ISSN 2447-6102

Article

Modelagem Matemática de um Robô com Simulações Computacionais usando Programação Orientada a Objetos

Diehl, P.[1], Martinelli, L.[2] e Molter, A.[3]

[1] Universidade Federal de Pelotas; diehl.pedro@ufpel.edu.br
[2] Universidade Federal de Pelotas; lidia.martinelli@ufpel.edu.br
[3] Universidade Federal de Pelotas; alexandre.molter@ufpel.edu.br

Correspondência: diehl.pedro@ufpel.edu.br

Received: 18/01/2024; Accepted: 25/01/2024; Published: 31/01/2024

Abstract: The objective of the work is to implement inverse kinematics in a two-link planar robot with object-oriented programming. The article highlights relevant mathematical concepts in robotics, such as the rotation matrix and the equation of direct and inverse kinematics, applied in a manipulator robot with two degrees of freedom. Simulations of the robot's performance in the creation of graphics are presented.

Keywords: Robotics; Simulation; Inverse Kinematics; Object-oriented programming.

1. Introdução

A robótica é uma área multidisciplinar que envolve diversos campos do conhecimento, como a engenharia mecânica, elétrica, de controle e de computação.

Neste artigo, será apresentada uma aplicação de uso da equação de cinemática direta e das soluções de cinemática inversa em um robô manipulador com duas juntas de revolução, ou robô RR, que possui dois graus de liberdade permitindo movimentos de rotação ao redor de eixos específicos, como se fossem dois ``braços'', que podem se mover em diferentes direções. Esses dois graus de liberdade são comumente chamados de ``ombro'' e ``cotovelo'' e são usados para controlar o movimento do manipulador em um plano 2D (Spong et al., 2005). O primeiro grau de liberdade permite que o robô gire em torno de um eixo vertical (eixo do ombro), e o segundo grau de liberdade permite que o robô se mova para cima e para baixo ao longo de um eixo horizontal (eixo do cotovelo). Combinando esses dois movimentos, o robô pode alcançar diferentes pontos em um plano 2D (Siciliano et al., 2009).

O objetivo é implementar o robô do tipo RR em um programa computacional usando Programação Orientada a Objetos (POO). A modelagem matemática da cinemática e da cinemática inversa do robô está baseada no livro de Siciliano et al. (2009).

Como exemplos do robô em operação, serão realizadas simulações que mostram os movimentos das juntas do robô na execução de tarefas. Os códigos computacionais serão apresentados nos anexos 1 e 2.

2. Modelagem matemática da cinemática inversa do robô

A modelagem matemática da cinemática do robô foi desenvolvida de acordo com os textos de Siciliano et al. (2009) e Spong et al. (2005), cujas equações serão apresentadas na sequência.

2.1. Matriz de rotação

Uma matriz de rotação é uma matriz quadrada 3x3 que descreve uma rotação em torno de um determinado eixo em três dimensões. Essa matriz é uma representação numérica da transformação linear que ocorre quando um objeto

é rotacionado em relação a um ponto fixo (Lay, 2012). A matriz de rotação é definida pela combinação de senos e cossenos dos ângulos de rotação em torno dos três eixos cartesianos.

Consequentemente, a rotação de um ângulo γ em relação ao eixo z é:

$$R_z(\gamma) = \begin{bmatrix} cos\,\gamma & -sen\,\gamma & 0 \\ sen\,\gamma & cos\,\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (1)$$

De maneira semelhante, pode-se mostrar que as rotações de um ângulo β em relação ao eixo y, e de um ângulo α em relação ao eixo x são, respectivamente, dadas por:

$$R_Y(\beta) = \begin{bmatrix} cos\,\beta & 0 & sen\,\beta \\ 0 & 1 & 0 \\ -sen\,\beta & 0 & cos\,\beta \end{bmatrix}, \quad (2)$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & -sen\,\alpha \\ 0 & sen\,\alpha & cos\,\alpha \end{bmatrix}. \quad (3)$$

2.2. Cinemática inversa

O problema de cinemática inversa consiste na determinação das variáveis conjuntas correspondentes a uma dada posição e orientação do efetuador final de um robô. A solução deste problema é de fundamental importância para transformar as especificações de movimento, atribuídas ao efetuador final no espaço de trabalho, nos correspondentes movimentos do espaço articular, que permitam a execução do movimento desejado (Siciliano et al., 2009).

Por outro lado, o problema da cinemática inversa é muito mais complexo: as equações a resolver são em geral não lineares, pelo que nem sempre é possível encontrar uma solução de forma fechada, podendo existir múltiplas soluções (Spong et al., 2005).

No cálculo de soluções de forma fechada é usado o método geométrico para encontrar os pontos significativos na estrutura, em relação aos quais é conveniente expressar posição e/ou orientação como uma função de um número reduzido de incógnitas, ou método algébrico, para encontrar as equações significativas que contêm as incógnitas (Siciliano et al., 2009). Nesse artigo é utilizado o método algébrico.

Para solucionar a cinemática inversa de um braço planar, de dois elos, foram efetuadas as etapas que compreendem às equações (4) a (14).

A Figura 1 apresenta um esboço do manipulador planar considerado neste trabalho.

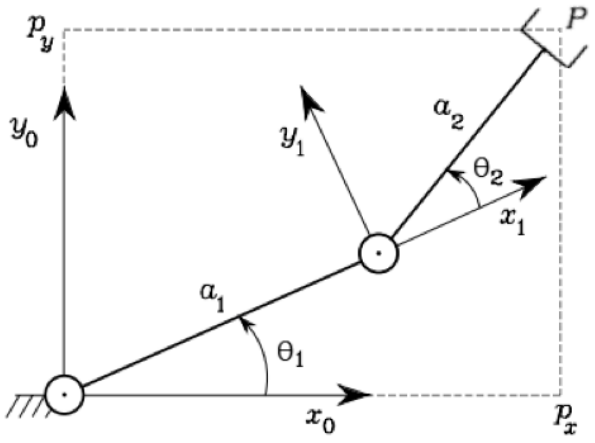

Figura 1: Esboço de um braço robótico com dois elos e um efetuador final na postura do cotovelo. Fonte: dos autores.

Na Tabela 1 estão descritos os significados dos símbolos utilizados no equacionamento e apresentados na Figura 1.

Tabela 1. Símbolos utilizados no problema da cinemática inversa.

Símbolo	Significado
θ_1	ângulo com o eixox_0
θ_2	ângulo entre o primeiro e segundo elo
ϕ	ângulo que representa a soma entre θ_1 e θ_2
a_1	comprimento do elo 1
a_2	comprimento do elo 2
p_x	Posição em x do efetuador final
p_y	Posição em y do efetuador final

Fonte: dos autores.

As equações que descrevem a posição do efetuador final do robô são dadas por:

$$\phi = \theta_1 + \theta_2, \tag{4}$$

$$p_x = a_1 \cos\theta_1 + a_2 \cos\phi, \tag{5}$$

$$p_y = a_1 \, sen\,\theta_1 + a_2 \, sen\,\phi. \tag{6}$$

Elevando (5) e (6) ao quadrado e somando-as, obtemos a relação:

$$p_x^2 + p_y^2 = a_1^2 + a_2^2 + 2a_1a_2cos\theta_2, \tag{7}$$

da qual, obtemos:

$$cos\theta_2 = \frac{p_x^2+p_y^2-a_1^2+a_2^2}{2a_1a_2}. \tag{8}$$

A existência de uma solução obriga a restrição:

$$-1 \leq cos\theta_2 \leq 1. \tag{9}$$

Caso contrário, o ponto informado estaria fora do espaço de trabalho alcançável do braço.

Usando a identidade $sen^2\theta_2 + cos^2\theta_2 = 1$ e substituindo o valor de $cos\theta_2$ obtido em (8), chegamos a equação:

$$sen\theta_2 = \pm\sqrt{1 - cos^2\theta_2}\,, \tag{10}$$

onde o sinal positivo da raiz está relacionado com a postura do cotovelo para baixo e o sinal negativo da raiz está relacionado com a postura do cotovelo para cima.

Das equações (8) e (10) podemos computar o ângulo θ_2 utilizando a função Atan2(x, y), a qual calcula o arco tangente da relação x/y, mas utiliza o sinal de cada argumento para determinar a qual quadrante o ângulo resultante pertence. Isso permite a determinação correta de um ângulo em uma faixa de 0 a 2π.

$$\theta_2 = Atan2(sen\theta_2, cos\theta_2). \tag{11}$$

Substituindo a equação (11) nas equações (5) e (6), obtemos os resultados:

$$sen\theta_1 = \frac{(a_1+a_2cos\theta_2)p_y-a_2sen\theta_2p_x}{p_x^2+p_y^2}, \tag{12}$$

$$cos\theta_1 = \frac{(a_1+a_2cos\theta_2)p_x-a_2sen\theta_2p_y}{p_x^2+p_y^2}. \tag{13}$$

Usando a função Atan2, calculamos θ_1, como:

$$\theta_1 = Atan2(sen\theta_1, cos\theta_1). \tag{14}$$

3. Programação Orientada a Objetos (POO)

Este modelo de programação se baseia na ideia de que um programa pode ser estruturado como um conjunto de objetos que interagem entre si para realizar uma tarefa específica (Meyer, 1997).

Cada objeto é uma instância de uma classe. A classe, por sua vez, é como um modelo que descreve as características que todos os objetos criados a partir dela terão em comum, como atributos e métodos. Atributos são as variáveis que armazenam informações sobre o objeto, enquanto métodos são as funções que definem o comportamento do objeto, ou seja, as ações que ele pode realizar.

Quando um objeto é criado a partir de uma classe, ele recebe automaticamente todos os atributos, que podem ser diferentes para cada, e métodos definidos na classe. Assim, os objetos podem ser pensados como instâncias únicas de uma classe, cada um com seu próprio estado e comportamento.

A utilização de objetos e classes na programação orientada a objetos permite uma maior organização e estruturação do código, além de possibilitar a reutilização de código e a criação de sistemas mais modulares e escaláveis. A Figura 2 mostra um fluxograma da organização de uma POO.

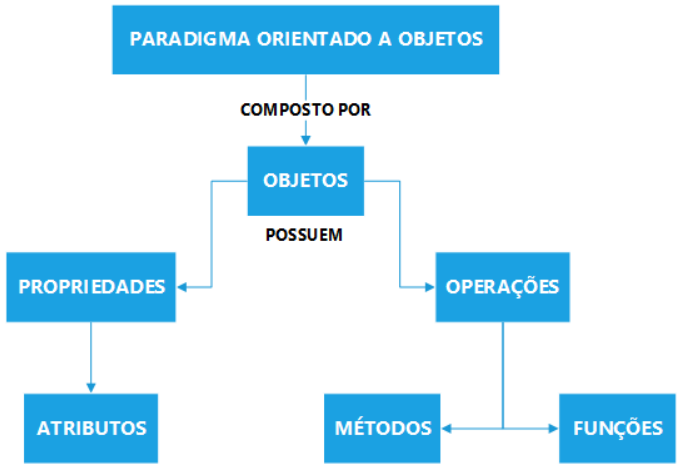

Figura 2: Diagrama exemplificando a programação orientada a objetos. Fonte: dos autores.

Uma POO possui como principais características a encapsulação - que permite que os dados e comportamentos de um objeto sejam escondidos de outros objetos, garantindo a segurança e a integridade do sistema - a herança - que permite que uma classe possa herdar as propriedades e comportamentos de outra classe, facilitando a reutilização de código e o desenvolvimento de sistemas mais complexos - e o polimorfismo - que permite que objetos de diferentes classes possam ser tratados de forma uniforme, tornando o código mais flexível e modular (Booch et al., 2001).

A POO é amplamente utilizada em linguagens de programação modernas, como Java, Python e C++, e é considerada uma das técnicas mais poderosas para desenvolver softwares complexos e de grande escala (Barnes, 2000).

4. Resultados e discussões

Para obter os resultados apresentados, utilizou-se as equações paramétricas para diversas figuras, como círculos, rosáceas, cardioides, entre outros. Após definir o intervalo dos pontos do conjunto de coordenadas polares que especificam a posição e a orientação de tais desenhos a serem realizados (iterou-se ponto a ponto através destes) foi calculada a cinemática inversa, e então, após mover os elos do manipulador, foi possível gerar as diferentes imagens que serão apresentadas nas Figuras 3, 4, 5, 6, 7 e 8. Os códigos-fonte para as implementações estão disponíveis nos Anexos 1 e 2.

Os gráficos das Figuras 3, 4, 5, 6, 7 e 8 apresentam as seguintes informações: (a) a representação dos dois elos que compõem o manipulador; (b) a trajetória do movimento total descrito pelo efetuador final, incluindo o deslocamento e o desenho; (c) o deslocamento do efetuador final sem a atividade de desenho; (d) o desenho final executado pelo efetuador.

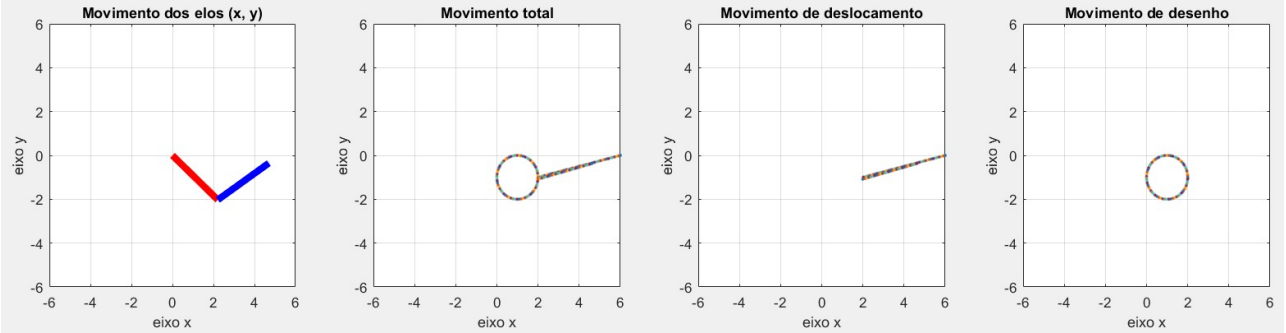

Figura 3: (a).(b). (c). (d). Fonte: dos autores.

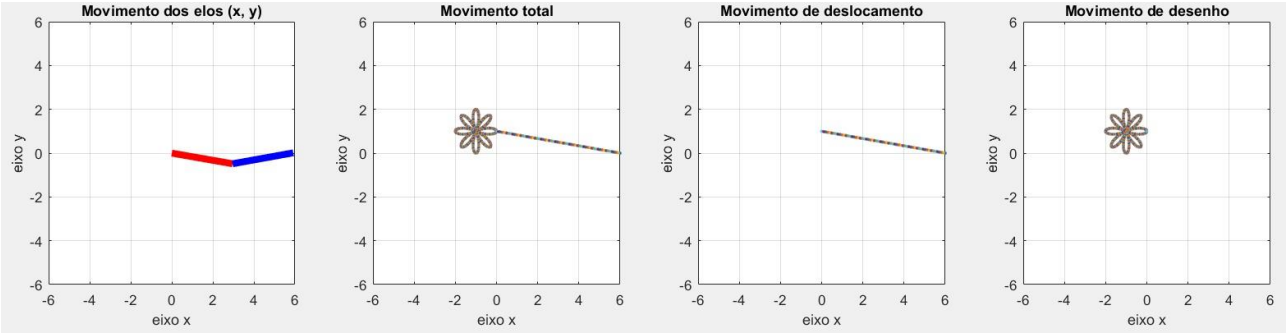

Figura 4: (a). (b). (c). (d). Fonte: dos autores.

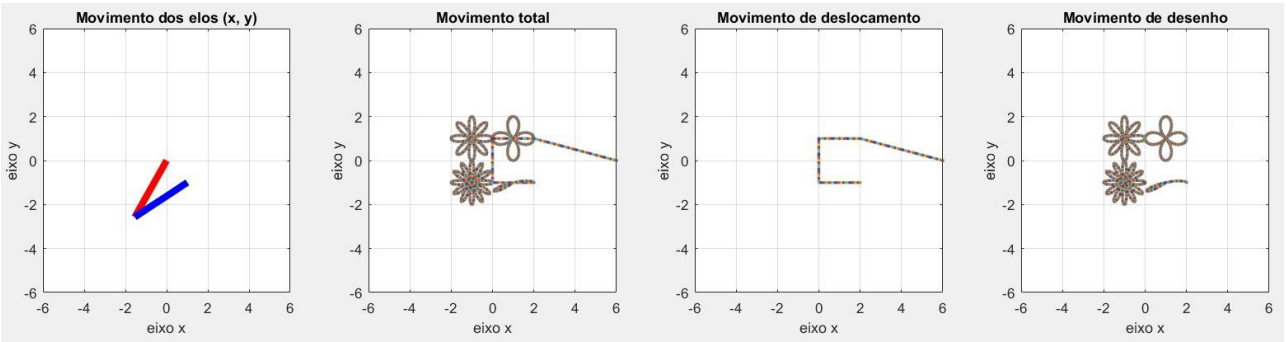

Figura 5: (a). (b). (c). (d). Fonte: dos autores.

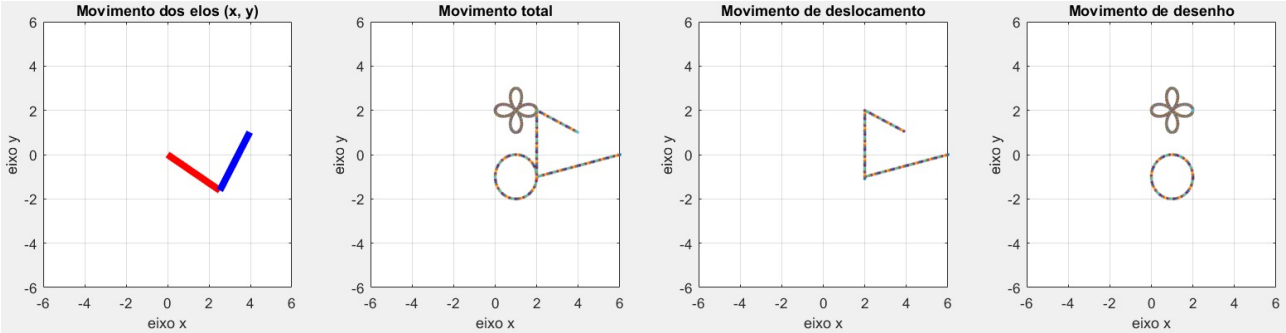

Figura 6: (a). (b). (c). (d). Fonte: dos autores.

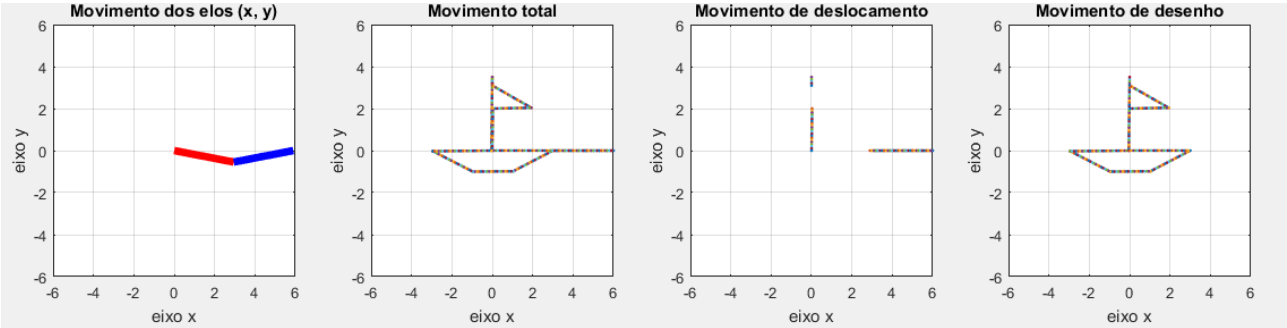

Figura 7: (a). (b). (c). (d). Fonte: dos autores.

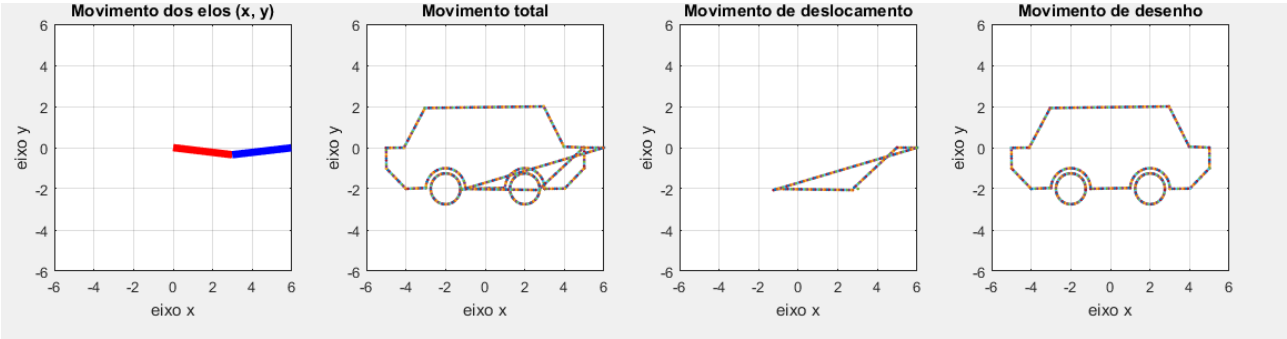

Figura 8: (a). (b). (c). (d). Fonte: dos autores.

5. Conclusões e perspectivas futuras

O presente trabalho teve como objetivo apresentar a aplicação da equação de cinemática inversa em um robô manipulador do tipo RR, que possui dois graus de liberdade para movimentos de rotação ao redor de eixos específicos denominados ``ombro'' e ``cotovelo''. Tais movimentos controlam o manipulador em um plano 2D, possibilitando alcançar diversos pontos nesse plano. Foi apresentado o equacionamento da cinemática inversa do robô e feita a implementação para um robô do tipo RR.

Concluiu-se que a implementação em *Matlab* foi bem-sucedida, uma vez que os resultados obtidos mostraram os desenhos de acordo com as instruções dadas, respeitando as leis da cinemática e as restrições da área de trabalho.

Os próximos passos a serem explorados no estudo de robótica compreendem a iniciação do estudo de dinâmica, a modelagem de outros tipos de robôs e a construção de um robô por meio da utilização de uma impressora 3D.

Agradecimentos: os autores agradecem pelo apoio da FAPERGS (Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul), que apoiou esta pesquisa através da bolsa de Iniciação Científica PROBIC.

Conflito de interesse: os autores declaram não haver conflito de interesse.

Referências

1. Barnes, D. Object-oriented programming with java: An introduction, 1st ed. Upper Saddle River, NJ: Pearson, 2000. ISBN: 978-0130869005.
2. Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Newkirk, J. W., Houston, K., e Conallen, J. Object-oriented analysis and design with applications, 3rd ed. Old Boston, MA: Addison Wesley, 2001. ISBN: 978-0201895513.
3. Lay, David C. Linear Algebra and Its Applications, 4th ed. Boston: Addison-Wesley, 2012. ISBN: 978-8521622093.
4. Meyer, B. Object-oriented software construction, 2nd ed. Old Tappan, NJ:Prentice Hall, 1997. ISBN: 978-0136291558.

5. Siciliano, B., Sciavicco, L., Villani, L., e Oriolo, G., “Robotics”. Em: Advanced Textbooks in Control and Signal Processing (2009), Springer, London.
6. Spong, M., Hutchinson, S., e Vidyasagar, M. Robot Modeling and Control. Nashville, TN.

Anexo 1

```
%Classe para definir o robô de dois elos (RR).
classdefRobotRR
properties
links % Propriedade para armazenar os elos do robô
end
  % CONSTRUTOR
methods
functionobj = RobotRR(a1, a2)
% Construtor que inicializa o robô com dois elos de comprimentos a1 e a2
obj.links = [Link(a1) Link(a2)];
end
end
  % METODOS
methods
  % CINEMÁTICA INVERSA
function [theta1, theta2] = inverseKinematics(obj, eePosition, angleUM, technique)
% Decompoe o vetor em x e y
px = eePosition(1, 1);
py = eePosition(2, 1);
%Confere se o ponto esta na area de trabalho do robo
if (sqrt(px^2 + py^2) > (obj.links(1).size + obj.links(2).size)) ||...
        (sqrt(px^2 + py^2) < (obj.links(1).size - obj.links(2).size))
error('The given end effector position is out side of the arm reachable space');
end
    switch technique
      case 'algebraic'
   %Calcula os valores trigonométricos para theta2
        c2 = (px^2 + py^2 - obj.links(1).size^2 - obj.links(2).size^2) / (2 * obj.links(1).size * obj.links(2).size);
% Confere se o ponto esta na area de trabalho do robo
if (c2 > 1) || (c2 < -1)
error('The given end effector position is out side of the arm reachable space');
end
        s2 = sqrt(1 - c2^2);
        ang_theta2 = atan2(s2, c2);
% Cacula os valores trigonométricos para theta1
        c1 = ((obj.links(1).size + obj.links(2).size * c2) * px + obj.links(2).size * s2 * py) / (px^2 + py^2);
        s1 = ((obj.links(1).size + obj.links(2).size * c2) * py - obj.links(2).size * s2 * px) / (px^2 + py^2);
        ang_theta1 = atan2(s1, c1);
      case 'geometric'
disp('TODO');
otherwise
error('Not valid technique informed');
end
    switch angleUM
      case 'rad'
        theta1 = ang_theta1;
        %theta1 = round(ang_theta1, 2);
        theta2 = ang_theta2;
        %theta2 = round(ang_theta2, 2);
      case 'deg'
        theta1 = rad2deg(ang_theta1);
       %theta1 = round(rad2deg(ang_theta1), 2);
        theta2 = rad2deg(ang_theta2);
       %theta2 = round(rad2deg(ang_theta2), 2);
otherwise
error('Not valid angle unit informed.');
end
end
    % MOVER MANIPULADOR
functionmoveManipulator(obj, pDesired)
startPos = obj.links(1).endPos + obj.links(2).endPos;
      % Vetor de distância do deslocamento
dist = pDesired - startPos;
   % Magnitude da hipotenusa para avancar sobre
hipMag = sqrt(dist(1)^2 + dist(2)^2 + dist(3)^2);
      for p = 0:0.1:hipMag
        % Decompoe a hipotenusa em x e y atraves do uso de cossenos e senos
        % Posicao atual + ou -, cosseno ou seno da posicao atual na hipotenusa
pDesloc = startPos + p * (dist / hipMag);
[theta1, theta2] = obj.inverseKinematics(pDesloc,...
                                         'deg',...
                                         'algebraic');
obj.links(1).updateEndPos(rM('z', theta1, 'deg'), theta1);
obj.links(2).updateEndPos(rM('z', (theta1 + theta2), 'deg'), theta2);
obj.plotArm();
obj.plotAllMovement();
subplot(2,4,3)
      %figure(3)
plot((obj.links(1).endPos(1) + obj.links(2).endPos(1)),...
  (obj.links(1).endPos(2) + obj.links(2).endPos(2)),...
          '.', 'linewidth', 0.0001);
```

```
        xlabel('eixo x')
        ylabel('eixo y')
         grid on
        hold on
        title('Movimento de deslocamento')
        axis([-6 6 -6 6])
        pause(5e-12);
        end
        end
% MOVER MANIPULADOR PARA A POSIÇÃO DE REPOUSO
        functionmoveManipulatorToRest(obj)
        obj.moveManipulator(obj.links(1).restPos + obj.links(2).restPos);
        end
            % GRAFICAR O BRAÇO DO ROBÔ
        functionplotArm(obj)
        % Método para visualizar o braço do robô
        subplot(2,4,1)
              %figure(1)
plot([0 obj.links(1).endPos(1)], [0 obj.links(1).endPos(2)], 'r',...
           [obj.links(1).endPos(1)  (obj.links(1).endPos(1) + obj.links(2).endPos(1))], [obj.links(1).endPos(2) (obj.links(1).endPos(2) + obj.links(2).endPos(2))], 'b',...
                  'linewidth', 5);
        xlabel('eixo x')
        ylabel('eixo y')
         grid on
        title('Movimento dos elos (x, y)')
        axis([-6 6 -6 6]);
         pause(5e-12)
        end
            % GRAFICAR DESENHO
        functionplotDraw(obj)
        subplot(2,4,4)
              %figure(2)
        plot((obj.links(1).endPos(1) + obj.links(2).endPos(1)),...
                    (obj.links(1).endPos(2) + obj.links(2).endPos(2)),...
                    '.', 'linewidth', 1e-12);
        xlabel('eixo x')
        ylabel('eixo y')
         grid on
        hold on
        title('Movimento de desenho')
        axis([-6 6 -6 6]);
         pause(5e-12)
        end
            % GRAFICAR TODO MOVIMENTO
        functionplotAllMovement(obj)
        subplot(2,4,2)
              %figure(4)
        plot((obj.links(1).endPos(1) + obj.links(2).endPos(1)),...
                    (obj.links(1).endPos(2) + obj.links(2).endPos(2)),...
                    '.', 'linewidth', 0.0001);
        xlabel('eixo x')
        ylabel('eixo y')
        grid on
        hold on
        title('Movimento total')
        axis([-6 6 -6 6]);
         pause(5e-12)
        end
            % DESENHAR LINHA
        functiondrawLine(obj, pDesired)
        startPos = obj.links(1).endPos + obj.links(2).endPos;
              % Vetor de distancia do deslocamento
        dist = pDesired - startPos;
              % Magnitude da hipotenusa para avancar sobre
        hipMag = sqrt(dist(1)^2 + dist(2)^2 + dist(3)^2);
              for p = 0:0.1:hipMag
                % Decompoe a hipotenusa em x e y atraves do uso de cossenos e senos
                % Posicao atual + ou -, cosseno ou seno da posicao atual na hipotenusa
        pDesloc = startPos + p * (dist / hipMag);
                [theta1, theta2] = obj.inverseKinematics(pDesloc,...
                                                         'deg',...
                                                         'algebraic');

        obj.links(1).updateEndPos(rM('z', theta1, 'deg'), theta1);
        obj.links(2).updateEndPos(rM('z', (theta1 + theta2), 'deg'), theta2);
        obj.plotArm();
        obj.plotAllMovement();
        obj.plotDraw();
         pause(5e-12);
        end
        end
            % DESENHAR CIRCULO
        functiondrawCircle(obj, xOffset, yOffset, radius, orientation)
                %{
                 Se 1 em orientation, direita para esquerda
                 Se -1 em orientation, esquerda para direita
                %}
                switch orientation
                  case -1
        startingPoint = 2 * pi;
        circlePoints = startingPoint:-0.1:0;
```

```
            case 1
        startingPoint = 0;
        circlePoints = startingPoint:0.1:(2 * pi);
        otherwise
        error('WrongorientationinformedtodrawCirclefunc
tion');
        end
            % Equacaoparametrica para o circulo
        p_circulo = [(xOffset + radius * cos(startingPoint));
                     (yOffset + radius * sin(startingPoint));
                                0];
        obj.moveManipulator(p_circulo);
        obj.plotAllMovement();
            for phi = circlePoints
            % Equacaoparametrica para o circulo
        p_circulo = [(xOffset + radius * cos(phi));
                         (yOffset + radius * sin(phi));
                                  0];
            [theta1,                theta2]                =
obj.inverseKinematics(p_circulo,...
                                          'deg',...
                                          'algebraic');
        obj.links(1).updateEndPos(rM('z',    theta1,    'deg'),
theta1);
        obj.links(2).updateEndPos(rM('z', (theta1 + theta2),
'deg'), theta2);
        obj.plotArm();
        obj.plotDraw();
        obj.plotAllMovement();
        end
        end
          % DESENHAR MEIO CIRCULO NA HORIZONTAL
        functiondrawHorizontalHalfCircle(obj,        xOffset,
yOffset, radius, orientation, upperOrLower)
            %{
             Se 1 em orientation, direita para esquerda
             Se -1 em orientation, esquerda para direita
            %}
            %{
Se 1 em upperOrLower, parte superior do semicirculo
Se -1 em upperOrLower, parte inferior do semicirculo
            %}
            switch orientation
              case -1
        startingPoint = pi;
                switch upperOrLower
                  case -1
        halfCirclePoints = startingPoint:0.1:(2 * pi);
                  case 1
        halfCirclePoints = startingPoint:-0.1:0;
        otherwise
        end
            case 1
        startingPoint = 0;
                switch upperOrLower
                  case -1
        halfCirclePoints = startingPoint:-0.1:-pi;
                  case 1
        halfCirclePoints = startingPoint:0.1:pi;
        otherwise
        error('WrongupperOrLowervalueinformedtodrawC
irclefunction');
        end
        otherwise
        error('WrongorientationvalueinformedtodrawCircl
efunction');
        end
            % Equacaoparametrica para o circulo
        p_circulo = [(xOffset + radius * cos(startingPoint));
                     (yOffset + radius * sin(startingPoint));
                                0];
        obj.moveManipulator(p_circulo);
        obj.plotAllMovement();
            for phi = halfCirclePoints
              % Equacaoparametrica para o circulo
        p_circulo = [(xOffset + radius * cos(phi));
                       (yOffset + radius * sin(phi));
                                0];
              [theta1,                theta2]                =
obj.inverseKinematics(p_circulo,...
                                          'deg',...
                                          'algebraic');

        obj.links(1).updateEndPos(rM('z',    theta1,    'deg'),
theta1);
        obj.links(2).updateEndPos(rM('z', (theta1 + theta2),
'deg'), theta2);
        obj.plotArm();
        obj.plotDraw();
        obj.plotAllMovement();
        end
        end
          % DESENHAR MEIO CIRCULO NA VERTICAL
        functiondrawVerticalHalfCircle(obj,          xOffset,
yOffset, radius, highOrLowStart, rightOrLeft)
            %{
Se 1 em highOrLowStart, inicio do desenho em cima
Se -1 em highOrLowStart, inicio do desenho embaixo
            %}
            %{
         Se 1 em rightOrLeft, parte direita do semicirculo
Se -1 em rightOrLeft, parte esquerda do semicirculo
            %}
            switch highOrLowStart
              case -1
        startingPoint = -pi / 2;
                switch rightOrLeft
                  case -1
```

```
    halfCirclePoints = startingPoint:-0.1:(-3 * pi / 2);
                    case 1
    halfCirclePoints = startingPoint:0.1:(pi / 2);
    otherwise
    error('WrongrightOrLeftvalueinformedtodrawCircl
efunction');
    end
            case 1
    startingPoint = pi / 2;
              switch rightOrLeft
                case -1
    halfCirclePoints = startingPoint:0.1:(3 * pi / 2);
                case 1
    halfCirclePoints = startingPoint:-0.1:(-pi / 2);
    otherwise
    error('WrongrightOrLeftvalueinformedtodrawCircl
efunction');
    end
    otherwise
    error('WrongorientationvalueinformedtodrawCircl
efunction');
    end
        % Equacaoparametrica para o círculo
    p_circulo = [(xOffset + radius * cos(startingPoint));
                 (yOffset + radius * sin(startingPoint));
                        0];
    obj.moveManipulator(p_circulo);
    obj.plotAllMovement();
        for phi = halfCirclePoints
          % Equacaoparametrica para o circulo
    p_circulo = [(xOffset + radius * cos(phi));
                    (yOffset + radius * sin(phi));
                          0];
            [theta1,              theta2]              =
obj.inverseKinematics(p_circulo,...
                                    'deg',...
                                    'algebraic');
obj.links(1).updateEndPos(rM('z', theta1, 'deg'), theta1);
obj.links(2).updateEndPos(rM('z',  (theta1  +  theta2),
'deg'), theta2);
    obj.plotArm();
    obj.plotDraw();
    obj.plotAllMovement();
    end
    end
      % DESENHAR ROSACEA
    functiondrawRose(obj, xOffset, yOffset, petalPairs,
orientation)
          %{
           Se 1 em orientation, direita para esquerda
           Se -1 em orientation, esquerda para direita
          %}
          switch orientation
            case -1
    startingPoint = 2 * pi;
    rosePoints = startingPoint:-0.01:0;
            case 1
    startingPoint = 0;
    rosePoints = startingPoint:0.01:(2 * pi);
    otherwise
error('WrongorientationinformedtodrawRosefunction');
    end
    r_rosa = cos(2 * petalPairs * startingPoint);
        % Equacao parametrica para a rosacea
    p_rosa = [xOffset + (r_rosa * cos(startingPoint));
    yOffset + (r_rosa * sin(startingPoint));
                    0];
    obj.moveManipulator(p_rosa);
    obj.plotAllMovement();
        for phi = rosePoints
    r_rosa = cos(2 * petalPairs * phi);
    p_rosa = [xOffset + (r_rosa * cos(phi));
    yOffset + (r_rosa * sin(phi));
                    0];
 [theta1, theta2] = obj.inverseKinematics(p_rosa,...
                                    'deg',...
                                    'algebraic');
obj.links(1).updateEndPos(rM('z', theta1, 'deg'), theta1);
obj.links(2).updateEndPos(rM('z',  (theta1  +  theta2),
'deg'), theta2);
    obj.plotArm();
    obj.plotDraw();
    obj.plotAllMovement();
    end
    end
      % DESENHAR CARDIOIDE
    functiondrawHorizontalCardioid(obj,          xOffset,
yOffset, circlesRadius, orientation, rightOrLeft)
          %{
           Se 1 em orientation, direita para esquerda
           Se -1 em orientation, esquerda para direita
          %}
          %{
Se 1 em rightOrLeft, cardioide tombada para direita
Se -1 em rightOrLeft, cardioide tombada para esquerda
          %}
          switch orientation
            case -1
    startingPoint = 2 * pi;
    cardioidPoints = startingPoint:-0.01:0;
            case 1
    startingPoint = 0;
    cardioidPoints = startingPoint:0.01:(2 * pi);
    otherwise
error('WrongorientationinformedtodrawRosefunction');
    end
    r_cardioid = 2 * circlesRadius * (1 - rightOrLeft *
cos(startingPoint));
```

```
        % Equacaoparametrica para a rosacea
    p_cardioid = [xOffset + (r_cardioid * cos(startingPoint));
    yOffset + (r_cardioid * sin(startingPoint));
                                0];
    obj.moveManipulator(p_cardioid);
    obj.plotAllMovement();
        for phi = cardioidPoints
    r_cardioid = 2 * circlesRadius * (1 - rightOrLeft * cos(phi));
    p_cardioid = [xOffset + (r_cardioid * cos(phi));
    yOffset + (r_cardioid * sin(phi));
                                0];
            [theta1, theta2] = obj.inverseKinematics(p_cardioid,...
                                'deg',...
                                'algebraic');
    obj.links(1).updateEndPos(rM('z', theta1, 'deg'), theta1);
    obj.links(2).updateEndPos(rM('z', (theta1 + theta2), 'deg'), theta2);
    obj.plotArm();
    obj.plotDraw();
    obj.plotAllMovement();
    end
    end
      % DESENHAR CARDIOIDE
    functiondrawVerticalCardioid(obj, xOffset, yOffset, circlesRadius, orientation, inverted)
        %{
        Se 1 em orientation, direita para esquerda
        Se -1 em orientation, esquerda para direita
        %}
        %{
        Se 1 em inverted, cardioide ponta cabeça
Se -1 em inverted, cardioide na orientacao vertical correta
        %}
        switch orientation
          case -1
    startingPoint = 2 * pi;
    cardioidPoints = startingPoint:-0.01:0;
          case 1
    startingPoint = 0;
    cardioidPoints = startingPoint:0.01:(2 * pi);
    otherwise
    error('WrongorientationinformedtodrawRosefunction');
    end
    r_cardioid = 2 * circlesRadius * (1 + inverted * sin(startingPoint));
        % Equacaoparametrica para a rosacea
    p_cardioid = [xOffset + (r_cardioid * cos(startingPoint));
    yOffset + (r_cardioid * sin(startingPoint));
                                0];
    obj.moveManipulator(p_cardioid);
    obj.plotAllMovement();
        for phi = cardioidPoints
r_cardioid = 2 * circlesRadius * (1 + inverted * sin(phi));
    p_cardioid = [xOffset + (r_cardioid * cos(phi));
    yOffset + (r_cardioid * sin(phi));
                                0];
 [theta1, theta2] = obj.inverseKinematics(p_cardioid,...
                                'deg',...
                                'algebraic');
obj.links(1).updateEndPos(rM('z', theta1, 'deg'), theta1);
    obj.links(2).updateEndPos(rM('z', (theta1 + theta2), 'deg'), theta2);
    obj.plotArm();
    obj.plotDraw();
    obj.plotAllMovement();
    end; end; end; end;
```

Anexo 2

```
    %Exemplo de uso com desenho de um carro
    format short
    clc, close all, clear all
    % Parametros dos elos do robo
    a1 = 3; % Comprimento do primeiroelo
    a2 = 3; % Comprimento do segundoelo
    % Criaumainstancia do robo
    robot = RobotRR(a1, a2);
    robot.moveManipulator([5; 0; 0]);
    robot.drawLine([5; -1; 0]);
    robot.drawLine([4; -2; 0]);
    robot.drawLine([3; -2; 0]);
    robot.drawHorizontalHalfCircle(2, -2, 1, 1, 1);
    robot.drawLine([-1; -2; 0]);
    robot.drawHorizontalHalfCircle(-2, -2, 1, 1, 1);
    robot.drawLine([-4; -2; 0]);
    robot.drawLine([-5; -1; 0]);
    robot.drawLine([-5; 0; 0]);
    robot.drawLine([-4; 0; 0]);
    robot.drawLine([-3; 2; 0]);
    robot.drawLine([3; 2; 0]);
    robot.drawLine([4; 0; 0]);
    robot.drawLine([5; 0; 0]);
    robot.drawCircle(2, -2, 0.75, 1);
    robot.drawCircle(-2, -2, 0.75, 1);
    robot.moveManipulatorToRest();
```