ĆWICZENIE 2

STEROWANIE ROBOTEM MOBILNYM KHEPERA PRZY POMOCY LOGIKI ROZMYTEJ - ZADANIE OMIJANIA PRZESZKÓD

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się ze sposobem syntezy systemu ekspertowego pełniącego rolę regulatora. Rozmyty system ekspertowy zostanie użyty do sterowania małym robotem mobilnym Khepera realizującym zadanie omijania przeszkód.

Laboratorium składa się z trzech zasadniczych części. Część I ma na celu zapoznanie się z budową oraz ze sposobem sterowania robotem mobilnym Khepera III za pomocą dedykowanych funkcji języka Python. W części II zostanie zaprojektowany system ekspertowego typu Mamdaniego z wykorzystaniem biblioteki **scikit-learn** realizujący zadanie **omijania przeszkód**. Część III laboratorium polega na praktycznej weryfikacji poprawności zrealizowanego systemu w sterowaniu rzeczywistym robotem.

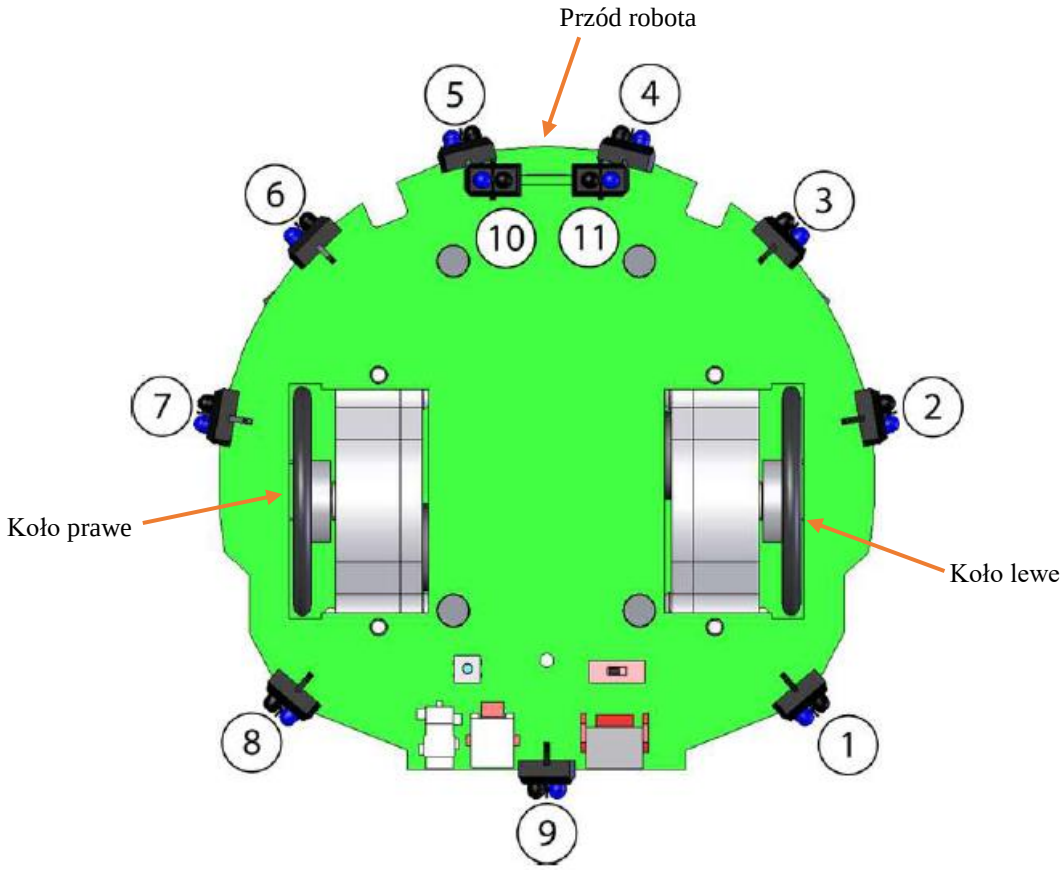
Część I: Robot Khepera III

Budowa

Najpełniejszym opisem robota Khepera III jest instrukcja użytkownika zamieszczona pod adresem [1].

Rys. 1. Robot Khepera III

Robot Khepera III (Rys. 1) jest cylindrycznym robotem mobilnym o średnicy 13 cm i dwóch kołach napędowych umieszczonych w odległości $2R$ = 88 mm. Na obwodzie robota rozmieszczono dziewięć czujników zbliżeniowych (Rys. 2) pozwalających na lokalizację przeszkód. Dzięki niewielkim wymiarom, małej wadze i zestawowi czujników zbliżeniowych robot Khepera umożliwia testowanie w warunkach laboratoryjnych szeregu algorytmów sterowania.



Rys. 2. Rozmieszczenie czujników IR - Widok od spodu robota Khepera III [1]

Oprogramowanie

Robot Khepera III został wyposażony w zestaw kilkudziesięciu poleceń (dodatek A pozycji [1]) pozwalających na jego elementarną obsługę za pomocą łącza RS232/Bluetooth (rozdział 4.4.1 [1]) i terminala VT100. Celem zapewnienia obsługi robota z poziomu konsoli Pythona stworzono dedykowany zestaw funkcji implementujący wybrane polecenia z dodatku A pozycji [1]. Polecenia te zostały zebrane w bibliotekę khepera3.py. Poniżej przedstawiono tabelaryczne zestawienie zaimplementowanych funkcji (Tab. 1).

Tabela 1. Zestawienie funkcji biblioteki khepera3.py. Definicje funkcji zamieszczono w dodatku I na końcu instrukcji

Nazwa funkcji	**Opis**
k3Init	Inicjalizacja łącza Bluetooth. Polecenie zwraca „uchwyt” programowy do robota.
k3GetBateryState	Wyświetlenie informacji o stopniu naładowania baterii. Poziom w pełni naładowanej baterii to 84%.
k3ReadProximitySensors	Odczyt wartości sygnałów czujników zbliżeniowych oraz przeskalowanie czujników 2, 5 oraz 9
k3ReadAmbientSensors	Odczyt wartości sygnałów czujników pracujących w trybie pomiaru światła zewnętrznego. Polecenie użyteczne przy realizacji zadania podążania do źródła światła
k3ReadSpeed	Odczyt prędkości poszczególnych kół
k3ReadSoftwareVersion	Odczyt wersji oprogramowania EEPROM
k3SetSpeed	Ustawienie prędkości poszczególnych kół
k3Stop	Zatrzymanie robota

k3SetPosition	Ustawienie początkowej liczby impulsów poszczególnych liczników kół. Jeden impuls odpowiada odległości 0,047mm. Typowo polecenie służy do zerowania stanu liczników kół (enkoderów).
k3SetTargetPosition	Sterowanie robotem w trybie pozycyjnym. Zadane zostają bezwzględne wartości odległości, którą mają osiągnąć poszczególne koła. Jeden impuls odpowiada odległości 0,047mm.
k3ReadPosition	Odczyt liczby impulsów z enkoderów kół.
k3Braitenberg	Uruchomienie wbudowanego algorytmu Braitenberga omijania przeszkód. mode=0 - Infrared sensors, mode==1 - Ultrasonic sensors, mode=2 - Stop Braitenberg mode
k3ReadProximitySensorsLoop	Odczyt sygnałów z czujników zbliżeniowych przez ustaloną liczbę cykli (domyślnie 1000). Stosowane do sporządzenia charakterystyki czujników.
k3SetProgrammableLed	Obsługa 2 programowanych diód. Stosowane do sygnalizacji zakończenia programu lub zaistnienia błędu jego wykonania.

Przebieg części I laboratorium

Przykład sesji pracy z wykorzystaniem poleceń do sterowania robotem mobilnym Khepera III.

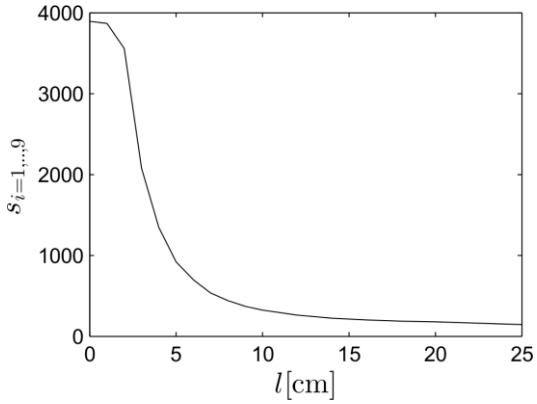
```
Uruchomić plik khepera3.py

Utworzyć „uchwyt programowy” do robota za pomocą funkcji k3Init
>>>s=k3Init()
COM3

>>>k3GetBateRys.tate(s)
Batery state: 57%

>>>k3ReadProximitySensors(s)
Out[9]: array([159, 279,  85,  72,  58,  43,  83, 235, 139, 279, 231])

>>>k3ReadAmbientSensors(s)
Out[10]: array([4023, 4003, 4035, 4040, 4051, 4059, 4047, 4011, 4010, 4023, 4021])

>>>k3ReadSpeed(s)
Out[11]: [0, 0]

>>>k3SetSpeed(s,2000,2000)

>>>k3Stop(s)

>>>k3SetPosition(s,0,0)

>>>k3SetTargetPosition(s,6000,6000)
Out[16]: [5951, 5951]

>>>k3ReadPosition(s)
Out[17]: [5829, 5949]

>>>k3Braitenberg(s,0)

>>>k3Braitenberg(s,2)

>>>k3SetProgrammableLed(s,1,1)

>>>k3SetProgrammableLed(s,0,1)

>>>k3SetProgrammableLed(s,0,0)

>>>k3SetProgrammableLed(s,1,0)
```

Część II: Zadanie omijania przeszkód

Wprowadzenie

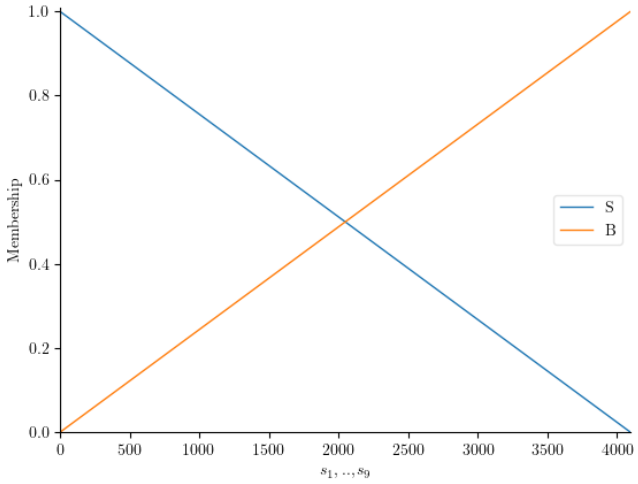
Realizacja zadania omijania przeszkód polega na zaproponowaniu takiego rozmytego systemu ekspertowego, który na podstawie sygnałów z czujników zbliżeniowych będzie sterował kołami robota celem unikania kontaktu z przeszkodami.

Zastosowane w robocie Khepera III czujniki zbliżeniowe wykorzystują promieniowanie podczerwone, w związku z czym wielkość sygnału z czujników istotnie zależy od koloru przeszkody – przeszkody jasne są wykrywane z większej odległości niż przeszkody ciemne. Zależność wartości sygnału czujnika s_i w funkcji odległości od przeszkody w kolorze białym przedstawiono na Rys. 3.



Rys. 3. Charakterystyka czujników zbliżeniowych nr 1, 3, 4, 6-8, 10-11 robota mobilnego Khepera III. Charakterystyka zbliżeniowa czujników 2, 5, 8 oraz 9 posiada podobny przebieg z tym, że wartość minimalna wynosiła odpowiednio 2000, 1100, 100 oraz 1000. Na osi odciętych przedstawiono odległość od przeszkody. Pomiarów dokonano dla przeszkody w kolorze białym.

Analogowy sygnał czujników jest przetwarzany na 12-bitową reprezentację cyfrową, stąd też teoretyczny zakres sygnału $s_i \in [0{,}4095]$. Zmierzony zakres wartości sygnału czujników 1, 3, 4, 6-8, 10-11 zawiera się w przedziale $[146{,}3894]$. Maksymalna odległość, przy której czujnik reagował na przeszkodę, to około 25 cm. W odległości równej 1 cm następowało nasycenie czujnika, tzn. dalsze zmniejszanie odległości od przeszkody nie powodowało zwiększenia odpowiadającego mu sygnału.

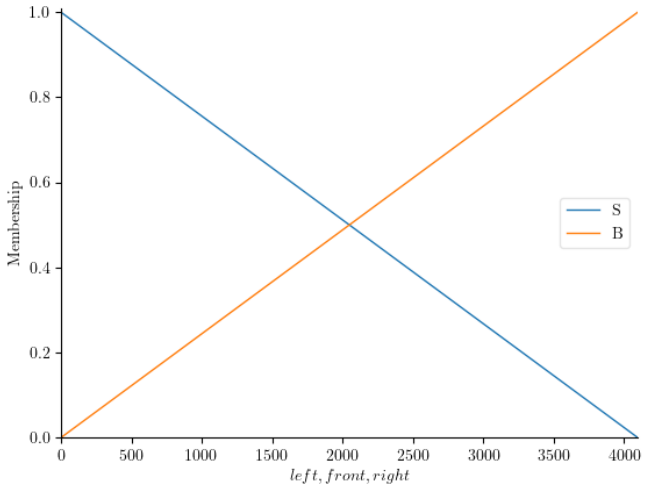


Rys. 4. Propozycja zbiorów rozmytych dla czujników zbliżeniowych $s_1, .., s_9$ robota Khepera III.

Jeżeli uniwersum sygnału każdego z czujników pokryje się dwoma zbiorami rozmytymi (Rys. 4), to w efekcie otrzyma się aż 2^9=512 reguł rozmytego systemu ekspertowego. Zaproponowanie przez eksperta następników tak dużej liczby reguł może być uciążliwe, a w trakcie laboratorium wręcz niemożliwe, stąd konieczność poszukiwania pewnych uproszczeń. **Uproszczenie pierwsze** będzie polegało na odrzuceniu informacji płynącej z tylnych czujników zbliżeniowych $s_7, .., s_9$. W rezultacie otrzymamy 2^6=64 reguł, co wciąż jest dużą liczbą do określenia w trakcie laboratorium. W związku z brakiem małych przeszkód w środowisku testowym nie jest wymagane rozróżnianie

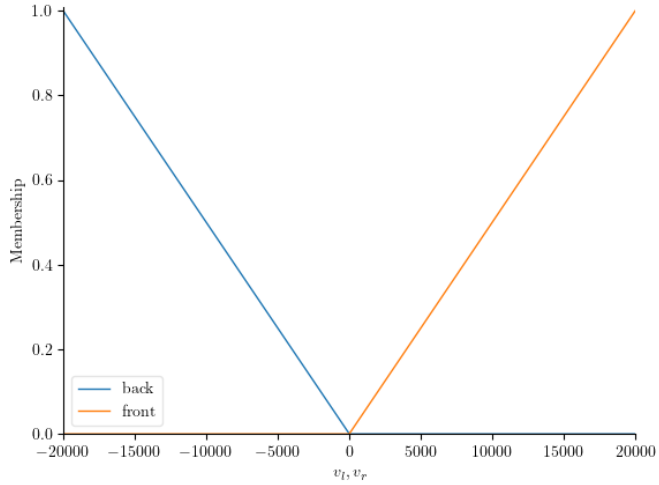
obecności przeszkód przy pojedynczym czujniku. W związku z powyższym **uproszczenie drugie** będzie obejmowało połączenie sygnałów z sąsiednich czujników. W efekcie powstaną metazmienne reprezentujące *trzy* strony robota:

$$left = \max(s_1, s_2)$$
$$front = \max(s_3, s_4)$$
$$right = \max(s_5, s_6)$$



Rys. 5. Propozycja zbiorów rozmytych dla zmiennych *left*, *front* i *right*.

Dziedziny zmiennych *left*, *front* i *right* pokryto również 2 zbiorami rozmytymi (Rys. 5) co prowadzi do liczby reguł wynoszącej 2^3=8. Zaproponowano również kształty trójkątne zbiorów rozmytych v_l oraz v_r reprezentujących prędkości kół (Rys. 6).



Rys. 6. Propozycja zbiorów rozmytych dla następników reguł – prędkości v_l oraz v_r.

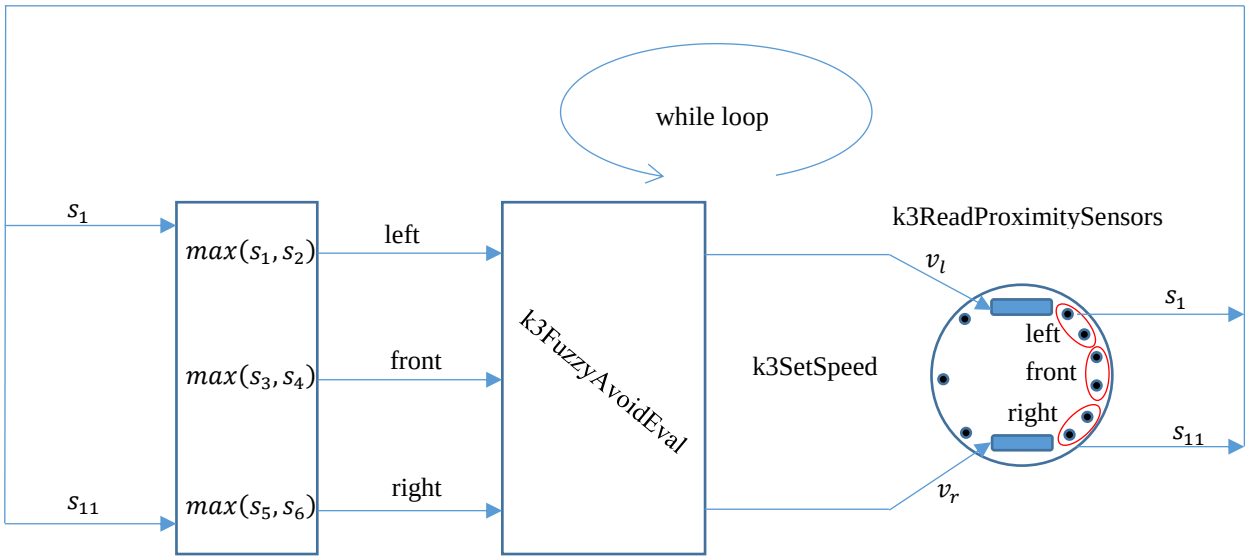
Tabela 2. Propozycja reguł sterowania robotem mobilnym Khepera III realizującym zadanie omijania przeszkód.

Lp.	*left*	*front*	*right*	v_l	v_r
1	S	S	S	front	front
2	S	S	B	back	front
3	S	B	S	back	front
4	S	B	B	back	front
5	B	S	S	front	back
6	B	B	S	front	back
7	B	S	B	front	front
8	B	B	B	back	front

Sterowanie będzie zatem odbywało się cyklicznie (w pętli) i będzie składało się z następujących kroków: odczyt czujników zbliżeniowych (funkcja `k3ReadProximitySensors`), wyznaczenie zmiennych *left*, *front*, *right*, obliczenie wyjścia systemu rozmytego o zbiorach z rys. 5 i 6 oraz regułach z Tablicy 2 (funkcja `k3FuzzyAvoidEval`). Wyjścia z funkcji `k3FuzzyAvoidEval` będą następnie aplikowane do robota Khepera za pomocą funkcji `k3SetSpeed`. Obieg sygnału w układzie sterowania przedstawiono na rys. 7. Realizację wszystkich elementów układu sterowania zawiera funkcja `k3FuzzyAvoidLoop` (Listing 1).

```
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import khepera3 as khep


def k3FuzzyAvoidEval(avoid_sym, val_left, val_front, val_right):
    # Compute the fuzzy system
    avoid_sym.input['left']    = val_left
    avoid_sym.input['front']   = val_front
    avoid_sym.input['right']   = val_right
    avoid_sym.compute()
    # print('vl=',avoid sym.output['vl'], ' vr=',avoid sym.output['vr'])
    return avoid_sym

def k3FuzzyAvoidLoop(s):
    avoid_sym = khep.k3FuzzyAvoidDef() # do przygotowania w II części laboratorium
    res = False
    iter = 0
    while iter <= 1000:
        sens = khep.k3ReadProximitySensors(s)
        # print(sens)
        val_left  = max(sens[1],sens[2])
        val_front = max(sens[3],sens[4])
        val_right = max(sens[5],sens[6])
        avoid_sym = k3FuzzyAvoidEval(avoid_sym, val_left, val_front, val_right)
        khep.k3SetSpeed(s,avoid_sym.output['vl'],avoid_sym.output['vr'])
        iter += 1
```

Listing 1. Funkcje `k3FuzzyAvoidEval` oraz `k3FuzzyAvoidLoop`



Rys. 7. Układ sterowania robotem mobilnym Khepera III realizującym zadanie omijania przeszkód. Graficzna ilustracja zawartości funkcji `k3FuzzyAvoidLoop`.

Przebieg części II laboratorium: zaprojektować system ekspertowego typu Mamdaniego z wykorzystaniem biblioteki scikit-learn realizujący zadanie omijania przeszkód

Zadaniem Państwa w tej części laboratorium jest zbudowanie systemu Mamdaniego sterującego robotem mobilnym Khepera III, którego zadaniem jest omijanie przeszkód. System powinien początkowo posiadać zbiory rozmyte dla wejść i wyjść takie, jak na Rys. 5 i 6 oraz reguły z tablicy 2. Zasadę projektowania systemu Mamdaniego poznali Państwo na poprzednich zajęciach laboratoryjnych. Poniżej przedstawiono zarys funkcji `k3FuzzyAvoidDef` (Listing 2), którą Państwo powinniście wypełnić właściwą treścią.

```
def k3FuzzyAvoidDef():
    MaxProximitiSignal = 4096
    MaxSpeed     = 20000
    left         = ctrl.Antecedent(np.arange(0,MaxProximitiSignal,1), 'left')
    front        =
    right        =
    vl           =
    vr           =

    left['S']    = fuzz.trimf(left.universe,  [0,                    0, MaxProximitiSignal])
    left['B']    =

    front['S']   =
    front['B']   =

    right['S']   =
    right['B']   =

    vl['back']   =
    vl['front']  =

    vr['back']   =
    vr['front']  =

    # left.view()
    # vl.view()

    # rules definition
    # Rule in a fuzzy control system, connecting antecedent(s) to consequent(s)
    rule1 = ctrl.Rule(antecedent=(left['S'] & front['S'] & right['S']),
                      consequent=(vl['front'], vr['front']) )
    rule2 =
    rule3 =
    rule4 =
    rule5 =
    rule6 =
    rule7 =
    rule8 =
```

```
    # The system is initialized and populated with a set of fuzzy Rules
    avoid_ctr = ctrl.ControlSystem([rule1,rule2,rule3,rule4,rule5,rule6,rule7,rule8])
    # Calculate results from a ControlSystem
    avoid_sym = ctrl.ControlSystemSimulation(avoid_ctr)
    return avoid_sym
```

Listing 2. Szkielet funkcji `k3FuzzyAvoidDef`

Przebieg części III laboratorium: praktyczna weryfikacja poprawności zrealizowanego systemu w sterowaniu rzeczywistym robotem

1. Wykonany w poprzednim punkcie system do omijania przeszkód (funkcję `k3FuzzyAvoidDef`) przesłać do komputera z podłączonym robotem.
2. Uruchomić skrypt Lab2_k3_avoid.py, którego elementem składowym jest funkcja `k3FuzzyAvoidDef`.
3. Uruchomić funkcję `k3FuzzyAvoidLoop` i udokumentować (opisać, utrwalić) zachowanie robota w otoczeniu przeszkód.
4. Zmodyfikować funkcję `k3FuzzyAvoidDef` tak, aby robot podążał za przeszkodą ruchomą.
5. Zrealizować zadanie podążania do źródła światła.
6. Do punktów 3-5 użyć innego systemu rozmytego, niż Mamdani, np.: Takagi-Sugeno zerowego, bądź pierwszego rzędu.

Sprawozdanie

W sprawozdaniu na ocenę **dobrą** należy udokumentować realizację punktów 1 - 3 z III części przebiegu laboratorium. W sprawozdaniu na ocenę **bardzo dobrą** należy dodatkowo wykonać punkty 4-6. W przypadku braku możliwości weryfikacji systemów sterujących za pomocą fizycznego robota można posłużyć się symulatorem.

Literatura

[1] http://ftp.k-team.com/KheperaIII/UserManual/Kh3.Robot.UserManual.pdf

Dodatki

Dodatek I: Zawartość pliku khepera3.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 17 11:32:56 2022

@author: RZ
"""

# pip install pyserial
import time
import numpy as np


def k3Init():
    import serial

    s = serial.Serial("COM3", 115200)
   # time.sleep(2)

    print(s.name)
    return s

def k3GetBateryState(s):
    s.write(b'V,5\r')
    serialString = s.readline()
    _ = s.read(1) #necessary to empty serial port
    serialString = serialString.decode("Ascii")
    serialString = serialString.strip() #remove the newline character from a string
    _ , bateryState, _ = serialString.split(",") #splits a string into a list
    print(f'Batery state: {bateryState}%\n')
    return bateryState

def k3ReadProximitySensors(s):
    MaxProximitiSignal = 4096
    res = False
    while not res:
        s.write(b'N\r')
        serialString = s.readline()
        _ = s.read(1) #necessary to empty serial port
        serialString = serialString.decode("Ascii")
        serialString = serialString.strip() #remove the newline character from a string
        proxSens = serialString.split(",") #splits a string into a list
        ProximitySensors = proxSens[1:-1] #remove first and last element
        ProximitySensors = np.array(list(map(int, ProximitySensors))) #convert string list to int list
        if len(ProximitySensors)==11:
            res = True
            # # some sensors need correction
            # # 1 - min val = 340, 4 - min val = 110, 7 - min val = 220 and 8 - min val = 120
            ProximitySensors[1] = (ProximitySensors[1]-2000)*(MaxProximitiSignal-0)/(MaxProximitiSignal-
2000)+0
            ProximitySensors[4] = (ProximitySensors[4]-1100)*(MaxProximitiSignal-0)/(MaxProximitiSignal-
1100)+0
            ProximitySensors[7] = (ProximitySensors[7]-100)*(MaxProximitiSignal-0)/(MaxProximitiSignal-100)+0
            ProximitySensors[8] = (ProximitySensors[8]-1000)*(MaxProximitiSignal-0)/(MaxProximitiSignal-
1000)+0
    return ProximitySensors

def k3ReadAmbientSensors(s):
    res = False
    while not res:
        s.write(b'O\r')
        serialString = s.readline()
        _ = s.read(1) #necessary to empty serial port
        serialString = serialString.decode("Ascii")
        serialString = serialString.strip() #remove the newline character from a string
```

```
        ambSens = serialString.split(",") #splits a string into a list
        AmbientSensors = ambSens[1:-1] #remove first and last element
        AmbientSensors = np.array(list(map(int, AmbientSensors))) #convert string list to int list
        if len(AmbientSensors)==11:
            res = True
    return AmbientSensors

def k3ReadSpeed(s):
    s.write(b'E\r')
    serialString = s.readline()
    _ = s.read(1) #necessary to empty serial port
    serialString = serialString.decode("Ascii")
    serialString = serialString.strip() #remove the newline character from a string
    Speed = serialString.split(",") #splits a string into a list
    Speed = Speed[1:] #remove first element
    Speed = list(map(int, Speed)) #convert string list to int list
    # print(f'Proximity sensors: {AmbientSensors}\n')
    return Speed

def k3ReadSoftwareVersion(s):
    s.write(b'B\r')
    serialString = s.readline()
    _ = s.read(1) #necessary to empty serial port
    serialString = serialString.decode("Ascii")
    serialString = serialString.strip() #remove the newline character from a string
    SV = serialString.split(",") #splits a string into a list
    SV = SV[1:-1] #remove first and last element
    SV = list(map(int, SV)) #convert string list to int list
    print(f'Software version stored in the robot\'s EEPROM is: {SV}\n')
    return SV

def k3SetSpeed(s,speed_motor_left,speed_motor_right):
    command=f'D,l{speed_motor_left},l{speed_motor_right}\r'
    s.write(str.encode(command))
    serialString = s.readline()
    _ = s.read(1) #necessary to empty serial port
    serialString = serialString.decode("Ascii")
    serialString = serialString.strip() #remove the newline character from a string
    # return serialString

def k3Stop(s):
    k3SetSpeed(s,0,0)

def k3ConfigureSpeedProfileController(s,max_speed,acceleration):
    # At the reset, these parameters aree set to standard values:
    # max_speed to MaxSpeed, acc to 64
    command=f'J,d{max_speed},{acceleration}\r'
    s.write(str.encode(command))
    _ = s.readline()
    _ = s.read(1) #necessary to empty serial port

def k3SetPosition(s,position_motor_left,position_motor_right):
    # Set the 32 bit position counter of the two motors. The unit is the pulse
    # each one corresponds to 0,047mm
    command=f'I,l{position_motor_left},l{position_motor_right}\r'
    s.write(str.encode(command))
    serialString = s.readline()
    _ = s.read(1) #necessary to empty serial port
    serialString = serialString.decode("Ascii")
    serialString = serialString.strip() #remove the newline character from a string
    # return serialString

def k3SetTargetPosition(s,target_position_motor_left,target_position_motor_right):
    positioning_accuracy = 0.01
    command=f'P,l{target_position_motor_left},l{target_position_motor_right}\r'
    s.write(str.encode(command))
    Position = k3ReadPosition(s)
    if target_position_motor_left==0:
        tpl = 1 # to avoid divide by 0
    else:
        tpl = target_position_motor_left
    if target_position_motor_right==0:
        tpr = 1
    else:
        tpr = target_position_motor_right
    while ( (abs((Position[0]-tpl)/tpl)) > positioning_accuracy) \
        & ( (abs((Position[1]-tpr)/tpr)) > positioning_accuracy):
        Position = k3ReadPosition(s) # read position until target one is reached
        time.sleep(0.02)
    k3Stop(s)
```

```
    return Position

def k3ReadPosition(s):
    res = False
    while not res:
        s.write(b'R\r')
        serialString = s.readline()
        _ = s.read(1) #necessary to empty serial port
        serialString = serialString.decode("Ascii")
        serialString = serialString.strip() #remove the newline character from a string
        Position = serialString.split(",") #splits a string into a list
        Position = Position[1:] #remove first element
        Position = list(map(int, Position)) #convert string list to int list
        if len(Position)==2:
            res = True
    return Position

def k3Braitenberg(s,mode):
    #mode==0 - Infrared sensors
    #mode==1 - Ultrasonic sensors
    #mode==2 - Stop Braitenberg mode
    command=f'A,{mode}\r'
    s.write(str.encode(command))
    _ = s.readline()
    _ = s.read(1) #necessary to empty serial port

def k3ReadProximitySensorsLoop(s):
    res = False
    iter = 0
    while not res:
        sens = k3ReadProximitySensors(s)
        print(sens)
        iter += 1
        if iter==1000:
            res = True


def k3SetProgrammableLed(s,LED,State):
    command=f'K,{LED},{State}\r'
    s.write(str.encode(command))
    _ = s.readline()
    _ = s.read(1) #necessary to empty serial port
```