Министерство науки и высшего образования Российской Федерации
Федеральное Государственное Автономное Образовательное Учреждение
Высшего Образования
“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»
Факультет Систем Управления и Робототехники

ЛАБОРАТОРНАЯ РАБОТА №3
по дисциплине
«Системы очувствления роботов»
«CoppeliaSim»

Выполнил: студент группы R41336c Рымкевич П.Д.
Преподаватель: Бжихатлов И.А.

Санкт-Петербург, 2020

Цели работы: Научиться создавать симуляцию в программе CoppeliaSim, управлять объектами в симуляции из внешнего скрипта, а также применять методы обработки сенсорной информации.

Задание:

1) Создать симуляцию простого мобильного робота в CoppeliaSim.

2) Наладить взаимодействие симуляции и внешнего скрипта, а именно, получение сенсорной информации из симуляции в скрипт, также управление моделью мобильного робота из скрипта на основе обработанной сенсорной информации.

3) Для обработки полученной сенсорной информации использовать один из изученных в рамках курса методов.

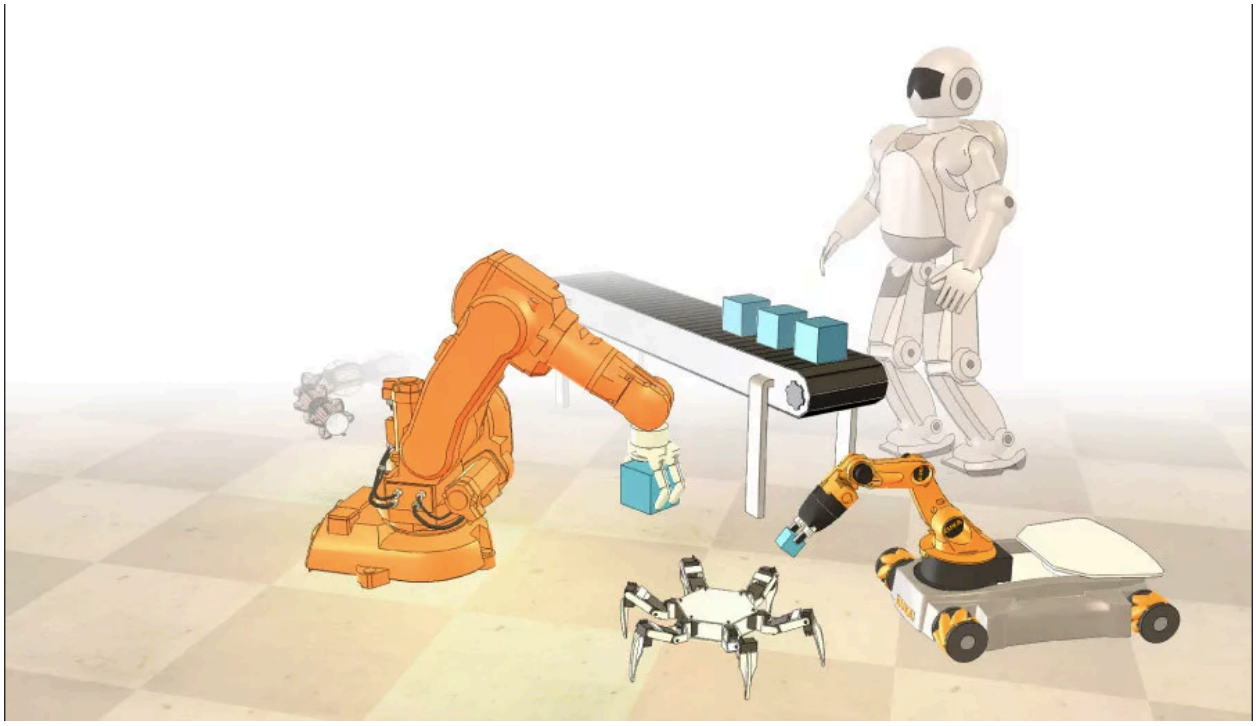

Рисунок 1 – Обложка пользовательского мануала CoppeliaSim.

В ходе выполнения лабораторной работы была создана сцена в симуляторе CoppeliaSim для реализации поставленных задач. В качестве мобильного робота был выбран робот Robotnik из стандартной библиотеки. Для получения сенсорной информации об окружении к роботу были добавлены 4 Proximity Sensor линейного типа, направленные назад, вперед, а также влево и вправо. Для управления движением робота использовались вращательные звенья у каждого колеса. Таким образом все колёса выбранного мобильного робота были ведущими.

В качестве окружения в сцене был создан периметр из стен, внутри которого подобные стены образовывали «рандомный лабиринт» при запуске скрипта. Целью для Robotnik было поставлено достижение ближайшего окружения модели человека (Работающего Билла), находя при этом выход из каждого «уровня» лабиринта посредством получения обработанной сенсорной информации.

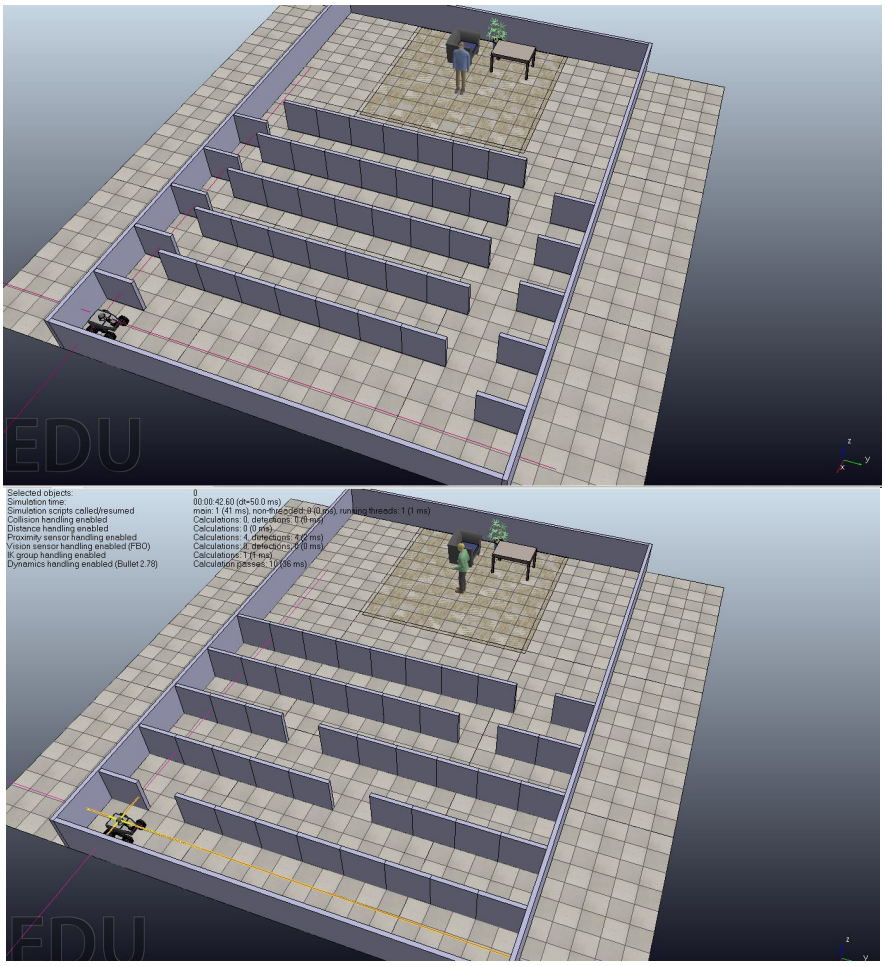

Рисунок 2 – Сцена до запуска скрипта (сверху) и после (снизу).

Для реализации скрипта на языке программирования Python 3.8 были подключены следующие библиотеки: sim; math; numpy; time; random; pykalman.

```
try:
    import sim
except:
    print ('--------------------------------------------------------------')
    print ('"sim.py" could not be imported. This means very probably that')
    print ('either "sim.py" or the remoteApi library could not be found.')
    print ('Make sure both are in the same folder as this file,')
    print ('or appropriately adjust the file "sim.py"')
    print ('--------------------------------------------------------------')
    print ('')

import sys
import math
import numpy as np
from time import sleep
from random import shuffle
from pykalman import KalmanFilter

pi = math.pi
sim.simxFinish(-1)
v=3 #standart vel
```

Рисунок 3 – Подключение библиотек и начальные данные.

Затем было произведено подключение скрипта к симуляции и получение данных об окружении следующим образом:

```
#Подключение к симуляции
clientID = sim.simxStart('127.0.0.1', 19999, True, True, 5000, 5)
if clientID!= -1: print("Connected to remote server")
else:
    print('Connection not successful')
    sys.exit('Could not connect')
#Моторы
errorCode, BL = sim.simxGetObjectHandle(clientID,'BL',sim.simx_opmode_blocking)
errorCode, BR = sim.simxGetObjectHandle(clientID,'BR',sim.simx_opmode_blocking)
errorCode, FL = sim.simxGetObjectHandle(clientID,'FL',sim.simx_opmode_blocking)
errorCode, FR = sim.simxGetObjectHandle(clientID,'FR',sim.simx_opmode_blocking)
#Сенсоры
errorCode, Fsen = sim.simxGetObjectHandle(clientID,'Fsen',sim.simx_opmode_blocking)
errorCode, Bsen = sim.simxGetObjectHandle(clientID,'Bsen',sim.simx_opmode_blocking)
errorCode, Lsen = sim.simxGetObjectHandle(clientID,'Lsen',sim.simx_opmode_blocking)
errorCode, Rsen = sim.simxGetObjectHandle(clientID,'Rsen',sim.simx_opmode_blocking)
#Объекты
errorCode, ROB = sim.simxGetObjectHandle(clientID,'Robotnik',sim.simx_opmode_blocking)
errorCode, base = sim.simxGetObjectHandle(clientID,'Base',sim.simx_opmode_blocking)
wall = np.zeros([5,7])
for i in range(5):
    for j in range(7):
        errorCode,wall[i][j] =  sim.simxGetObjectHandle(clientID,'w'+str(i)+str(j),sim.simx_opmode_blocking)
if errorCode == -1:
    print('Can not find object')
    sys.exit()
```

Рисунок 4 – Подключение к симуляции получение данных об объектах.

Для упрощения работы с кодом были созданы функции считывания сенсорной информации и управления движением мобильного робота. Так, например, сенсорная информация со всех датчиков выдавалась переменной

типа dict (словарь). Управление движением робота осуществлялось посредством управления скорость пар моторов (левых и правых). Таким образом повороты робота реализованы по принципу танка.

```python
#Данные с датчиков
def sensor():
    rightInputF = sim.simxReadProximitySensor(clientID, Fsen, sim.simx_opmode_oneshot_wait)
    rightInputB = sim.simxReadProximitySensor(clientID, Bsen, sim.simx_opmode_oneshot_wait)
    rightInputL = sim.simxReadProximitySensor(clientID, Lsen, sim.simx_opmode_oneshot_wait)
    rightInputR = sim.simxReadProximitySensor(clientID, Rsen, sim.simx_opmode_oneshot_wait)
    return {'F': rightInputF[2][2],'B': rightInputB[2][2],'R': rightInputR[2][2],'L': rightInputL[2][2]}

#движение моторов
def go(velocityL=v, velocityR='velocityL'):
    if (velocityR == 'velocityL'): velocityR = velocityL
    errorCode=sim.simxSetJointTargetVelocity(clientID,BL,velocityL, sim.simx_opmode_oneshot_wait)
    errorCode=sim.simxSetJointTargetVelocity(clientID,BR,-velocityR, sim.simx_opmode_oneshot_wait)
    errorCode=sim.simxSetJointTargetVelocity(clientID,FL,velocityL, sim.simx_opmode_oneshot_wait)
    errorCode=sim.simxSetJointTargetVelocity(clientID,FR,-velocityR, sim.simx_opmode_oneshot_wait)

stop = lambda: go(0)

#поворот, angle>0 - налево, angle<0 - направо
def rotate(angle=2*pi,velocity=v):
    per = 0;
    returnCode,eulerAngles=sim.simxGetObjectOrientation(clientID,ROB,base,sim.simx_opmode_oneshot_wait)

    A = angle+eulerAngles[2]
    per = abs(A)//pi
    A = A - 2*((per+1)//2)*np.sign(angle)*pi
    V = np.sign(angle)*velocity
    go(-V,V)
    data = eulerAngles[2]
    if np.sign(data+angle) != np.sign(data): per = per + 1
    while V!=0:
        returnCode,eulerAngles=sim.simxGetObjectOrientation(clientID,ROB,base,sim.simx_opmode_oneshot_wait)
        if np.sign(data)!= np.sign(eulerAngles[2]):
            per = per - 1
            data = eulerAngles[2]
        if per==0 and angle>0 and eulerAngles[2]>=A:
            stop()
            break
        elif per==0 and angle<0 and eulerAngles[2]<=A:
            stop()
            break
```

Рисунок 5 – Функции управления и получения сенсорной информации.

В качестве обработки сенсорной информации по новым значениям в реальном времени был выбран Фильтр Калмана, реализованный с помощью библиотеки pykalman. Моделью предсказания было выбрано расстояние в каждом измеряемом направлении, а моделью измерения – данные с датчиков. Так, предсказание происходило по формуле:

$$x_{k-1} = F_k x_k + G_k w_k, \quad (1)$$

где x_{k-1}, x_k – предсказания в каждом направлении $[\, x_F \; x_B \; x_R \; x_L \,]$;

F_k – матрица предсказания $[\, 1 \; 1 \; 1 \; 1 \,]$;

$G_k w_k = 0$ – шум.

Корректировка происходила за счет измерений сенсоров, таким образом y_k – показания каждого датчика.

```
#Начальные значения фильтра Калмана
def startKalman():
    kf = KalmanFilter(transition_matrices=[1], observation_matrices=[1])
    a = sensor()
    x = [a['L']]
    x1 = [a['R']]
    x2 = [a['F']]
    x3 = [a['B']]
    a = sensor()
    x.append(a['L'])
    x1.append(a['R'])
    x2.append(a['F'])
    x3.append(a['B'])
    m = {'L': np.mean(x),'R': np.mean(x1),'F': np.mean(x2),'B': np.mean(x3)}
    c = {'L': np.cov(x), 'R': np.mean(x1), 'F': np.mean(x2),'B': np.mean(x3)}
    return m,c,kf
#Фильтр калмана по новым значениям
def Kalman(m,c,kf):
    a = sensor()
    m['L'],c['L'] = kf.filter_update(m['L'],c['L'],a['L'])
    m['R'],c['R'] = kf.filter_update(m['R'],c['R'],a['R'])
    m['F'],c['F'] = kf.filter_update(m['F'],c['F'],a['F'])
    m['B'],c['B'] = kf.filter_update(m['B'],c['B'],a['B'])
    m = {'L': m['L'][0][0],'R': m['R'][0][0],'F': m['F'][0][0],'B': m['B'][0][0]}
    return m,c,kf
```

Рисунок 6 – Реализация фильтра Калмана.

Следующая функция – прохождение каждого уровня лабиринта. Движение вперед/назад мобильного робота основано на постоянной корректировке по условию равенства расстояния до стенок «уровня». В это же время происходит поиск проёма в уровне с помощью отфильтрованных показаний левого датчика. При получении данных о наличии проёма действиями робота будут следующие:

- поворот на 90 градусов против часовой стрелки;
- проезд через проём до наличия стенки впереди;
- поворот в направлении движения.

Затем цикл повторяется для каждого «уровня».

```
def finder(v=2,door='L'):
    endFlag = 0
    while 1:
        if v>0: direct='F'
        else: direct='B'
        if door=='L': wall='R'
        elif door=='R': wall='L'
        m,c,kf = startKalman()
        while m[door]>0.4 and m[door]<1.0:
            if m[wall]<0.4 or m[wall]>1.0: m[wall]=m[door]
            k = m[door]/m[wall]
            k = math.sqrt(k)
            go(v,k*v)
            m,c,kf = Kalman(m,c,kf)
            if m[direct]<1.0:
                v=-v
                if v>0: direct='F'
                else: direct='B'
        dir0=m[direct]
        go(v)
        while dir0-m[direct]<0.35:
            m,c,kf = Kalman(m,c,kf)
        go(-1,1)
        returnCode,data=sim.simxGetObjectOrientation(clientID,ROB,base,sim.simx_opmode_oneshot_wait)
        while 1:
            returnCode,eulerAngles=sim.simxGetObjectOrientation(clientID,ROB,base,sim.simx_opmode_oneshot_wait)
            if np.sign(data[2])!= np.sign(eulerAngles[2]): break
        go(3)
        m,c,kf = Kalman(m,c,kf)
        returnCode,data=sim.simxGetObjectPosition(clientID,ROB,base,sim.simx_opmode_oneshot_wait)
        while m['F']>1.0:
            m,c,kf = Kalman(m,c,kf)
            returnCode,data=sim.simxGetObjectPosition(clientID,ROB,base,sim.simx_opmode_oneshot_wait)
            if data[0]<-3.0:
                endFlag = 1
                break
        if endFlag: break
        rotate(-pi/2,1)
```

Рисунок 7 – Функция нахождения проёма.

В основной части программы первым делом идёт расстановка компонентов сцены при помощи функции random.sguffle(). Каждая стена образует те самые «уровни» лабиринта, а оставшееся координата без стены образует проём.

Затем реализуются описанная выше функция поиска проёма на каждом уровне. После достижения финишной области сцены робот движется непосредственно к модели Работающего Билла.

```
#Создание сцены
placeY = np.zeros(8)
for i in range(len(placeY)):
    placeY[i] = -3.5+i
for i in range(5):
    placeX = 5.95 - 1.5*i
    shuffle(placeY)
    for j in range(7):
        pos = [placeX,placeY[j],0.5]
        returnCode = sim.simxSetObjectPosition(clientID,int(wall[i][j]),base,pos,sim.simx_opmode_oneshot_wait)

#Основная часть программы по поиску выхода
finder()
returnCode,data=sim.simxGetObjectPosition(clientID,ROB,base,sim.simx_opmode_oneshot_wait)
s=0
if data[1]<0: s=-1
elif data[1]>0: s=1
rotate(s*pi/2)
go(3)
nu=data[1]
while np.sign(data[1])==np.sign(nu):
    returnCode,data=sim.simxGetObjectPosition(clientID,ROB,base,sim.simx_opmode_oneshot_wait)
rotate(-s*pi/2)
sim.simxAddStatusbarMessage(clientID,'Hello, Bill!',sim.simx_opmode_oneshot)
```

Рисунок 8 – Основная часть программы.

Вывод

В ходе выполнения лабораторной работы был реализован скрипт, при подключении к симуляции которого мобильный робот выполняет поиск выхода из «многоуровневого лабиринта». При тестировании работоспособности симуляция показала успешность достижения цели сравнимую с 85%. Основной проблемой выполнения скрипта является задача корректировки движения между стенами «уровня». Скорее всего, это связано с наличием больших задержек при выполнении данной части кода. Для минимизации ошибки было решено установить низкую скорость движения и низкий коэффициент корректировки движения для дифференцированного привода.

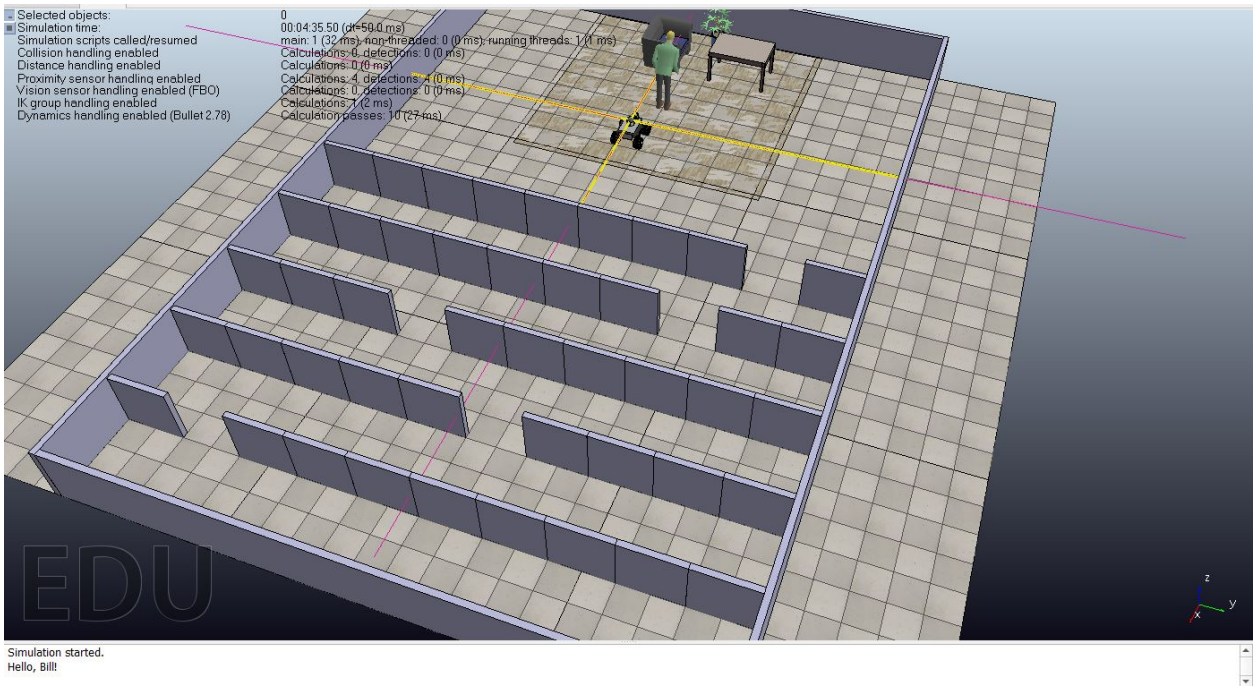

Рисунок 9 – Результат выполнения скрипта.