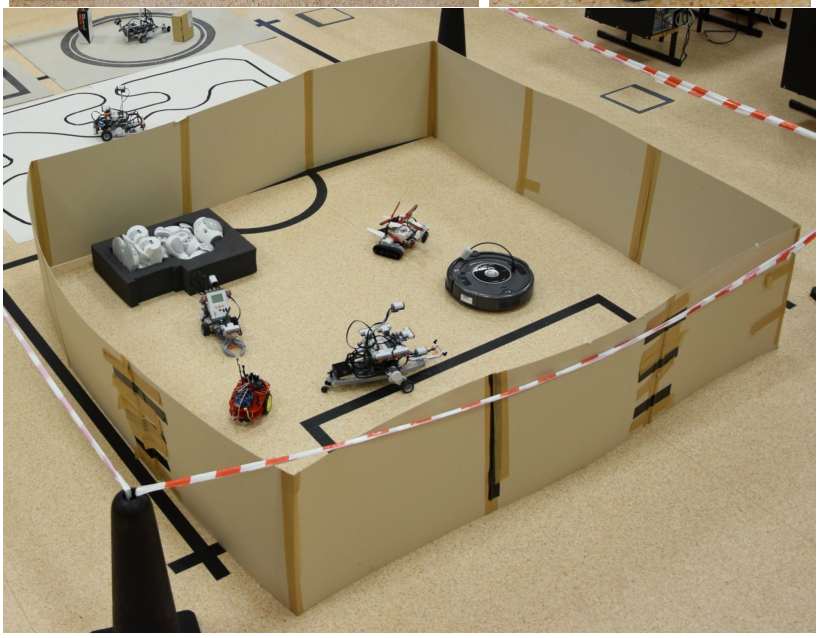
Ćwiczenie 4, 5

Lokalizacja za pomocą Filtru Histogramowego



Zadanie do wykonania

1) Wczytujemy na stronie `https://repl.it` kod języka python z pliku filtr1D.txt

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

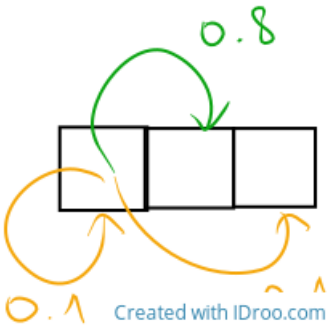
- Podstawy
  `https://docs.python.org/3/library/turtle.html#introduction`
- Zaawansowane rysowanie
  `https://blog.furas.pl/rysowanie-w-turtle-okrag-luk-kolo-i-elipsa.html`
- Rysowanie wypełnionych przeszkód
  `https://www.tutorialsandyou.com/python/how-to-draw-color-filled-shapes-in-python-turtle-17.html`

3) Zapoznajemy się z częścią teoretyczną (patrz Rozdział 0.1) oraz praktyczną w środowisku Turtle (patrze Rozdział 0.2) - opisującą działanie filtru Histogramowego,

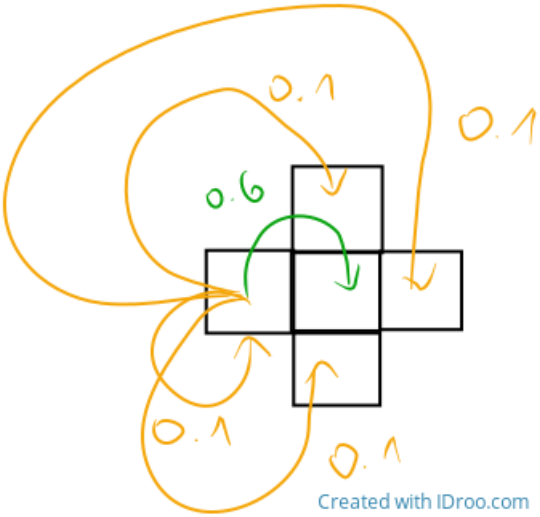
4) Modyfikujemy kod z pliku filtr1D.txt konstruując filtr 2D w wersji (i) oraz (ii) działający na mapie z pliku mapa_do-filtru2D.txt. Zapisujemy wynik pracy do pliku, filtr-2D.txt i udostępniamy prowadzącemu zajęcia.

Założenia ogólne do implementacji:
- robot jest wyposażony w kompas, czyli potrafi wykrywać kierunek swojego ruchu,
- pojedynczy ruch wykonujemy o jedno pole w kierunkach N(Północ), S(Południe), E(Wschód), W(Zachód),
- robot jest wyposażony w czujnik koloru, bada kolor kratek na mapie,
- świat 2D nie jest cykliczny, zakładamy, że mapa jest ograniczona ścianą, robot nie wyjedzie poza mapę,
- pamiętamy, że rozważamy ruch w czterech kierunkach co wymaga rozważenia czterech opcji ruchu w poniższych wariantach.

Wariant (i) (jako ćwiczenie 4):
- precyzja pomiaru wynosi 0.8
- robot dociera do celu z prawdopodobieństwem 0.8,
- pozostaje w swojej kratce z prawdopodobieństwem 0.1,
- trafia o jedną kratkę dalej z prawdopodobieństwem 0.1,



Wariant (ii) (jako ćwiczenie 5):
- precyzja pomiaru wynosi 0.9
- robot dociera do celu z prawdopodobieństwem 0.6,
- pozostaje w swojej kratce z prawdopodobieństwem 0.1,
- trafia o jedną kratkę dalej z prawdopodobieństwem 0.1,
- trafia o jedną kratkę w lewo od kratki docelowej z prawdopodobieństwem 0.1,
- trafia o jedną kratkę w prawo od kratki docelowej z prawdopodobieństwem 0.1,



5) Materiały dodatkowe dla chętnych - rozszerzenie wiedzy o filtracji Histogramowej:
`http://www.robotics.tu-berlin.de/fileadmin/fg170/Publikationen_pdf/Jonschkowski-16-NIPS-WS.pdf`

Zasady oceny zadania: Ćwiczenie składa się z dwóch pod-zadań (punkty 4(i), 4(ii)), każde oceniane w wymiarze 1 punktu.

0.1 Część teoretyczna - Lokalizacja za pomocą filtru Histogramowego

Algorytm pozwalający na lokalizację w danym środowisku opisanym mapą. Polega na naprzemiennym wykonywaniu pomiarów za pomocą sensorów i poruszaniu się, przy każdorazowym odświeżaniu prawdopodobieństw położenia na mapie.

Etap pomiaru za pomocą sensorów, może wyglądać następująco,

Przyjmując, że robot znajduje się w cyklicznym świecie postaci,

$$świat = [ściana, ściana, drzwi, drzwi, ściana]$$

Robot może się poruszać w prawo, jeżeli jest w ostatnim stanie w kolejnym kroku przechodzi do pierwszego.

Zakładając, że prawdopodobieństwo bycia w jednym z obszarów świata jest jednakowe, czyli

$$[0.2, 0.2, 0.2, 0.2, 0.2]$$

Przy ustalonej dokładności sensora 0.8, gdy robot w pierwszym pomiarze zarejestruje drzwi, prawdopodobieństwa położenia robota aktualizujemy następująco

$$[0.2 * 0.2, 0.2 * 0.2, 0.2 * 0.8, 0.2 * 0.8, 0.2 * 0.2]$$

czyli

$$[0.04, 0.04, 0.16, 0.16, 0.04]$$

Otrzymane wartości są prawdopodobieństwami nieznormalizowanymi, zauważmy, że ich suma wynosi 0.44, aby prawdopodobieństwa sumowały się do jedynki, dzielimy każdą wartość przez otrzymaną sumę otrzymując w przybliżeniu

$$[0.091, 0.091, 0.3635, 0.3635, 0.091]$$

teraz prawdopodobieństwa sumują się do jedynki.

W rzeczywistości, mechanizm odświeżania prawdopodobieństwa działa na zasadzie reguły Bayesa,

$$P(x_i|Pomiar) = \frac{P(Pomiar|x_i) * P(x_i)}{P(Pomiar)}$$

gdzie $P(Pomiar|x_i)$ jest prawdopodobieństwem pomiaru danej wartości (dokładnością pomiaru), $P(x_i)$ jest prawdopodobieństwem przed aktualizacją, P(Pomiar), jest sumą prawdopodobieństw każdej pozycji świata, wartością, którą normalizujemy prawdopodobieństwo.

Kolejnym etapem działania filtru, jest aktualizacja po wykonaniu ruchu, tak zwany splot (konwolucja). Prawdopodobieństwo porusza się wraz z robotem jest odświeżane w zależności od precyzji wykonanego ruchu. Przyjmując, że robot porusza się w sposób dokładny. Jeżeli prawdopodobieństwa przed aktualizacją mają wartość.

$$[0, 0, 0.5, 0.5, 0]$$

to po ruchu robota w prawo i aktualizacji wynoszą

$$[0, 0, 0, 0.5, 0.5]$$

W rzeczywistości roboty poruszają się w sposób niedokładny. Załóżmy przykładowo, że robot zamierza przemierzyć dwa pola świata w prawo i prawdopodobieństwo tego ruchu wynosi 0.8, robot może nie dotrzeć do celu wykonując jeden ruch z prawdopodobieństwem 0.1, oraz może przejść o jedną kratkę dalej od celu z prawdopodobieństwem 0.1. Przy takich założeniach po jednym ruchu, aktualizacja, dla rozkładu prawdopodobieństwa,

$$[1, 0, 0, 0, 0]$$

jest postaci,

$$[0, 0.1, 0.8, 0.1, 0]$$

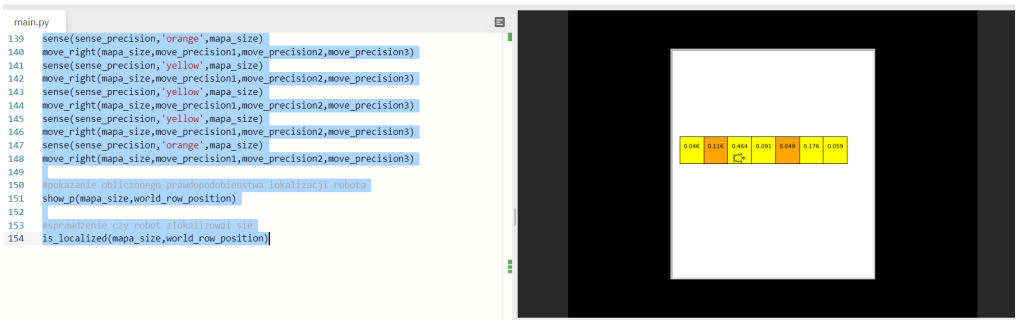
aktualizacja pola2, to przykładowo 0*0.1+0*0.8+1*0.1, czyli 0.1

Aktualizacja po wykonanym ruchu to w rzeczywistości wyliczanie prawdopodobieństwa całkowitego,

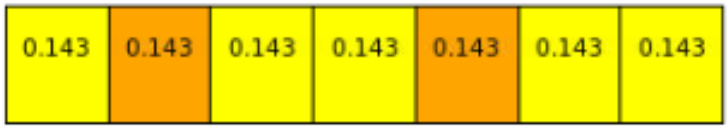
$$P(x_i) = \sum_i P(x_j) * P(x_i|x_j)$$

Oznacza prawdopodobieństwo bycia w komórce x_i po wykonaniu ruchu z komórek x_j z ustalonym prawdopodobieństwem.

0.2 Filtr Histogramowy w lokalizacji agenta w środowisku Turtle



Poniżej prezentujemy kod, który był omówiony na wykładzie. Przedstawmy funkcje przydatne przy filtracji Histogramowej, do jej symulacji w poniższym świecie cyklicznym,



Funkcja draw_box pozwala na rysowanie mapy.

```
#rysowanie pola mapy, na podstawie zdefiniowanego lewego gornego rogu
def draw_box(left_top_corner_x,left_top_corner_y,color):
    #szerokosc przeszkody
    obstacle_width=1;
    #wysokosc przeszkody
    obstacle_height=1;
    #ukrycie ikony agenta,
    turtle1.hideturtle()
    #wylaczenie rysowania
    turtle1.penup()
    turtle1.setx(left_top_corner_x*factor)
    turtle1.sety(left_top_corner_y*factor)
    #wlaczenie rysowania
    turtle1.pendown()
    #rysownie wypelnionego kolorem niebieskim prostokata,
    turtle1.fillcolor(color)
    turtle1.begin_fill()
    #ustawiamy poczatkowy kierunek na polnoc, polnoc jest skierowna w prawo,
    turtle1.setheading(0)
    turtle1.forward(obstacle_width*factor)
    turtle1.right(90)
    turtle1.forward(obstacle_height*factor)
    turtle1.right(90)
    turtle1.forward(obstacle_width*factor)
    turtle1.right(90)
    turtle1.forward(obstacle_height*factor)
    turtle1.right(90)
    turtle1.end_fill()
```

Funkcja show_p pozwala na wypisanie aktualnego prawdopodobieństwa na mapie. Nie odświeża kratek pol, stąd najlepiej ją używać na końcu kodu, ew. zmodyfikować odświeżanie.

```
#pokazanie aktualnego prawdopodobienstwa na mapie,
#do przerobienia gdy zmienimy wiersz wysowania mapy
def show_p(mapa_size,world_row_position):
  for i in range(0,mapa_size):
    turtle1.penup()
    temp=i+0.2
    turtle1.setx(temp*factor)
    turtle1.sety((world_row_position-0.5)*factor)
    turtle1.write(np.around(probability[i],decimals=3))
```

Funkcja sense to część pomiarowa filtru Histogramowego, zbieranie informacji z otoczenia za pomocą czyjnika koloru,

```
#update prawdopodobienstwa po pomiarze, step1
def sense(sense_precision,sense_temp,mapa_size):
  for i in range(0,mapa_size):
    if(mapa[i]==sense_temp):
      probability[i]=probability[i]*sense_precision
    else:
          probability[i]=probability[i]*(1.-sense_precision)
  #update prawdopodobienstwa po pomiarze, step2
  #normalizacja,
  suma=np.sum(probability)
  for i in range(0,mapa_size):
    probability[i]=probability[i]/suma
```

Funkcja move_right to część aktualizacji prawdopodobieństwa po ruchu, w tym konkretnym przypadku w prawo,

```
#pamietamy ze swiat jest cykluczny z 6 trafiamy do 0, za pomoca modulo,
def move_right(mapa_size,move_precision1,move_precision2,move_precision3):
  probability_apriori=probability.copy()
  for i in range(0,mapa_size):
    probability[i]=probability_apriori[i]*move_precision1
    probability[i]=probability[i]+probability_apriori[(i-1)%mapa_size]*
        move_precision2
    probability[i]=probability[i]+probability_apriori[(i-2)%mapa_size]*
        move_precision3
```

Funkcja find_max to funkcja do szukania maksymalnego prawdopodobieństwa na mapie,

```
def find_max(mapa_size):
  #szukam max,
  temp_max=0
  #numer pola z wartoscia max
  max_index=0
  for i in range(0,mapa_size):
    if(probability[i]>temp_max):
      temp_max=probability[i]
      max_index=i
  return max_index
  print('temp_max = ',temp_max)
```

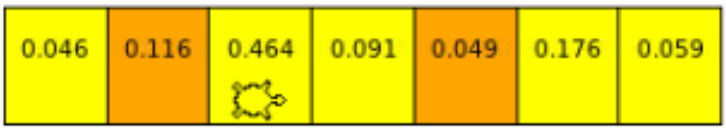
Funkcja show_localized_agent pokazuje agenta na mapie, gdy istnieje tylko jedna pozycja najbardziej prawdopodobna,

```
def show_localized_agent(max_index):
    for i in range(0,mapa_size):
      turtle1.penup()#wylaczenie rysowania
      turtle1.setx((max_index+0.4)*factor)
      turtle1.sety((world_row_position-0.8)*factor)
     #turtle1.write('here')
    turtle1.showturtle()
    print('Agent localized itself')
    #arrow, turtle, circle, square, triangle,classic
    turtle1.shape("turtle")
```

Funkcja is_localized sprawdza czy robot został zlokalizowany, używa funkcji find_max show_localized_agent

```
def is_localized(mapa_size,world_row_position):
  max_index=find_max(mapa_size)
  temp_max=probability[max_index]
  #sprawdzam czy max jest tylko jeden, jezeli tak wstawiam ikone agenta,
  count=0
  for i in range(0,mapa_size):
    if probability[i]==temp_max:
      count=count+1
  if count==1:
    show_localized_agent(max_index)
```

Przejdźmy do demonstracji filtru, po wykonaniu poniższego kodu, robot zlokalizuje się i otrzymamy następujący wynik na mapie,




```
#PROGRAM GLOWNY
#rysowani przestszeni, na podstawie lewej dolnej i prawej górnej koordynaty
drawing_area=Screen()
setworldcoordinates(-20, -20, 1200, 1200)

turtle1=Turtle() #definicja agenta,
turtle1.hideturtle()#ukrycie ikony agenta,
turtle2=Turtle() #definicja agenta,
turtle2.hideturtle()#ukrycie ikony agenta,

#liczba pixeli na jednostkę,
factor=150

turtle1.speed(10) #szybkosc rysowania od 1 do 10,

#definicja swiata, jest cykliczny, ruch tylko w prawo,
#przechodzac z pola 6 w prawo trafiamy do 0,
mapa=['yellow','orange','yellow','yellow','orange','yellow','yellow']
mapa_size=len(mapa)

#wizualizacja swiata
```

```
world_row_position=5
for i in range(0,mapa_size):
  draw_box(i,world_row_position,mapa[i])

#inicjacja prawdopodobienstwa
probability=[]
for i in range(0,mapa_size):
  probability.append(1/mapa_size)

#prezentacja finalnego filra histogramowego
sense_precision=0.8
move_precision1=0.1#pozostanie w miejscu
move_precision2=0.8#trafiamy do kratki docelowej
move_precision3=0.1#trafiamy do kratki kolejnej
sense(sense_precision,'orange',mapa_size)
move_right(mapa_size,move_precision1,move_precision2,move_precision3)
sense(sense_precision,'yellow',mapa_size)
move_right(mapa_size,move_precision1,move_precision2,move_precision3)
sense(sense_precision,'yellow',mapa_size)
move_right(mapa_size,move_precision1,move_precision2,move_precision3)
sense(sense_precision,'yellow',mapa_size)
move_right(mapa_size,move_precision1,move_precision2,move_precision3)
sense(sense_precision,'orange',mapa_size)
move_right(mapa_size,move_precision1,move_precision2,move_precision3)

#pokazanie obliczonego prawdopodobienstwa lokalizacji robota
show_p(mapa_size,world_row_position)

#sprawdzenie czy robot zlokalizowal sie
is_localized(mapa_size,world_row_position)
```

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