1 Cel programu

Celem programu jest automatyczne wyznaczenie pozycji punktu podając zasięgi z 6 skanerów UWB, obliczenie uchybu względem rzeczywistego położenia punktu pomiarowego, oraz graficzne przedstawienie trilateracji w układzie współrzędnych. Program został napisany w języku Python.

2 Klasy

2.1 Klasa punktu skanującego

Utworzona został klasa punktu skanującego:

```
class Scanner():
    """Prezentacja skanera"""

    def __init__(self, xPosition = 0, yPosition = 0, diameter = 10):

        self.position = {'x': 0, 'y': 0}

        self.position['x'] = xPosition
        self.position['y'] = yPosition
        self.diameter = diameter

    def writePosition(self, xPosition, yPosition):
        self.position['x'] = xPosition
        self.position['y'] = yPosition

    def writeDiameter(self, diameter):
        self.diameter = diameter

    def readPosition(self):
        print(f"position: {self.position['x']},{self.position['y']}")
        return self.position
```

```
    def readDiameter(self):
        print(f"Diameter: {self.diameter} cm")
```

Metoda __init__() tworzy instancję punktu skanującego. W argumentach przyjmuje swoją pozycję(x,y) oraz startowy promień zasięgu.

Tworzone są również odpowiadające im atrybuty, do których zapisane będą wartości podane w argumentach.

Metody writePosition() oraz writeDiameter() pozwalają na aktualizację atrybutów. Odpowiadające im metody read pozwalają odczytać aktualne wartości atrybutów.

2.2 Klasa punktu lokalizowanego

Utworzona została klasa punktu lokalizowanego

```
class MeasuringPoint():
    """prezentacja punktu lokalizowanego"""
    def __init__(self, xPositionActual = 0, yPositionActual = 0):

        self.positionActual = {'x': xPositionActual, 'y': yPositionActual}



        self.distance = [0, 0, 0]

        #utwórz słownik przechowujący 3 wyliczone pozycje
        positionCalculated1 = {'x': 0, 'y': 0}
        positionCalculated2 = {'x': 0, 'y': 0}
        positionCalculated3 = {'x': 0, 'y': 0}



        self.positionCalculated = [positionCalculated1, positionCalculated2, positionCalculated3]

        # utwórz zmienną przechowującą numer najbardziej poprawnego pomiaru
        self.correctMeasurement = ''



    def readActualPosition(self):
        """Zwraca aktualną pozycję"""
        return self.positionActual



    def writeActualPosition(self, x, y):
        """zapisuje rzeczywistą pozycję punktu lokalizowanego"""

        self.positionActual['x'] = x
        self.positionActual['y'] = y
```

```
def calculateOffense(self):
    self.shiftX = math.fabs(self.positionCalculated[self.correctMeasurement]['x'] - self.positionActual['x'])
    self.shiftY = math.fabs(self.positionCalculated[self.correctMeasurement]['y'] - self.positionActual['y'])

    self.squareX = self.shiftX**2
    self.squareY = self.shiftY**2

    self.sum = self.squareX + self.squareY

    self.offense = round(math.sqrt(self.sum))

    print(f"Uchyb = {self.offense} cm")

def calculateDistanceFrom(self, xPoint, yPoint, numberOfPoint):
    """odczytuje dystans od podanego punktu"""
    self.shiftX = math.fabs(self.positionCalculated[numberOfPoint]['x'] - xPoint)
    self.shiftY = math.fabs(self.positionCalculated[numberOfPoint]['y'] - yPoint)

    self.squareX = self.shiftX**2
    self.squareY = self.shiftY**2

    self.sum = self.squareX + self.squareY

    self.distance[numberOfPoint] = round(math.sqrt(self.sum))

    prompt = ''

    if numberOfPoint == 0:
        prompt = "1:>   "
    elif numberOfPoint == 1:
        prompt = "2:>   "
    elif numberOfPoint == 2:
        prompt = "3:>   "

    print(f"{prompt}{self.distance[numberOfPoint]}")
```

Metoda __init__() tworzy instancję punktu lokalizowanego. Jako argumenty przyjmuje jego rzeczywistą pozycję w układzie współrzędnych, która zostaje zapisana w słowniku z atrybutami przechowującymi odpowiadające im wartości.

Utworzony zostaje atrybut self.distance - dystansu od wybranego punktu w układzie współrzędnych. Ma on postać listy zawierającej 3 wartości-będą to odległości od przyjętego środka układu współrzędnych.

W kolejnym kroku zostają utworzone 3 atrubuty positionCalculated w postaci słowników. Każdy z nich przechowuje pozycję wyliczoną z jednej kombinacji punktów skanujących.

Następnie utworzona zostaje lista zawierająca 3 powyższe słowniki, oraz atrubut self.correctMeasurement służący do przechowywania najbardziej prawdopodobnej pozycji. Docelowo będzie tu przeniesiony jeden ze słowników positionCalculated.

Metoda writeActualPosition() pozwala na modyfikację rzeczywistej pozycji punktu lokalizowanego.

Metoda calculateOffense() przelicza i zapisuje do listy self.distance dystans pomiędzy rzeczywistą pozycją punktu, a najbardziej prawdopodobnym położeniem wyliczonym, którego numer będzie przechowywany w self.correctMeasurement.

Metoda calculateDistanceFrom() ma za zadanie obliczenie oraz zapisanie dystansu punktu lokalizowanego od punktu o współrzędnych podanych jako argumenty metody. Przyjmuje także argument indeksu dystansu, dzięki czemu może zapisać wynik do jednego z 3 elementów listy self.distance.

3 Instancje punktów

Utworzone zostają instancje punktów skanujących oraz punktów lokalizowanych. Do poszczególnych instancji zostają wpisane rzeczywiste pozycje. Ustalony zostaje także umowny środek układu.

```
# ustaw punkt centralny układu współrzędnych
xCenterPoint = 82.5
# yCenterPoint = -260
yCenterPoint = -350

# utwórz punkty pomiarowe
point1 = MeasuringPoint()
point2 = MeasuringPoint()
point3 = MeasuringPoint()
point4 = MeasuringPoint()
point5 = MeasuringPoint()
point6 = MeasuringPoint()

# utwórz skanery
scannerA = Scanner()
scannerB = Scanner()
scannerC = Scanner()
scannerD = Scanner()
scannerE = Scanner()
scannerF = Scanner()

# wpisz rzeczywiste pozycje punktów pomiarowych(w celu obliczenia błędu)
points = [point1, point1, point2, point3, point4, point5, point6]

points[1].writeActualPosition(82.5, 0)
```

```
points[2].writeActualPosition(82.5, -120)
points[3].writeActualPosition(165, -190)
points[4].writeActualPosition(0, -260)
points[5].writeActualPosition(0, -400)
points[6].writeActualPosition(82.5, -490)

# wpisz pozycje punktów pomiarowych
scannerA.writePosition(0, 0)
scannerB.writePosition(165, 0)
scannerC.writePosition(82.5, -260)
scannerD.writePosition(0, -450)
scannerE.writePosition(165, -450)
scannerF.writePosition(82.5, -600)
```

4 Funkcje trilateracji

4.1 Trilateracja 3 punktowa

Utworzona została funkcja licząca trilaterację 3 punktową opisaną w raporcie 3_CZB_005, rozdział 3.

```
def calculatePosition(scannerA, scannerB, scannerC, point, numberOfPoint):
    """funkcja wykonująca trilaterację 3 punktową"""

    factorA = 2*scannerB.position['x'] - 2*scannerA.position['x']
    factorB = 2*scannerB.position['y'] - 2*scannerA.position['y']
    factorC = scannerA.diameter**2 - scannerB.diameter**2 - scannerA.position['x']**2\
            +scannerB.position['x']**2 - scannerA.position['y']**2 + scannerB.position['y']**2
    factorD = 2*scannerC.position['x'] - 2*scannerB.position['x']
    factorE = 2*scannerC.position['y'] - 2*scannerB.position['y']
    factorF = scannerB.diameter**2 - scannerC.diameter**2 - scannerB.position['x']**2\
            +scannerC.position['x']**2 - scannerB.position['y']**2 + scannerC.position['y']**2

    point.positionCalculated[numberOfPoint]['x'] = round((factorC * factorE - factorF * factorB)\
                        /(factorE * factorA - factorB * factorD))

    point.positionCalculated[numberOfPoint]['y'] = round((factorC * factorD - factorA * factorF)\
                        /(factorB * factorD - factorA * factorE))

    prompt = ''

    #dodatkowe punkty
    if numberOfPoint == 0:
        prompt = "1:>   "
    elif(numberOfPoint == 1):
        prompt = "2:>   "
    elif(numberOfPoint == 2):
        prompt = "3:>   "
```

```
print(f"{prompt}x: {point.positionCalculated[numberOfPoint]['x']},"
      f" y: {point.positionCalculated[numberOfPoint]['y']}")
```

Jako parametry przyjmuje instancje trzy obiekty skanerów, obiekt punktu lokalizowanego oraz numer trilateracji. Obliczoną pozycję zapisuje w atrybucie punktu lokalizowanego positionCalculated o numerze pozycji podanym w argumencie.

4.2 Trilateracja 6 punktowa

Utworzona została funkcja która ustala najbardziej prawdopodobną pozycję punktu lokalizowanego opierając się o 3 pozycje wyznaczone przez wywoływanie funkcji trilateracji 3 punktowej, przy użyciu 3 kombinacji punktów skanujących.

```
def calculatePosition6(scannerA, scannerB, scannerC, scannerD, scannerE, scannerF, point, xCenterPoint,
yCenterPoint):
    """funkcja wykonująca trilaterację przy użyciu 6 skanerów metodą punktu najbliższego środkowi łodzi"""
    print("\r\nPrawdopodobne pozycje:")
    calculatePosition(scannerA, scannerB, scannerC, point, 0)
    calculatePosition(scannerC, scannerD, scannerE, point, 1)
    calculatePosition(scannerD, scannerE, scannerF, point, 2)



    print("\r\nOdległości od środka:")
    point.calculateDistanceFrom(xCenterPoint, yCenterPoint, 0)
    point.calculateDistanceFrom(xCenterPoint, yCenterPoint, 1)
    point.calculateDistanceFrom(xCenterPoint, yCenterPoint, 2)

    #sprawdź który pomiar zwrócił najmniejsyz dystans od środka
    minimal = point.distance.index(min(point.distance))

    #wpisz numer poprawnego pomiaru do obiektu
    point.correctMeasurement = minimal



    position = point.positionCalculated[minimal]

    print(f"\r\nNajbardziej prawdopodobne położenie punktu to: {position}")
```

Funkcja przyjmuje jako argumenty 6 obiektów skanerów, punkt lokalizowany oraz umowny środek układu współrzędnych.

Na początku liczone są 3 prawdopodobne położenia punktu obliczone w wykorzystaniem kombinacji punktów skanujących ABC, CDE oraz DEF.

Następnie zostają obliczone odległości od umownego środka układu przy użyciu metody calculateDistanceFrom().

Pozycja najbliższa środkowi układu zostaje podana strumieniem wyjściowym. Numer pomiaru zostaje zapisany w atrybucie correctMeasurement punktu lokalizowanego.

5 Główny moduł programu

Główny moduł programu wygląda następująco:

```
from points import *
import trilateration
import draw
import json



diametersFile = 'diameters.json'
diameters = ''

measuringPointsPositonsFile = 'measuringPointsPositions.json'
measuringPointsPositions = ''

scannersPositionsFile = 'scannersPositions.json'
scannersPositions = ''

# pozycje skanerów
answer = input("czy chcesz załadować pozycje skanerów z pliku? t/n \r\n> ")
if answer == 't' or answer == 'y':

    try:
        with open(scannersPositionsFile) as f:
            scannersPositions = json.load(f)
            scannerA.positionActual = scannersPositions[0]
            scannerB.positionActual = scannersPositions[1]
            scannerC.positionActual = scannersPositions[2]
            scannerD.positionActual = scannersPositions[3]
            scannerE.positionActual = scannersPositions[4]
            scannerF.positionActual = scannersPositions[5]

    except FileNotFoundError:
        print("nie znaleziono pliku. Wpisz ręcznie pozycje skanerów:\r\n")

        scannerA.writePosition(int(input("skanerA x> ")), int(input("skanerA y> ")))
        scannerB.writePosition(int(input("skanerB x> ")), int(input("skanerB y> ")))
        scannerC.writePosition(int(input("skanerC x> ")), int(input("skanerC y> ")))
        scannerD.writePosition(int(input("skanerD x> ")), int(input("skanerD y> ")))
        scannerE.writePosition(int(input("skanerE x> ")), int(input("skanerE y> ")))
        scannerF.writePosition(int(input("skanerF x> ")), int(input("skanerF y> ")))



        answer = input("zapisać pozycje skanerów w pliku? t/n \r\n> ")
```

```
        if answer == 't' or answer == 'y':
            scannersPositions = [scannerA.readPosition(), scannerB.readPosition(), scannerC.readPosition(),
                                 scannerD.readPosition(), scannerE.readPosition(), scannerF.readPosition()]

            with open(scannersPositionsFile, 'w') as f:
                json.dump(scannersPositions, f)
                print("zapisano pozycje skanerów")

else:
    scannerA.writePosition(int(input("skanerA x> ")), int(input("skanerA y> ")))
    scannerB.writePosition(int(input("skanerB x> ")), int(input("skanerB y> ")))
    scannerC.writePosition(int(input("skanerC x> ")), int(input("skanerC y> ")))
    scannerD.writePosition(int(input("skanerD x> ")), int(input("skanerD y> ")))
    scannerE.writePosition(int(input("skanerE x> ")), int(input("skanerE y> ")))
    scannerF.writePosition(int(input("skanerF x> ")), int(input("skanerF y> ")))


    answer = input("zapisać rzeczywiste pozycje w pliku? t/n \r\n> ")

    if answer == 't' or answer == 'y':
        scannersPositions = [scannerA.readPosition(), scannerB.readPosition(), scannerC.readPosition(),
                             scannerD.readPosition(), scannerE.readPosition(), scannerF.readPosition()]

        with open(measuringPointsPositonsFile, 'w') as f:
            json.dump(measuringPointsPositions, f)
            print("zapisano pozycje skanerów")

# pozycje punktów lokalizowanych
answer = input("czy chcesz załadować pozycje punktów lokalizowanych z pliku? t/n \r\n> ")
if answer == 't' or answer == 'y':

    try:
        with open(measuringPointsPositonsFile) as f:
            measuringPointsPositions = json.load(f)
            point1.positionActual = measuringPointsPositions[0]
            point2.positionActual = measuringPointsPositions[1]
            point3.positionActual = measuringPointsPositions[2]
            point4.positionActual = measuringPointsPositions[3]
            point5.positionActual = measuringPointsPositions[4]
            point6.positionActual = measuringPointsPositions[5]

    except FileNotFoundError:
        print("nie znaleziono pliku. Wpisz ręcznie pozycje punktów lokalizowanych:\r\n")

        point1.writeActualPosition(float(input("punkt1 x> ")), float(input("punkt1 y> ")))
        point2.writeActualPosition(float(input("punkt2 x> ")), float(input("punkt2 y> ")))
        point3.writeActualPosition(float(input("punkt3 x> ")), float(input("punkt3 y> ")))
        point4.writeActualPosition(float(input("punkt4 x> ")), float(input("punkt4 y> ")))
        point5.writeActualPosition(float(input("punkt5 x> ")), float(input("punkt5 y> ")))
        point6.writeActualPosition(float(input("punkt6 x> ")), float(input("punkt6 y> ")))


        answer = input("zapisać pozycje punktów w pliku? t/n \r\n> ")

        if answer == 't' or answer == 'y':
            measuringPointsPositions = [point1.readActualPosition(), point2.readActualPosition(), point3.readActualPosi-
tion(),
                                        point4.readActualPosition(), point5.readActualPosition(), point6.readActualPosition(), ]
```

```
        with open(measuringPointsPositonsFile, 'w') as f:
            json.dump(measuringPointsPositions, f)
            print("zapisano pozycje puntków")

else:
    point1.writeActualPosition(int(input("punkt1 x> ")), int(input("punkt1 y> ")))
    point2.writeActualPosition(int(input("punkt2 x> ")), int(input("punkt2 y> ")))
    point3.writeActualPosition(int(input("punkt3 x> ")), int(input("punkt3 y> ")))
    point4.writeActualPosition(int(input("punkt4 x> ")), int(input("punkt4 y> ")))
    point5.writeActualPosition(int(input("punkt5 x> ")), int(input("punkt5 y> ")))
    point6.writeActualPosition(int(input("punkt6 x> ")), int(input("punkt6 y> ")))


    answer = input("zapisać rzeczywiste pozycje w pliku? t/n \r\n> ")

    if answer == 't' or answer == 'y':
        measuringPointsPositions = [point1.readActualPosition(), point2.readActualPosition(), point3.readActualPosi-
tion(),
                                    point4.readActualPosition(), point5.readActualPosition(), point6.readActualPosition(), ]

        with open(measuringPointsPositonsFile, 'w') as f:
            json.dump(measuringPointsPositions, f)
            print("zapisano pozycje punktów pomiarowych")


# podaj numer punktu do określenia pozycji:
pointNumber = int((input("Punkt pomiarowy > ")))

# średnice
answer = input("czy chcesz załadować promienie z pliku? t/n \r\n> ")
if answer == 't' or answer == 'y':

    try:
        with open(diametersFile) as f:
            diameters = json.load(f)
            scannerA.writeDiameter(diameters[0])
            scannerB.writeDiameter(diameters[1])
            scannerC.writeDiameter(diameters[2])
            scannerD.writeDiameter(diameters[3])
            scannerE.writeDiameter(diameters[4])
            scannerF.writeDiameter(diameters[5])

    except FileNotFoundError:
        print("nie znaleziono pliku. Wpisz ręcznie promienie okręgów:\r\n")
        # wpisz odległości zarejestrowane przez punkty pomiarowe)
        scannerA.writeDiameter(int(input("Promień A> ")))
        scannerB.writeDiameter(int(input("Promień B> ")))
        scannerC.writeDiameter(int(input("Promień C> ")))
        scannerD.writeDiameter(int(input("Promień D> ")))
        scannerE.writeDiameter(int(input("Promień E> ")))
        scannerF.writeDiameter(int(input("Promień F> ")))

        answer = input("zapisać promienie okręgów w pliku? t/n \r\n> ")

        if answer == 't' or answer == 'y':
            diameters = [scannerA.diameter, scannerB.diameter, scannerC.diameter, scannerD.diameter,
                         scannerE.diameter, scannerF.diameter]

            with open(diametersFile, 'w') as f:
                json.dump(diameters, f)
```

```
            print("zapisano promienie")

else:
    scannerA.writeDiameter(int(input("Promień A> ")))
    scannerB.writeDiameter(int(input("Promień B> ")))
    scannerC.writeDiameter(int(input("Promień C> ")))
    scannerD.writeDiameter(int(input("Promień D> ")))
    scannerE.writeDiameter(int(input("Promień E> ")))
    scannerF.writeDiameter(int(input("Promień F> ")))

    answer = input("zapisać promienie okręgów w pliku? t/n \r\n> ")

    if answer == 't' or answer == 'y':
        diameters = [scannerA.diameter, scannerB.diameter, scannerC.diameter, scannerD.diameter,
                     scannerE.diameter, scannerF.diameter]

        with open(diametersFile, 'w') as f:
            json.dump(diameters, f)
            print("zapisano promienie")



# wykonaj trilaterację 6 punktową
trilateration.calculatePosition6(scannerA, scannerB, scannerC,
                                 scannerD, scannerE, scannerF,
                                 points[pointNumber], xCenterPoint, yCenterPoint)

points[pointNumber].calculateOffense()

running = True

# print("wciśnij enter aby zakończyć > ")

while running:
    draw.drawCircles(points[pointNumber])
```

Główny moduł importuje klasy punktów, funkcje trilateracji oraz funkcje rysujące.

Następnie prosi użytkownika o podanie bądź wczytanie z pliku pozycji punktów lokalizowanych, skanerów numeru lokalizowanego punktu oraz promieni.

W kolejnym kroku moduł wykonuje trilaterację 6 punktową, oraz oblicza uchyb pomiędzy najbardziej prawdopodobną pozycją punktu lokalizowanego, a jego rzeczywistym położeniem. Wynik zostaje podany na strumień wyjściowy.

Następnie moduł uruchamia pętlę w której wykonuje funkcję rysującą graficzne przedstawienie układu.

6 Rysowanie układu

W programie została użyta biblioteka turtle umożliwiająca proste rysowanie geometrii metodą „żółwia”.

Utworzone zostały funkcje pomocnicze, rysujące okrąg, krzyżyk oraz linię:

```
def cross(xPosition, yPosition, radius, color):
    xPosition = xPosition / divider
    yPosition = yPosition / divider
    radius = radius / divider

    pen.penup()
    pen.goto(xPosition, yPosition)
    pen.pendown()
    pen.color(color)
    pen.goto(xPosition + radius, yPosition)
    pen.goto(xPosition - radius, yPosition)
    pen.goto(xPosition, yPosition)
    pen.goto(xPosition, yPosition+radius)
    pen.goto(xPosition, yPosition-radius)
    pen.penup()

def circle(xPosition, yPosition, radius, color):
    xPosition = xPosition / divider
    yPosition = yPosition / divider
    radius = radius/divider

    pen.penup()
    pen.goto(xPosition, yPosition-radius)
    pen.pendown()
    pen.color(color)
    pen.circle(radius)
    pen.penup()

def line(xStart, yStart, xEnd, yEnd, color):
    xStart = xStart / divider
    yStart = yStart / divider
    xEnd = xEnd / divider
    yEnd = yEnd / divider

    pen.penup()
    pen.goto(xStart, yStart)
    pen.pendown()
    pen.color(color)
    pen.goto(xEnd, yEnd)
    pen.penup()
```

została także utworzona głowna funkcja rysująca, wywoływana w module głownym:

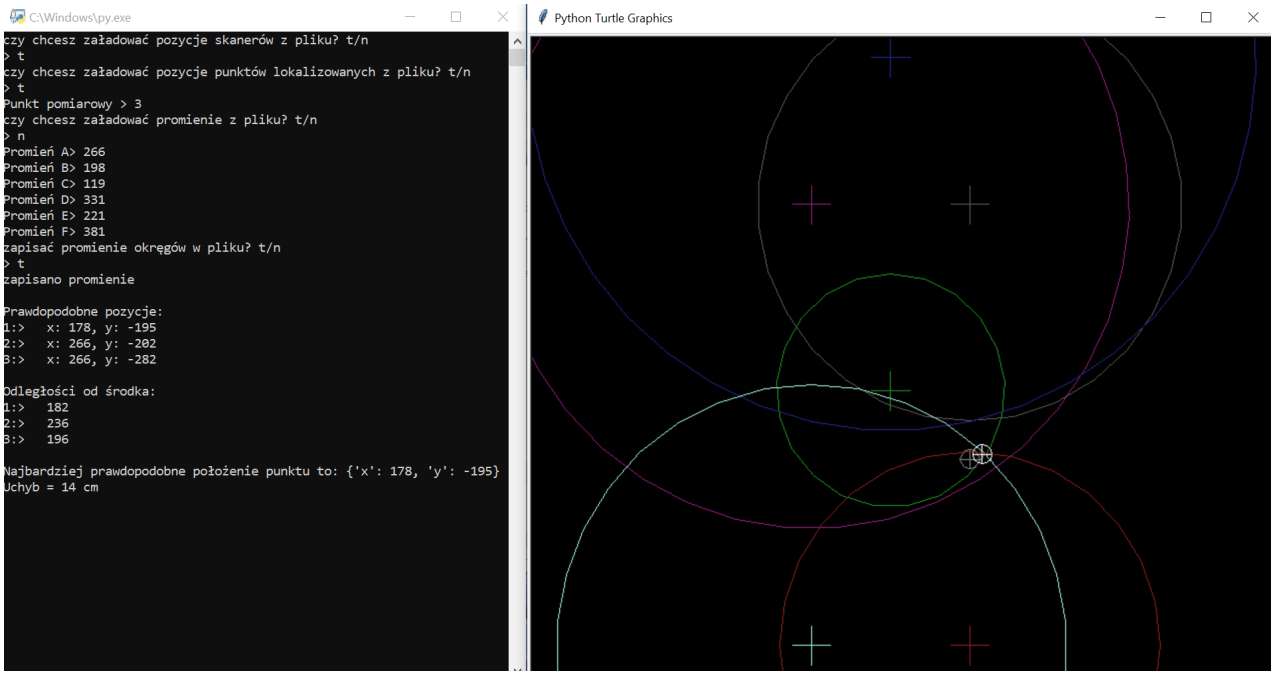
```
pen = turtle.Turtle()



def drawCircles(point):

    pen.hideturtle()
    pen.speed(0)
```

```
window = turtle.Screen()
window.bgcolor("#000000")
# window.screensize(800, 800)
window.setworldcoordinates(-140, 10, 240, -310)
window.tracer(100)

# window.setup(500, 500, None, 100)

pen.color("#ffffff")
pen.penup()
pen.goto(10, 10)
pen.pendown()

#tutaj rysuj

cross(point.positionActual['x'], point.positionActual['y'], 10, "grey")
circle(point.positionActual['x'], point.positionActual['y'], 10, "grey")

cross(point.positionCalculated[point.correctMeasurement]['x'],
    point.positionCalculated[point.correctMeasurement]['y'], 10, "white")

circle(point.positionCalculated[point.correctMeasurement]['x'],
     point.positionCalculated[point.correctMeasurement]['y'], 10, "white")

cross(scannerA.position['x'], scannerA.position['y'], 20, '#98ebd7')
cross(scannerB.position['x'], scannerB.position['y'], 20, '#891a1a')
cross(scannerC.position['x'], scannerC.position['y'], 20, '#1c891a')
cross(scannerD.position['x'], scannerD.position['y'], 20, '#891a7a')
cross(scannerE.position['x'], scannerE.position['y'], 20, '#525252')
cross(scannerF.position['x'], scannerF.position['y'], 20, '#212183')

circle(scannerA.position['x'], scannerA.position['y'], scannerA.diameter, '#98ebd7')
circle(scannerB.position['x'], scannerB.position['y'], scannerB.diameter, '#891a1a')
circle(scannerC.position['x'], scannerC.position['y'], scannerC.diameter, '#1c891a')
circle(scannerD.position['x'], scannerD.position['y'], scannerD.diameter, '#891a7a')
circle(scannerE.position['x'], scannerE.position['y'], scannerE.diameter, '#525252')
circle(scannerF.position['x'], scannerF.position['y'], scannerF.diameter, '#212183')


pen.getscreen().update()
```

Funkcja zaznacza na ekranie krzyżykami-punkty skanujące, okręgami- zasięgi punktów skanujących, kółkiem z wpisanym krzyżykiem białego koloru- obliczoną najbardziej prawdopodobną pozycję punktu lokalizowanego, oraz kółkiem z wpisanym krzyżykiem szarego koloru- pozycję rzeczywistą punktu lokalizowanego.

7 Działanie programu

Zostało przedstawione na poniższych zrzutach ekranu. Program został wykorzystany raporcie 3_CZB_005, rozdział 7.



Wykonał: Bartosz Pracz