Slovenská Technická Univerzita – Fakulta Elektrotechniky a Informatiky

Ilkovičova 3, 812 19 Bratislava

Zadanie č.3

Mobilný kolesový robot

ROB

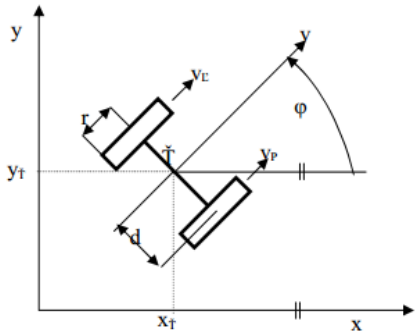
Filip Dubovský

2024

OBSAH

1. ZADANIE

Navrhnite a realizujte vizualizáciu diferenciálneho podvozku. Na tomto type zadania by ste si mali precvičiť implementáciu odvodených kinematických rovníc diferenciálneho podvozku a zafixovať tak preberané učivo.



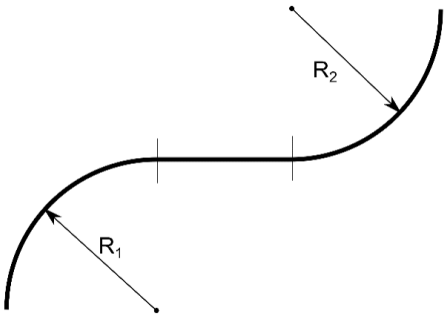
d = 0,1 m
r = 0,05 m

Parametre manipulátora:

L (rozchod kolies) = 200 [mm]
r (polomer kolesa) = 50 [mm]

V rámci riešenia zadania sa zamerajte na nasledovné úlohy:

1. Vykreslite trajektórie **ťažiska** a **kolies** (rôznymi farbami). Vstupným argumentom pre vykresľovanie budú vektory: času, rýchlostí ľavého, pravého kolesa. Majme napríklad takéto tri ľubovoľné vektory: časový (napr. t=[0 5 10 15 20], rýchlosť ľavého kolesa=[2 -1 0 2 1], rýchlosť pravého kolesa=[2 1 0 -2 1]). Vytvorte si tri takéto vektory s vlastnými hodnotami, ktoré vykreslia trajektórie kolies a ťažiska.
2. Vykreslite trajektóriu štvorec prostredníctvom ťažiska robota. Dovoľte užívateľovi definovať dĺžku strany štvorca a na základe toho vygenerujte príslušné časy a rýchlosti. Vykreslite aj trajektórie kolies.
3. Vykreslite trajektóriu krivka podľa obr. 2 prostredníctvom ťažiska robota. Dovoľte užívateľovi definovať R1, L1,R2 a na základe toho vygenerujte príslušné časy a rýchlosti. Vykreslite aj trajektórie kolies.
4. Vytvorte hru, kde pomocou šípok alebo W,A,S,D budete ovládať robota.



2. ROZBOR

Cieľom tohto projektu bolo realizovať vizualizáciu diferenciálneho podvozku pomocou kinematických rovníc.

2.1. Postup riešenia

2.1.1. Výpočet rýchlosti a uhlovej rýchlosti ťažiska

$$V_T = \frac{V_R + V_L}{2}$$

Rovnica 1 – Výpočet rýchlosti ťažiska

$$\omega_T = \frac{V_R - V_L}{L}$$

Rovnica 2 – Výpočet uhlovej rýchlosti ťažiska

2.1.2. Výpočet polohy a otočenia ťažiska

$$V_x = V_T \cos\varphi$$

$$V_y = V_T \sin\varphi$$

Rovnica 3 – Výpočet zložiek rýchlosti

$$\Delta x_T = V_x \cdot \Delta t$$

$$\Delta y_T = V_y \cdot \Delta t$$

Rovnica 4 – Výpočet prejdenej dráhy

$$\begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{matrix} x_T + \Delta x_T \\ y_T + \Delta y_T \end{matrix}$$

Rovnica 5 – Výpočet polohy

$$\Delta\varphi = \omega_T \cdot \Delta t$$

Rovnica 6 – Výpočet uhla prejdenej dráhy

$$\varphi = \varphi + \Delta\varphi$$

Rovnica 7 – Výpočet uhla

2.1.3. Výpočet rýchlostí na základe zadaného polomeru

Vychádzame zo vzorca:

$$R = \frac{L}{2} \cdot \frac{V_R + V_L}{V_R - V_L}$$

Rovnica 8 – Výpočet polomeru zo zadaných rýchlostí

Po upravení boli získané:

$$V_L = \pi L + \pi R$$

$$V_R = -\pi L + \pi R$$

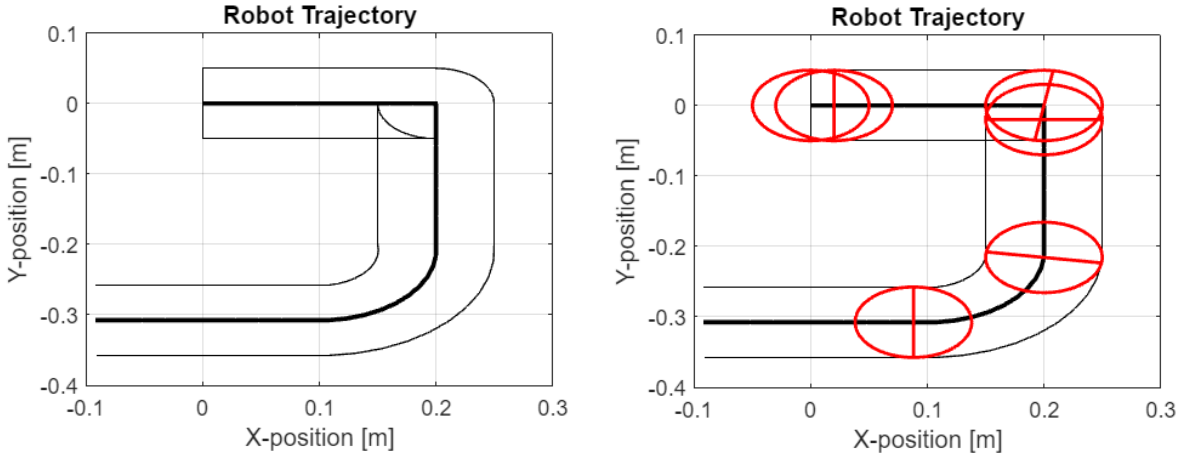
Rovnica 9 – Výpočet rýchlostí zo zadaného polomeru

3. ZHODNOTENIE

3.1. Vykreslenie trajektórie

Práca začala s jednoduchým vykreslením trajektórie pomocou vzorcov. Vstupné vektory rýchlosti pravého a ľavého kolesa slúžili ako podklad pre výpočty.

$$vl = [0 \quad 2 \quad \frac{\pi}{2} \quad 2 \quad \pi \quad 2]/10 \quad [\text{m/s}]$$
$$vr = [0 \quad 2 \quad -\frac{\pi}{2} \quad 2 \quad 0 \quad 2]/10 \quad [\text{m/s}]$$

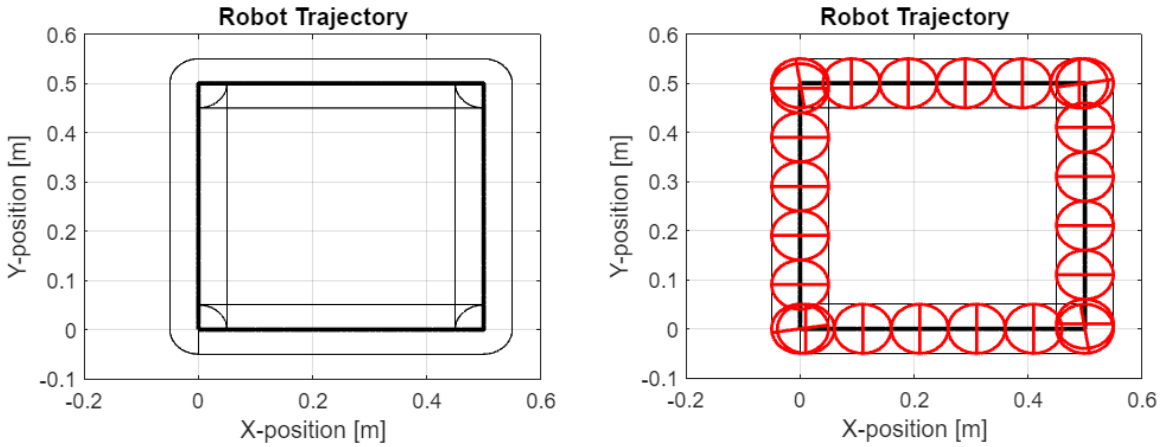


Obrázok 1 - Vykreslenie trajektórie

Na základe zadaného parametra K sa výpočty zhustili, čím sa dosiahlo spresnenie trajektórie. Následne bola vykreslená trajektória a pozícia robota po každej sekunde.

3.2. Vykreslenie štvorca

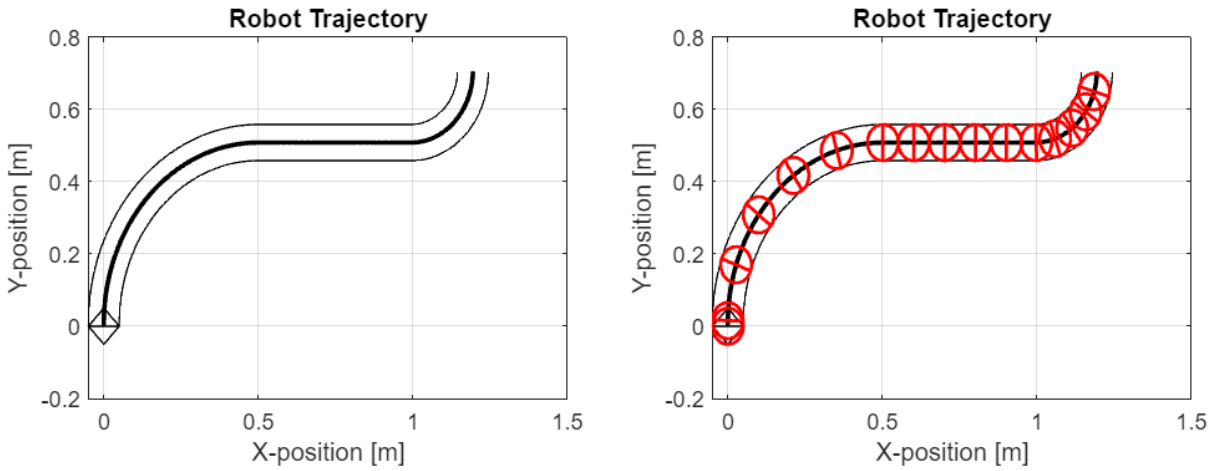
Na základe zadanej dĺžky strany štvorca boli vypočítané vektory rýchlostí. Trajektória robota bola následne vykreslená rovnakým spôsobom ako v predošlom kroku.



Obrázok 2 - Vykreslenie trajektórie štvorca

3.3. Vykreslenie krivky

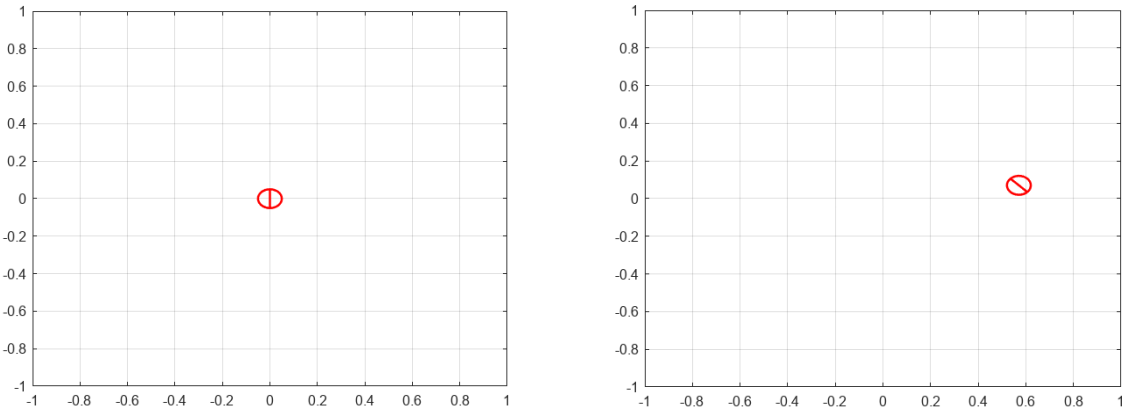
Na základe zadanej polomerov a dĺžky úsečky boli vypočítané vektory rýchlostí. Trajektória robota bola následne vykreslená rovnakým spôsobom ako v prvom kroku.



Obrázok 3 - Vykreslenie trajektórie krivky

3.4. WASD hra

Aktuálna poloha robota sa vypočítavala na základe stlačených klávesov. Následne bola vykreslená jeho aktuálna pozícia. Vzhľadom na to, že Matlab nepodporuje real-time komunikáciu s klávesnicou, robot sa po každom pohybe zastavil a čakal na ďalší povel z klávesnice. Tento problém by sa dal vyriešiť implementáciou iného programovacieho jazyka.

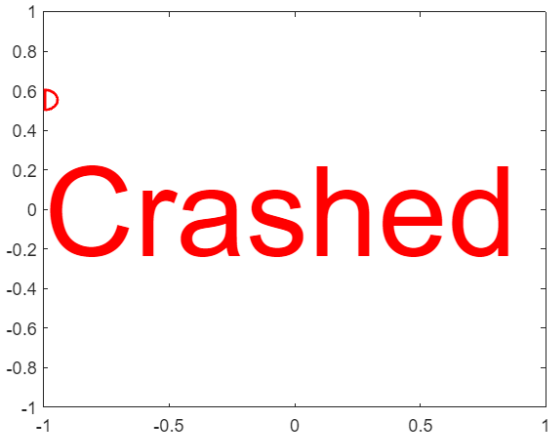


Obrázok 4 - Pozícia robota

Robot sa pohybuje a riadi pomocou klávesov `W`, `A`, `S`, `D`, `Q` a `R`.

- **Klávesy `W` a `S`** slúžia na nastavenie rýchlosti robota. Opakovaným stlačením toho istého klávesa sa rýchlosť násobí.
- **Klávesy `A` a `D`** slúžia na nastavenie rýchlosti otáčanie robota. Opakovaným stlačením toho istého klávesa sa rýchlosť násobí.
- **Kláves `R`** vynuluje všetky rýchlosti robota.
- **Kláves `Q`** slúži na ukončenie celého programu.

Program sa taktiež automaticky ukončí, ak robot narazí do okraja mapy.



Obrázok 5 - Ukončenie narazením do okraju mapy

4. KÓD

4.1. Trieda Robot

Kód bol rozdelený do viacerých tried, čím sa dosiahla lepšia prehľadnosť a modularita. Trieda s názvom "Robot" slúži na správu parametrov robota a implementuje funkcie na jeho vykreslenie a otáčanie.
V kontexte robotického vysávača táto trieda obsahuje funkcie na vykreslenie kruhu (reprezentujúceho telo vysávača).

```
classdef Robot
    properties
        d
        r
        armL
        armR
    end

    methods
        function self = Robot(d,r)
           self.d = d;
           self.r = r;
           self.armL = [0; self.d];
           self.armR = [0; -self.d];
        end

        function [] = draw(self,center, armL,armR,k)
            for i = 1:k:length(center)
                plot([armL(i,1) armR(i,1)], [armL(i,2) armR(i,2)], 'red',
'LineWidth',1.5);
                hold on;
                self.circle(center(i,1),center(i,2));
            end
        end

        function [] = circle(self, x, y)
            th = 0:pi/50:2*pi;
            xunit = self.r/2 * cos(th) + x;
            yunit = self.r/2 * sin(th) + y;
            plot(xunit, yunit, 'red', 'LineWidth',1.5);
        end

        function [armLr,armRr] = rotate(self, angle)
            rot = [cos(angle) -sin(angle) 0 0;
                  sin(angle) cos(angle) 0 0;
                  0 0 1 0;
                  0 0 0 1];

            armLr = rot * [self.armL; 0; 1];
            armRr = rot * [self.armR; 0; 1];
        end
    end
end
```

4.2. Trieda Trajektórie

Trieda s názvom " `Trajectory` " slúži na správu trajektórie robota. Implementuje funkcie na výpočet (Rovnica 1-Rovnica 7) a vizualizáciu trajektórie. Umožňuje vykreslenie samotnej trajektórie, ako aj trajektórie s označením pozícií robota v rôznych časových bodoch.

```
classdef Trajectory
    properties
        Robot
        k
        center
        armL
        armR
    end

    methods

        function self = Trajectory(vl,vr,Robot,k)
            self.Robot = Robot;
            self.k = k;

            dt = 1/k;

            vt = (vr+vl)/2;
            wt = ((vr-vl))/(Robot.d*4);

            steps = length(vl)*k;
            center = zeros(steps,2);
            armL = zeros(steps,2);
            armR = zeros(steps,2);
            psit = zeros(steps,1);

            % Initial position
            center(1,:) = [0 0];
            psit(1) = 0;

            for tm = 2:steps
                index = floor((tm - 1) / k) + 1;
                vx = vt(index) * cos(psit(tm-1));
                vy = vt(index) * sin(psit(tm-1));

                dxt = vx * dt;
                dyt = vy * dt;

                center(tm,:) = [center(tm-1,1) + dxt center(tm-1,2) + dyt];

                dpsi = wt(index)*dt;
                psit(tm) = psit(tm-1) + dpsi;

                [p1r,p2r] = Robot.rotate(psit(tm));

                armL(tm,:) = [center(tm,1)+p1r(1) center(tm,2)+p1r(2)];
                armR(tm,:) = [center(tm,1)+p2r(1) center(tm,2)+p2r(2)];
            end
```

```
            self.center = center;
            self.armL = armL;
            self.armR = armR;
        end

        function [] = drawTrajectory(self)
            figure

            plot(self.center(:,1),self.center(:,2),'black','LineWidth',2);
            hold on;
            plot(self.armL(:,1),self.armL(:,2),'black');
            plot(self.armR(:,1),self.armR(:,2),'black');
            xlabel('X-position [m]');
            ylabel('Y-position [m]');
            title('Robot Trajectory');
            grid on;

        end

        function [] = drawRobot(self)
            self.drawTrajectory();
            self.Robot.draw(self.center,self.armL,self.armR,self.k);
        end
    end
end
```

4.3. Podtriedy jednotlivých úloh

4.3.1. Prvá úloha

Podtrieda vytvorí objekt typu Trajectory, do ktorého uloží zadanú trajektóriu.

```
classdef Task1 < Trajectory
    methods
        function self = Task1(vl,vr,Robot,k)
            self = self@Trajectory(vl,vr,Robot,k);
        end
    end
end
```

4.3.2. Druhá úloha

Podtrieda vypočíta trajektóriu na základe zadanej strany štvorca. Následne vytvorí objekt typu Trajectory a uloží doň vypočítanú trajektóriu.

```
classdef Task2 < Trajectory
    methods
        function self = Task2(side,Robot,k)
            steps = (side*k+1)*4;
            vl = zeros(steps,1);
```

```
            vr = zeros(steps,1);

            for step = 1:1:steps
                vl(step+1)=1/k;
                vr(step+1)=1/k;
                if mod(step,steps/4) == 0
                    vl(step+1)=-pi/(2*k);
                    vr(step+1)=pi/(2*k);
                end
            end
            self = self@Trajectory(vl,vr,Robot,k);
        end
    end
end
```

4.3.3. Tretia úloha

Podtrieda vypočíta (Rovnica 9) trajektóriu na základe zadaných parametrov krivky. Následne vytvorí objekt typu Trajectory a uloží doň vypočítanú trajektóriu.

```
classdef Task3 < Trajectory
    methods
        function self = Task3(R1,L, R2,Robot,k)
            steps = L/(1/k);
            vl = zeros(steps,1);
            vr = zeros(steps,1);

            vl(1)=-pi/2;
            vr(1)=pi/2;

            for step = 1:1:6
                 vl(step+1)=(pi*2*Robot.d + pi*R1)/10;
                 vr(step+1)=(-pi*2*Robot.d + pi*R1)/10;
            end

            for step = 1:1:steps
                 vl(step+6)=1/k;
                 vr(step+6)=1/k;
            end

            for step = 1:1:5
                 vl(step+6+steps)=(-pi*2*Robot.d + pi*R2)/10;
                 vr(step+6+steps)=(pi*2*Robot.d + pi*R2)/10;
            end

            self = self@Trajectory(vl,vr,Robot,k);
        end
    end
end
```

4.4. Štvrtá úloha

Trieda si najprv definovala vlastné štartovacie parametre, ktoré slúžili na konfiguráciu jej fungovania. Následne spustila nekonečnú slučku, ktorá čakala na stlačenie klávesu. Po stlačení klávesu trieda analyzovala stlačený kláves a na základe jeho hodnoty vykonala príslušnú akciu. V závislosti od stlačeného klávesu trieda aktualizovala stav robota a jeho pozíciu v simulovanom prostredí. Nakoniec trieda vykreslila robota v aktuálnej polohe a stave na obrazovke.

```
classdef Task4
    methods
        function self = Task4(Robot)
            d = Robot.d;
            center = [0 0];
            armL = [0 d];
            armR = [0 -d];
            psi = 0;
            v = 0;
            w = 0;

            figure
            Robot.drawRobot(center,armL,armR)

            while true
                if abs(center(1)) > 0.9 || abs(center(2)) > 0.9
                    text(-1, 0, 'Crashed', 'FontSize', 70, 'Color', 'red');
                    break
                end

                key = getkey;

                switch key
                    case 'w'
                        v = v + 0.1;
                    case 's'
                        v = v - 0.1;
                    case 'a'
                        w =w + pi/4;
                    case 'd'
                        w =w - pi/4;
                    case 'r'
                        v = 0;
                        w = 0;
                    case 'q'
                        break;
                end

                psi = psi + w;
                center = center + [v*cos(psi) v*sin(psi)];

                [p1r,p2r] = Robot.rotate(psi);

                armL = center + [p1r(1) p1r(2)];
                armR = center + [p2r(1) p2r(2)];
```

```
                clf;

                Robot.drawRobot(center,armL,armR)
                drawnow;
            end
        end
    end
end
```

4.5. Príklad main-u

Definition

```
d = 0.1;

r = 0.05;

L = 2*d;

k = 10; % k-times per second

Bobik = Robot(r,d);
```

Task1

```
vl = [0; 2; pi/2; 2; pi; 2]/10;

vr = [0; 2; -pi/2; 2; 0; 2]/10;

Task1 = Task1(vl,vr,Bobik,k);

Task1.drawTrajectory();

Task1.drawRobot();
```

Task2

```
side = 0.5;

Task2 = Task2(side,Bobik,k);

Task2.drawTrajectory();

Task2.drawRobot();
```

Task3

```
R1 = 0.5;

L = 0.5;

R2 = 0.2;

Task3 = Task3(R1,L,R2,Bobik,k);

Task3.drawTrajectory();

Task3.drawRobot();
```

Task4

```
Task4 = Task4(Bobik);
```

5. ZÁVER

Tento projekt zaoberal vizualizáciou diferenciálneho podvozku robota. Vďaka použitiu objektovo orientovaného programovania sa podarilo dosiahnuť prehľadný a variabilný kód, čím sa zjednodušila údržba a rozšíriteľnosť.

Napriek obmedzeniam Matlabu v oblasti real-time vykresľovania sa podarilo dosiahnuť požadované výsledky. Vizualizácia robota bola plynulá a dostatočne detailná na pochopenie jeho pohybu a interakcie s prostredím.

Project by sa dal vylepšiť implementáciou real-time vykresľovania pomocou iného programovacieho jazyka (napr. Python, C++).

„Zadanie som vypracoval sám. Čestne prehlasujem, že som ho neskopíroval a nikomu inému neposkytol. Nech mi je Isaac Asimov svedkom.“

6. ZDROJE

[1.] MathWorks. (n.d.). MATLAB Documentation. Dostupné z: https://www.mathworks.com/help/matlab/
[2.] MathWorks. (n.d.). MATLAB Documentation - getkey. Dostupné z: https://ch.mathworks.com/matlabcentral/fileexchange/7465-getkey?status=SUCCESS
[3.] OpenAI. (n.d.). Gemini. Dostupné z: https://gemini.google.com/app