(x2(counter+1),y2(counter+1),z2(counter+1),'r--*');
        plot3(x2,y2,z2,'r');
        pause(Ts)
        hold off
    end

    v = [v1' v1'];

    counter = counter + 1; %contador de iterações

     norm_erro_vec(counter) = abs(norm(q_erro));
     xf_erro_vec(counter) = q_erro(1);
     yf_erro_vec(counter) = q_erro(2);
     zf_erro_vec(counter) = q_erro(3);
     rho_f_erro_vec(counter) = q_erro(4);
     a_f_erro_vec(counter) = q_erro(5);
     b_f_erro_vec(counter) = q_erro(6);

     veloc_lin_vec1(counter) = v1(1);
     veloc_lin_vec2(counter) = v2(1);

end
%calculando tempo de percurso
tempo_total = counter*Ts;
time = linspace(0,tempo_total,counter);
end
```