# Robotyka - Laboratorium: "Odometria"

Konrad Borowik 141023 Cezary Wawrzyniak 141131 Wojciech Krysiak 140264

29 marca 2021

## Zadanie 1

**a)** Równanie stanu dla wektora stanu q = $\begin{bmatrix} \varphi & x & y \end{bmatrix}$ oraz wektora sterującego u = $\begin{bmatrix} \omega & v \end{bmatrix}$

$$\dot{q} = \begin{bmatrix} \dot{\varphi} \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 \\ \cos(\varphi) \\ \sin(\varphi) \end{bmatrix} \cdot v \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \omega$$

**b)** Równanie stanu dla wektora stanu q = $\begin{bmatrix} \varphi & x & y \end{bmatrix}$ oraz wektora sterującego: u = $\begin{bmatrix} \omega_P & \omega_L \end{bmatrix}$

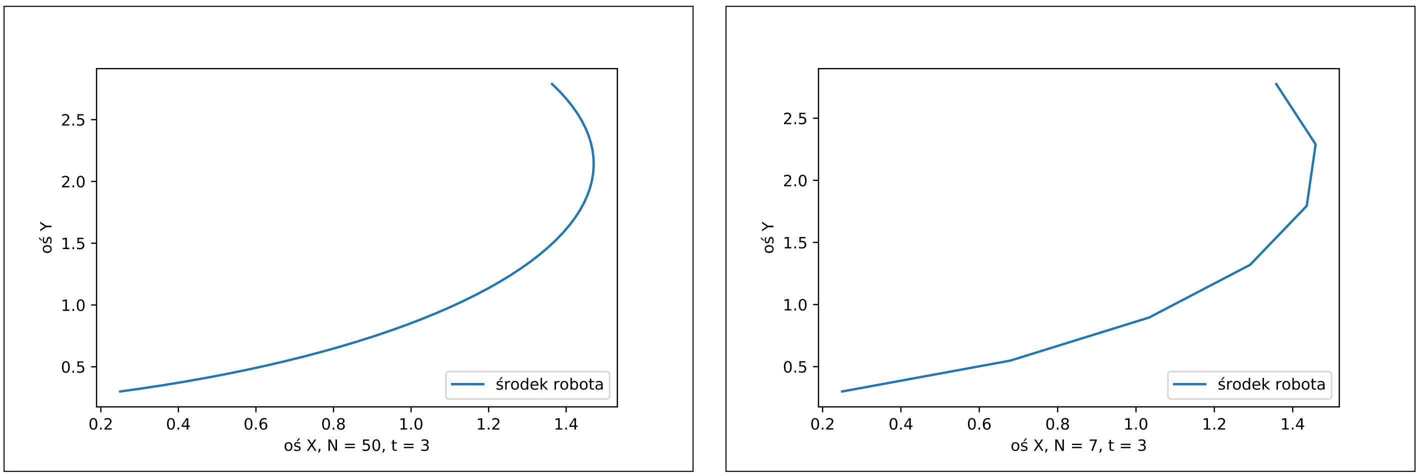
$$\dot{q} = \begin{bmatrix} \dot{\varphi} \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{R}{d} \\ \frac{\cos(\varphi) \cdot R}{2} \\ \frac{\sin(\varphi) \cdot R}{2} \end{bmatrix} \cdot \omega_P \begin{bmatrix} \frac{-R}{d} \\ \frac{\cos(\varphi) \cdot R}{2} \\ \frac{\sin(\varphi) \cdot R}{2} \end{bmatrix} \cdot \omega_L$$

## Zadanie 2

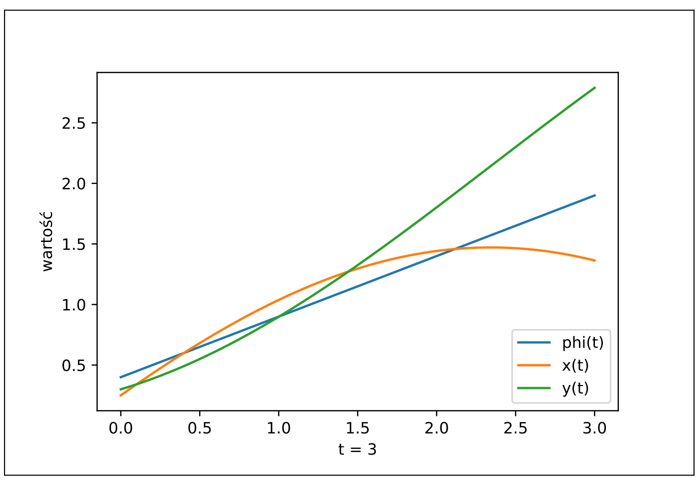
**a)** W kodzie kolejne współrzędne obliczane są skalarnie na podstawie poprzedniej wartosci. Wykorzystana została funkcja trapz() biblioteki numpy, do wyliczania całek, natomiast do wyrysowania wykresów użyty został moduł matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

N = 50
initCondi = [0.4, 0.25, 0.3]

w = np.linspace(.5, .5, N)
vx = np.linspace(1, 1, N)
vy = np.linspace(1, 1, N)
t = np.linspace(0, 3, N)

phi_t = [initCondi[0]]
for i in range(len(t)-1):
    calka = phi_t[i] + np.trapz([w[i], w[i+1]],[t[i], t[i+1]])
    phi_t.append(calka)

x_t = [initCondi[1]]
for i in range(len(t)-1):
    calka = x_t[i] + np.trapz([vx[i]*np.cos(phi_t[i]),
```

```
                                    vx[i+1]*np.cos(phi_t[i+1])],[t[i], t[i+1]])
    x_t.append(calka)

y_t = [initCondi[2]]
for i in range(len(t)-1):
    calka = y_t[i] + np.trapz([vy[i]*np.sin(phi_t[i]),
                                    vy[i+1]*np.sin(phi_t[i+1])],[t[i], t[i+1]])
    y_t.append(calka)

plt.plot(x_t, y_t, label="srodek robota")
plt.xlabel("os X, N = 50, t = 3")
plt.ylabel("os Y")
plt.legend(loc=4)
```

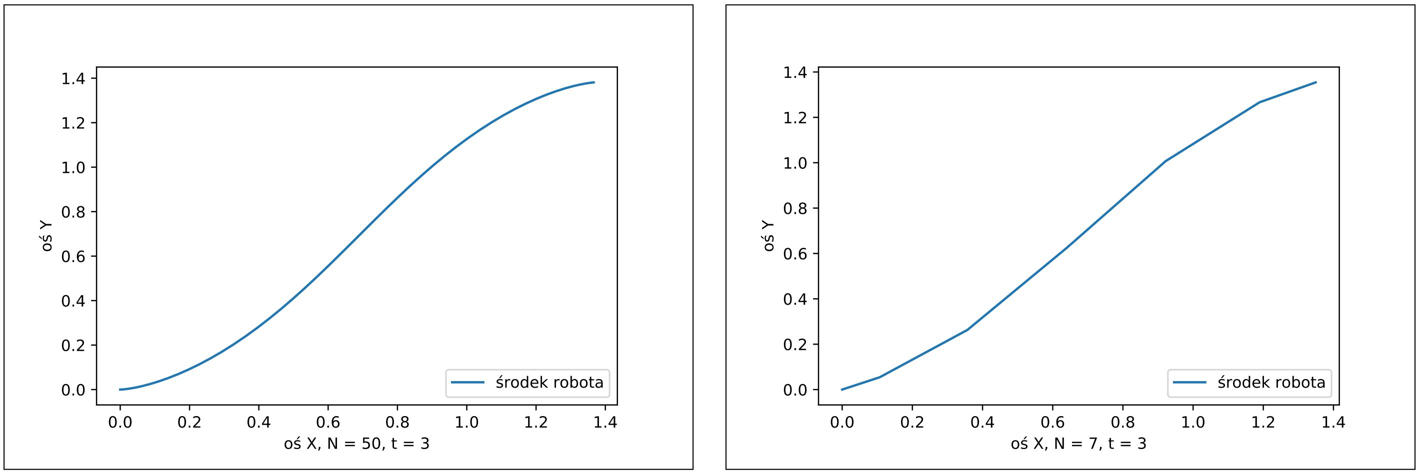



Wykresy położenia robota w układzie kartezjańskim dla różnych stałych próbkowania 'N'

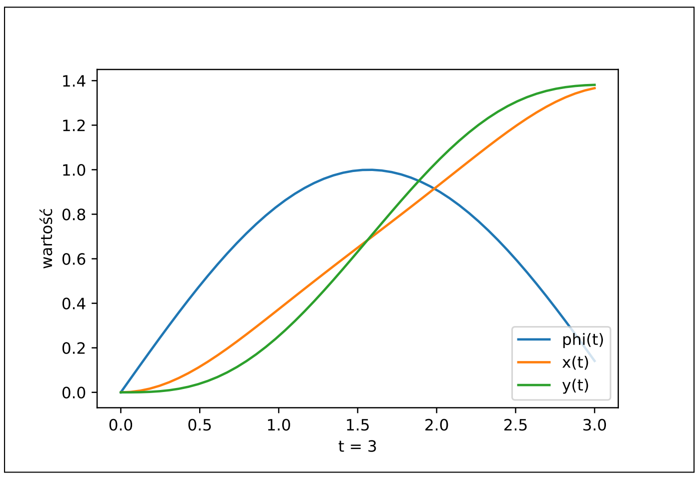



Funkcje zmiennych stanu w zależnosci od czasu

**b)** Ruch robota zamodelowany jest w taki sam sposób jak w podpunkcie 2a), jednak dla innych warunków początkowych i innych wartosci sterowania.



Wykresy położenia robota w układzie kartezjańskim dla różnych stałych próbkowania 'N'

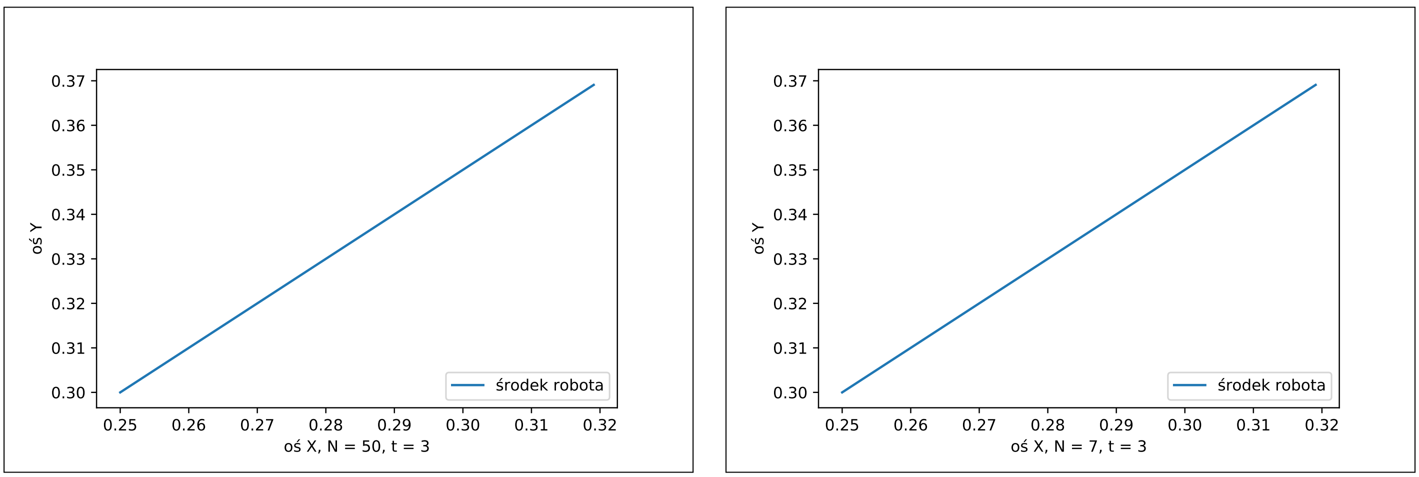


Funkcje zmiennych stanu w zależnosci od czasu

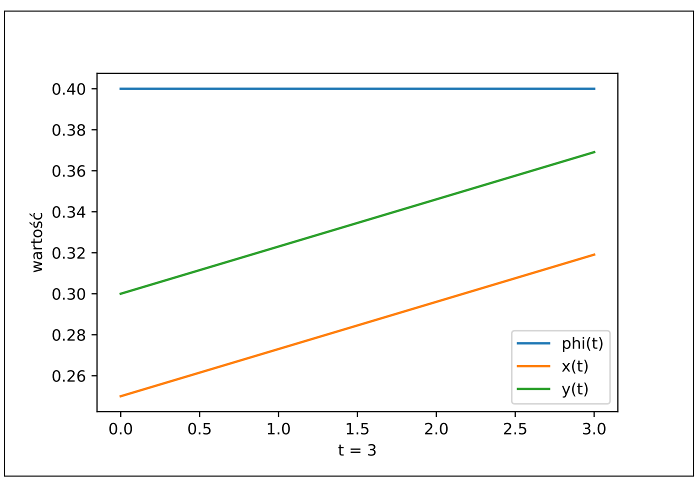
**c)** Kod funkcjonuje w ten sam sposób jak w poprzednich dwóch podpunktach, z tą różnicą, że całki wyliczane są ze wzorów wykorzystujących prędkosci kół. (nowe sterowanie)

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

N = 7
initCondi = [0.4, 0.25, 0.3]
r = 0.025
d = 0.145

wl = np.linspace(1, 1, N)
wr = np.linspace(1, 1, N)
t = np.linspace(0, 3, N)

phi_t = [initCondi[0]]
for i in range(len(t)-1):
    calka = phi_t[i] + np.trapz([(wr[i]-wl[i])*r/d, (wr[i+1]-wl[i+1])*r/d],[t[i], t[i+1]])
    phi_t.append(calka)
```

```python

x_t = [initCondi[1]]
for i in range(len(t)-1):
    calka = x_t[i] + np.trapz([(wr[i]+wl[i])*r/2*np.cos(phi_t[i]),
                                (wr[i+1]+wl[i+1])*r/2*np.cos(phi_t[i+1])],[t[i], t[i+1]])
    x_t.append(calka)

y_t = [initCondi[2]]
for i in range(len(t)-1):
    calka = y_t[i] + np.trapz([(wr[i]+wl[i])*r/2*np.cos(phi_t[i]),
                                (wr[i+1]+wl[i+1])*r/2*np.cos(phi_t[i+1])],[t[i], t[i+1]])
    y_t.append(calka)

plt.plot(x_t, y_t, label="srodek robota")
plt.xlabel("os X, N = 7, t = 3")
plt.ylabel("os Y")
plt.legend(loc=4)
```

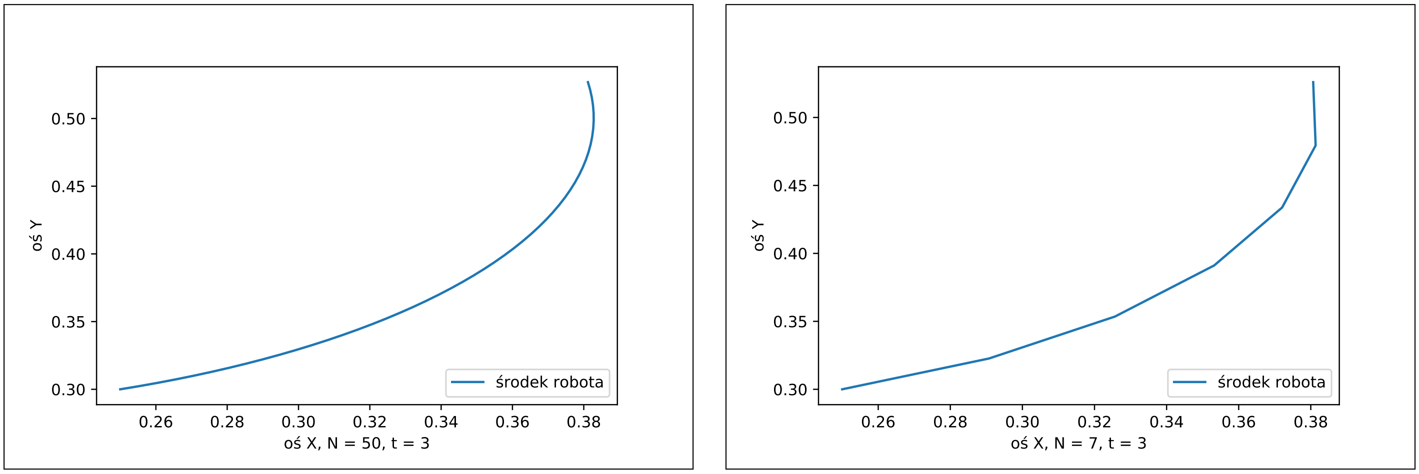


Wykresy położenia robota w układzie kartezjańskim dla różnych stałych próbkowania ’N’

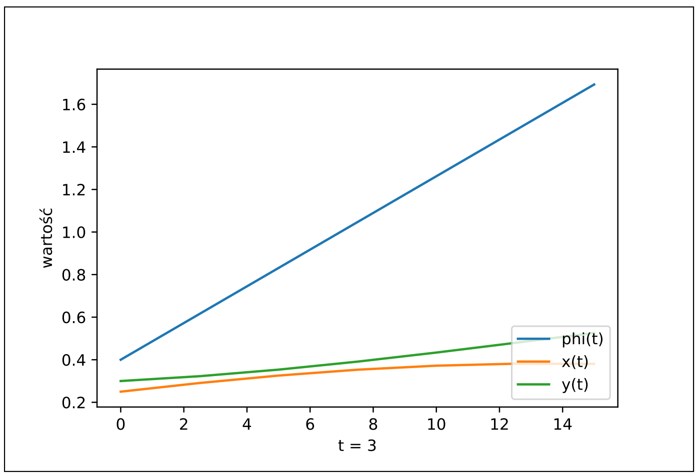


Funkcje zmiennych stanu w zależnosci od czasu

**d)** Ruch robota zamodelowany jest w taki sam sposób jak w podpunkcie 2c), jednak dla innych warunków początkowych i innych wartosci sterowania.




Wykresy położenia robota w układzie kartezjańskim dla różnych stałych próbkowania 'N'




Funkcje zmiennych stanu w zależnosci od czasu

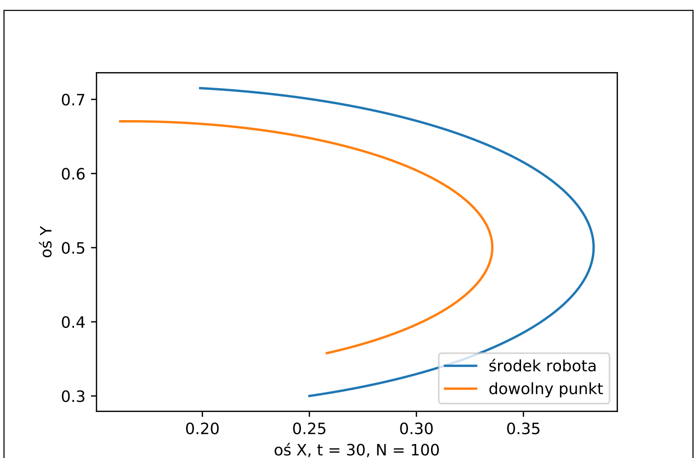
## Zadanie 3

**a)** Ruch robota oraz dowolnie obranego punktu w układzie lokalnym robota, przedstawiony na płaszczyźnie X-Y układu globalnego. Za wyliczenie punktów obranego punktu P odpowiedzialna jest pętla, która skalarnie wylicza punkty 'newPx' oraz 'newPy', na podstawie równania:

$$P = \begin{bmatrix} x \\ y \end{bmatrix} + R(\varphi) \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

initCondi = [0.4, 0.25, 0.3]
N = 100
wl = np.linspace(.5, .5, N)
wr = np.linspace(1, 1, N)
t = np.linspace(0, 30, N)
r = 0.025
```

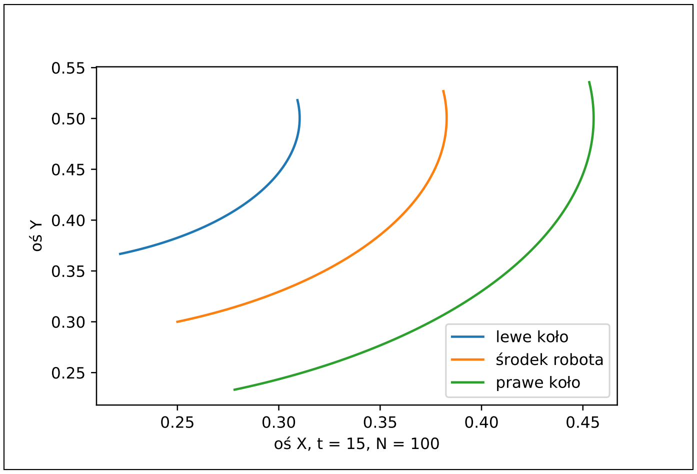
```
d = 0.145

phi_t = [initCondi[0]]
for i in range(len(t)-1):
    calka = phi_t[i] + np.trapz([(wr[i]-wl[i])*r/d, (wr[i+1]-wl[i+1])*r/d],[t[i], t[i+1]])
    phi_t.append(calka)

x_t = [initCondi[1]]
for i in range(len(t)-1):
    calka = x_t[i] + np.trapz([(wr[i]+wl[i])*r/2*np.cos(phi_t[i]),
                              (wr[i+1]+wl[i+1])*r/2*np.cos(phi_t[i+1])],[t[i], t[i+1]])
    x_t.append(calka)

y_t = [initCondi[2]]
for i in range(len(t)-1):
    calka = y_t[i] + np.trapz([(wr[i]+wl[i])*r/2*np.sin(phi_t[i]),
                                (wr[i+1]+wl[i+1])*r/2*np.sin(phi_t[i+1])],[t[i], t[i+1]])
    y_t.append(calka)

a = .03
b = .05
Px = []
Py = []
for i in range(len(t)):
    newPx = x_t[i] + np.cos(phi_t[i])*a + (-1)*np.sin(phi_t[i])*b
    Px.append(newPx)
    newPy = y_t[i] + np.sin(phi_t[i])*a + np.cos(phi_t[i])*b
    Py.append(newPy)

plt.plot(x_t, y_t, label="srodek robota")
plt.plot(Px,Py, label="dowolny punkt")
plt.xlabel("os X, t = 30, N = 100")
plt.ylabel("os Y")
plt.legend(loc=4)
```



Wykres położenia robota oraz punktu P o początkowych współrzędnych (0,03; 0,05)

**b)** Ruch robota oraz jego kół. Współrzędne kół wyliczane są tak samo jak współrzędne dowolnie obranego punktu, z tą różnicą, że ich współrzędne lokalne to $(0; \pm 0,0725)$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
initCondi = [0.4, 0.25, 0.3]
N = 100
wl = np.linspace(.5, .5, N)
wr = np.linspace(1, 1, N)
t = np.linspace(0, 15, N)
r = 0.025
d = 0.145

phi_t = [initCondi[0]]
for i in range(len(t)-1):
    calka = phi_t[i] + np.trapz([(wr[i]-wl[i])*r/d, (wr[i+1]-wl[i+1])*r/d],[t[i], t[i+1]])
    phi_t.append(calka)

x_t = [initCondi[1]]
for i in range(len(t)-1):
    calka = x_t[i] + np.trapz([(wr[i]+wl[i])*r/2*np.cos(phi_t[i]),
                              (wr[i+1]+wl[i+1])*r/2*np.cos(phi_t[i+1])],[t[i], t[i+1]])
    x_t.append(calka)

y_t = [initCondi[2]]
for i in range(len(t)-1):
    calka = y_t[i] + np.trapz([(wr[i]+wl[i])*r/2*np.sin(phi_t[i]),
                                (wr[i+1]+wl[i+1])*r/2*np.sin(phi_t[i+1])],[t[i], t[i+1]])
    y_t.append(calka)

a = 0
b = -0.0725
Pxr = []
Pyr = []
for i in range(len(t)):
    newPx = x_t[i] + np.cos(phi_t[i])*a + (-1)*np.sin(phi_t[i])*b
    Pxr.append(newPx)
    newPy = y_t[i] + np.sin(phi_t[i])*a + np.cos(phi_t[i])*b
    Pyr.append(newPy)

a = 0
b = 0.0725
Pxl = []
Pyl = []
for i in range(len(t)):
    newPx = x_t[i] + np.cos(phi_t[i])*a + (-1)*np.sin(phi_t[i])*b
    Pxl.append(newPx)
    newPy = y_t[i] + np.sin(phi_t[i])*a + np.cos(phi_t[i])*b
    Pyl.append(newPy)

plt.plot(Pxl,Pyl, label="lewe koło")
plt.plot(x_t, y_t, label="srodek robota")
plt.plot(Pxr,Pyr, label="prawe koło")
plt.xlabel("os X, t = 15, N = 100")
plt.ylabel("os Y")
plt.legend(loc=4)
```



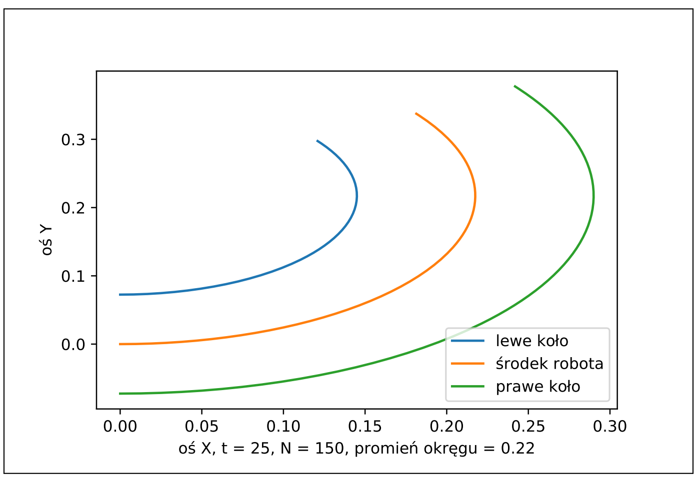

Wykres położenia środka robota oraz jego kół na płaszczyźnie XY

**Zadanie 4** Na podstawie prędkości kątowych kół, promień okręgu zakreślanego przez robota można obliczyć ze wzoru:

$$R = \frac{\omega_r + \omega_l}{\omega_r - \omega_l} \cdot \frac{d}{2}$$

Funkcja circleRadius przyjmuje wartości prędkości kątowych kół oraz ich promień i na podstawie powyższego równania oblicza promień okręgu, po którym porusza się robot.

```
def circleRadius(w1, w2, d):
    R = (w1+w2)/(w1 - w2)*d/2
    return R
```

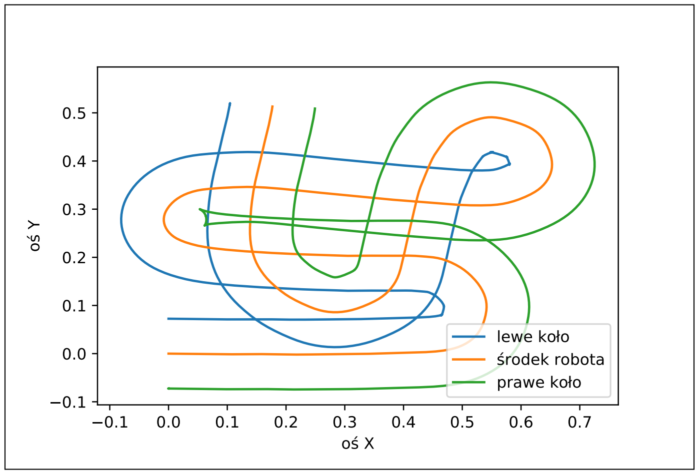



Wykres położenia środka robota raz jego kół na płaszczyźnie XY z wyliczonym promieniem dla prędkości kół

$$[\omega_r, \omega_l] = [1; 0,5]$$

**Zadanie 5** Pliki zostały wczytane komendą read_csv biblioteki pandas. Trzeba było najpierw zmienić przecinki na kropki, aby umożliwić odczytanie ułamków. Pliki typu 'csv' zawierają wartości oddzielone przecinkami, dlatego odczytując ten plik, użyta została komenda 'sep="\t", która odczytuje tabulatory jako oddzielenie danych. Następnie każda kolumna danych przypisana została do odpowiedniej zmiennej w postaci tablicy danych. Na tej podstawie wyliczone zostało położenie robota oraz jego kół w każdej zadanej chwili czasu.

```
CustomValues = pd.read_csv("Profile_predkosci_V2tsv.txt", sep="\t")
wl = CustomValues["wl"]
wr = CustomValues["wp"]
t = CustomValues["t"]
```



Mapa przejazdu robota

**Zadanie 6** Funkcje bloków Difference oraz Gain1 to przeprowadzenie transformacji Eulera wstecz (odpowiada też metodzie całkowania poprzez sumowanie pól prostokątów).

Wszystkie wykresy położenia robota z zadań 2-4 prezentowane w globalnym układzie XY, powinny być fragmentami okręgów. Niestety osie x i y obierały różne skale, dlatego te wykresy wyglądają jak fragmenty elips.