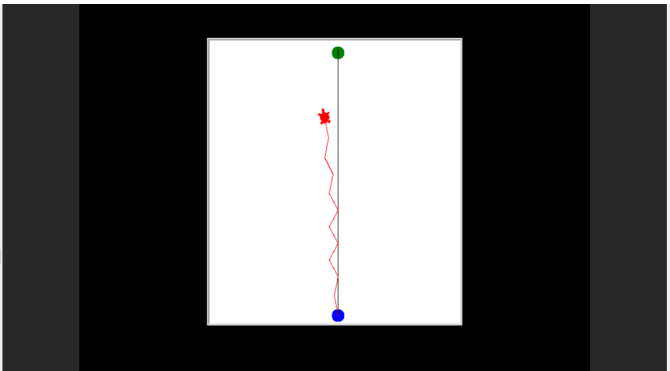


Ćwiczenie 3

Sterowanie za pomocą kontrolera PID - wersja dla środowiska Turtle

```
90     if cte>steering_range:
91         turtle2_left_wheel=0
92     else:
93         turtle2_left_wheel=turtle2_right_wheel-(ksi*cte)
94     print("turtle2_left_wheel=",turtle2_left_wheel,"turtle2_right_wheel=",turtle2_right_wheel)
95     nowy_kierunek=kierunek_cel+beta*(turtle2_right_wheel-turtle2_left_wheel)
96     nowy_kierunek=new_kierunek%360
97     print("nowy_kierunek=",nowy_kierunek)
98     return nowy_kierunek
99
100 #sterowanie za pomocą kontrolera P,
101 #ustalamy bazową prędkość kół
102 turtle2_left_wheel=50
103 turtle2_right_wheel=50
104 steering_range=10
105 #startujemy kierunek robota
106 temp_direction=random.choice([kierunek_cel+2*steering_range,kierunek_cel-2*steering_range])
107 step=0.5
108 ksi=0.5
109 beta=0.5
110 while turtle2_pos()[1]>factor*stop-2:
111     reality_gap=random.choice([-10,10])
112     temp_direction=steer(turtle2_left_wheel,turtle2_right_wheel,kierunek_cel,temp_direction,
113                         steering_range,reality_gap,ksi,beta)
114     turtle2.setheading(temp_direction)
115     turtle2.forward(step*factor)
116     g*****dodatek*****
117     #random.randrange(0,180,1) #losowanie liczb z przedziału, co ustaloną wartość,
```



```
nowy_kierunek= 75.04564930671381
inside steer
cte= 15.0
turtle2_left_wheel= 0 turtle2_right_wheel= 50
nowy_kierunek= 125.04564930671381
inside steer
cte= -15.0
turtle2_left_wheel= 50 turtle2_right_wheel= 0
nowy_kierunek= 75.04564930671381
inside steer
cte= 15.0
turtle2_left_wheel= 0 turtle2_right_wheel= 50
nowy_kierunek= 105.04564930671381
* []
```

Zadanie do wykonania

1) Wczytujemy na stronie <https://repl.it> kod języka python z pliku P-kontroler.txt

2) Jeżeli nie mamy utrwalonej umiejętności programowania w środowisku Turtle, zapoznajemy się z kursami użytkowania środowiska Turtle:

- Podstawy
<https://docs.python.org/3/library/turtle.html#introduction>
- Zaawansowane rysowanie
<https://blog.furas.pl/rysowanie-w-turtle-okrag-luk-kolo-i-elipsa.html>
- Rysowanie wypełnionych przeszkód
<https://www.tutorialsandyou.com/python/how-to-draw-color-filled-shapes-in-py.html>

3) Zapoznajemy się z częścią teoretyczną - opisującą działanie kontrolera PID,

4) Uruchamiamy kod z pliku P-kontroler.txt dla różnych parametrów ksi, beta, step, steering_range, szukamy parametrów, które minimalizują błąd odchylenia od wyrysowanej (trasy), prostej. Parametry przeglądamy automatycznie obliczając błąd odstępstwa od perfekcyjnej trasy. Kod z zaznaczeniem w komentarzu najlepszych

znalezionych parametrów zapisujemy do pliku tuning-P-kontroler.txt i udostępniamy prowadzącemu ćwiczenia.

Ograniczenia wartości parametrów:

$$ksi \in [0.01, 1]$$

$$beta \in [0.1, 1]$$

$$step \in [0.1, 1]$$

$$steering_range \in [20, 50]$$

,
Parametry *turtle2_left_wheel*, *turtle2_right_wheel*, mogą być zmieniane tylko w obrębie funkcji *steer*, poza nią przyjmują wartość 50. Nie modyfikujemy parametru *reality_gap* i startowej wartości *temp_direction*.

5) Implementujemy kontroler PD na bazie kodu kontrolera P, P-kontroler.txt. Zapisujemy wynik pracy do pliku, PD-kontroler-nazwisko.txt i udostępniamy prowadzącemu zajęcia,

6) Implementujemy kontroler PID na bazie kodu kontrolera P, P-kontroler.txt. Zapisujemy wynik pracy do pliku, PID-kontroler-nazwisko.txt i udostępniamy prowadzącemu zajęcia,

7) Materiały dodatkowe dla chętnych - rozszerzenie wiedzy o kontrolerach PID: <https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf>

Zasady oceny zadania: Ćwiczenie składa się z trzech podzadań (punkty 4),5),6)), każde oceniane w wymiarze $\frac{1}{3}$ punktu.

Część teoretyczna - sterowania za pomocą kontrolera PID

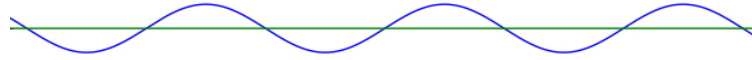
Zasada działania kontrolera PID - ogólny opis

Zacznijmy od opisanie P-kontrolera. Jeżeli *cte* - jest odchyleniem od ustalonej wartości docelowej, α kątem sterowania robota, ξ współczynnikiem zmiany kąta sterowania. Wtedy kąt sterowania α wyliczany proporcjonalnie do odchylenia *cte* jest wyliczany jako

$$\alpha = -\xi * cte$$

Sterowanie robotem mobilnym w ustalonym kierunku z pomocą P-kontrolera może dać efekt widoczny na rysunku 1.

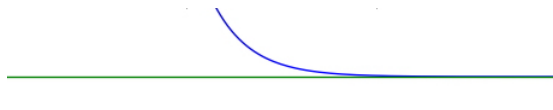
Robot szybko zwraca się w kierunku docelowym, gdy jest już na odpowiedniej trajektorii delikatnie oscyluje (patrz rysunek 1). Aby zniwelować efekt oscylacji można w kontrolerze uwzględnić zmianę błędu toru jazdy pomiędzy iteracjami sterowania - (stworzyć PD-kontroler). Ustalamy współczynnik ϱ , i każdorazowo wyliczamy różnicę *cte* w poprzednim i następnym kroku, czyli w czasie $t - 1$ i t , różnica między



Rysunek 1: Efekt działania kontrolera proporcjonalnego do błędu cte (P-kontrolera)

kolejnymi odchyleniami od toru jazdy jest obliczana następująco, $Dcte = \frac{cte_t - cte_{t-1}}{\Delta t}$, przyjmując, że różnica jest sprawdzana cyklicznie z tym samym odstępem czasu, kąt sterowania PD-kontrolera może być wyliczony następująco,

$$\alpha = -\xi * cte - \varrho * Dcte$$



Rysunek 2: Efekt działania kontrolera proporcjonalnego do cte i zmian błędu pomiędzy iteracjami kontroli (PD kontroler)

Kolejnym problemem jaki możemy napotkać podczas sterowania robotem mobilnym jest niedokładność części jezdnej prowadząca do rozbieżności położenia robota od zamierzonej ścieżki. Problem obrazujemy na rysunku 3.



Rysunek 3: Rozbieżność położenia robota od zamierzonej ścieżki jazdy

Problem może być niwelowany przez uwzględnienie przeskalowanej wartości sumy wszystkich zaobserwowanych błędów, przyjmijmy parametr rozbieżności jako τ , po uwzględnieniu ewentualnego odchylenia od toru jazdy kąt sterowania może przyjąć wartość

$$\alpha = -\xi * cte - \varrho * Dcte - \tau * \sum cte$$

Kontroler PID w sterowaniu agentem turtle2 w środowisku Turtle

Przykładowy kontroler *P-kontroler.txt* może służyć do sterowania agentem *turtle2* w ustalonym kierunku *kierunek_cel* na podstawie odczytów aktualnego kierunku jazdy *temp_direction*. W tym wariantcie w przypadku agenta *turtle2* kontrola sprowadza się do sterowania poszczególnymi kołami agenta (*turtle2_left_wheel*, *turtle2_right_wheel*) z ustaloną prędkością.

Przyjmując, że (*turtle2_left_wheel*, *turtle2_right_wheel*), są odpowiednio prędkością koła lewego i koła prawego, *kierunek_cel* jest kierunkiem docelowym robota, *temp_direction* jest obecnym odczytem z kompasu, który konwertujemy za pomocą funkcji *convert(kierunek_cel,temp_direction)*, *ksi* jest parametrem sterującym kontrolera, *steering_range* definiuje w jakim przedziale odchylenia od docelowego kierunku działa kontroler, *reality_gap* symuluje losowy poślizg robota, *beta* określa jak szybko robot zmienia swój kierunek jazdy na podstawie różnicy prędkości kół. Prosta symulacja sterowania oparta na proporcji błędu (P-kontroler) może wyglądać następująco,

```
def steer(turtle2_left_wheel, turtle2_right_wheel, kierunek_cel, temp_direction,
         steering_range, reality_gap, ksi, beta):
    print('inside steer')
    cte=convert(kierunek_cel, temp_direction)
    print('cte=', cte)
    if cte <= 0:
        if abs(cte) > steering_range:
            turtle2_right_wheel = 0
        else:
            turtle2_right_wheel = turtle2_left_wheel + (ksi * cte)
    if cte > 0:
        if cte > steering_range:
            turtle2_left_wheel = 0
        else:
            turtle2_left_wheel = turtle2_right_wheel - (ksi * cte)
    print('turtle2_left_wheel=', turtle2_left_wheel, 'turtle2_right_wheel=',
          turtle2_right_wheel)
    nowy_kierunek = kierunek_cel + beta * (turtle2_right_wheel - turtle2_left_wheel)
    nowy_kierunek = nowy_kierunek % 360
    nowy_kierunek = (nowy_kierunek + reality_gap) % 360
    print('nowy_kierunek=', nowy_kierunek)
    return nowy_kierunek
```

Działanie powyższego kodu można sprawdzić następująco,

```
#sterowanie za pomocą kontrolera P,
#ustalam bazową prędkość kół
turtle2_left_wheel=50
turtle2_right_wheel=50
steering_range=50
#startowy kierunek robota
temp_direction=random.choice([kierunek_cel-2*steering_range, kierunek_cel+2*
                              steering_range])
step=0.5
ksi=1
beta=1
while turtle2.pos()[1]/factor < y_stop:
    reality_gap=random.choice([-10,10])
    temp_direction=steer(turtle2_left_wheel, turtle2_right_wheel, kierunek_cel,
                        temp_direction, steering_range, reality_gap, ksi, beta)
    turtle2.setheading(temp_direction)
    turtle2.forward(step*factor)
```

Do demonstracji sterowania została użyta składnia języka python, środowiska Turtle.

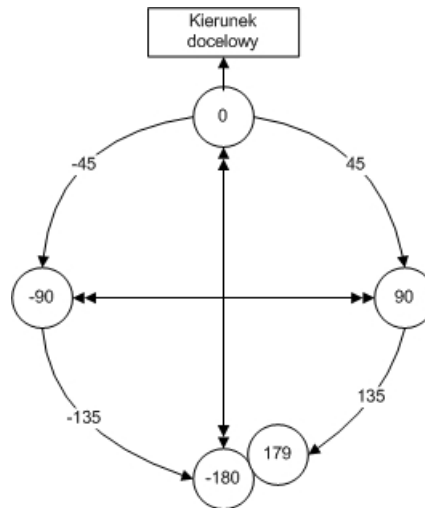
Pobieranie wartości kierunku z kompasu w środowisku Turtle

Kompas w środowisku Turtle pobiera wartość kierunku agenta *turtle2* za pomocą komendy *turtle2.heading()*. Spodziewane odczyty należą do przedziału [0, 359]

0 - Wschód,
180 - Zachód,
90 - Północ,
270 - Południe.

Przykładowe skalowanie odczytu z kompasu

W celu uproszczenia sterowania agentem odczytane wartości z kompasu *temp_direction* możemy przeskalować względem docelowego kierunku *kierunek_cel* w następujący sposób,



Rysunek 4: Kierunki kompasu po skalowaniu

Pseudokod procedury skalowania jest następujący,

```
def convert(kierunek_cel, temp_direction):  
    #print('temp_direction=', temp_direction)  
    #='kierunek_cel')  
    temp_direction=kierunek_cel-temp_direction  
    if temp_direction > 180:  
        temp_direction=temp_direction-360  
    if temp_direction < -180:  
        temp_direction=360+temp_direction  
    return temp_direction
```