МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет информационных систем и технологий

Кафедра «Вычислительная техника»

Дисциплина «Программирование робототехнических комплексов»

**Лабораторная работа №1**

«ПИД-регулятор»

Вариант 12

Выполнил:

студент гр. ИВТВМбд-31

Кондратьев П.С.

Проверил:

Святов К.В.

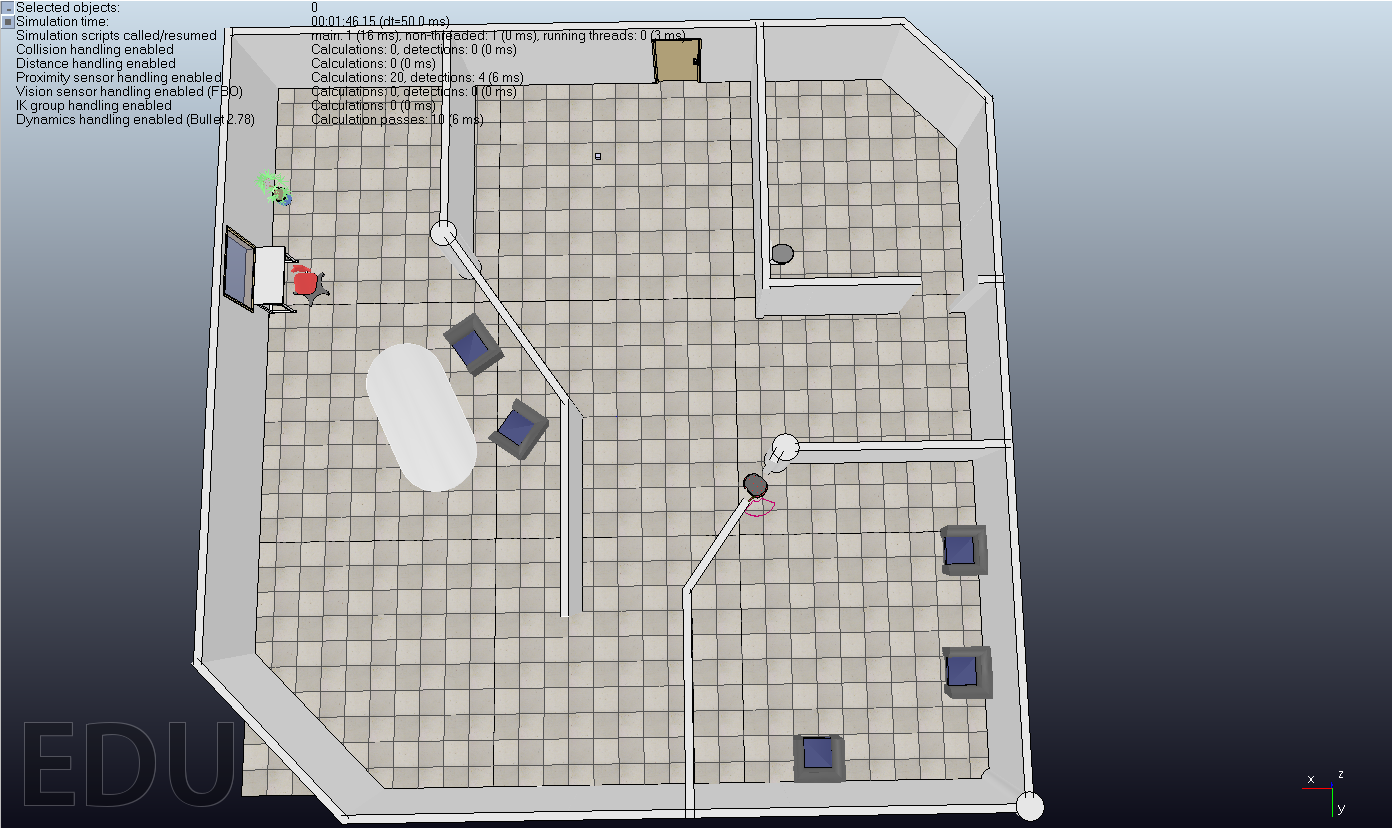
Ульяновск, 2019

**Техническое задание**

1. В среде V-Rep Pro Edu необходимо создать сцену (lab1.ttt файл), в которую поместить модель помещения со следующими параметрами:
   1. размер не менее 10x10м;
   2. количество комнат не менее 4;
   3. не менее 7 интерьерных объектов (столы, стулья, цветы и т.п.).
2. В сцену необходимо добавить мобильного робота (за основу можно взять модель Pioneer P3DX, либо другого робота), добавить к нему датчик приближения (ультразвуковой). Датчик необходимо сориентировать перпендикулярно направлению движения робота в соответствии с вариантом (справа).
3. Написать код на языке python, который позволит роботу двигаться вдоль одной (правой) стены на фиксированном расстоянии (0.6) от нее с использованием ПИД-регулятора. Движение должно происходить с объездом препятствий.
4. По итогу проезда робота необходимо вывести график изменения ошибки целевого расстояния до стены, а также управляющего воздействия (скорости), измеряемой во время тактов движения робота.

**Реализация**

В среде V-Rep Pro Edu была создана следующая сцена (15x15 см) и добавлен робот Pioneer P3DX:

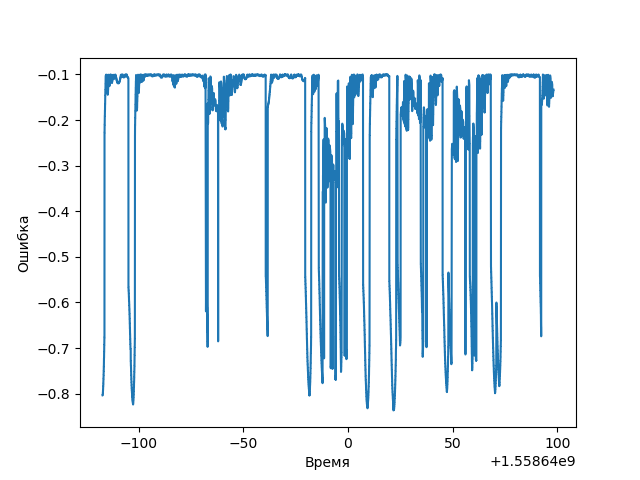


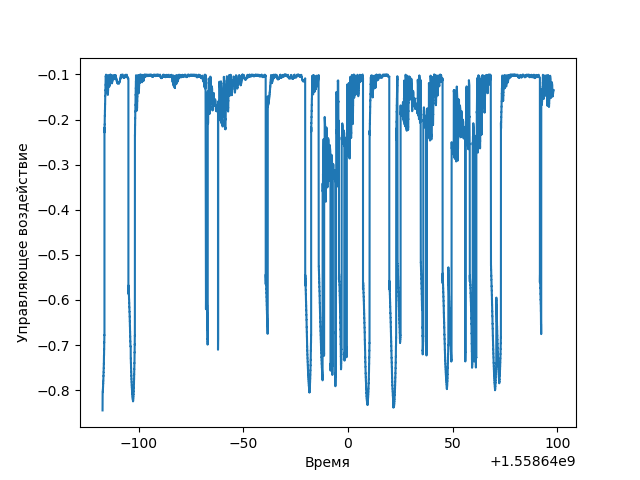
На робота было установлено два индуктивный датчика: датчик справа держит робота на дистанции 0.6 от стены, датчик спереди позволяет объезжать препятствия перед роботом. Скрипты своих собственных датчиков у робота были отключены и задана начальная скорость равная 0.

После создания сцены, добавления робота и датчиков к нему был написан код на языке python, осуществляющий необходимое движение робота и вывод графиков ошибки и управляющего воздействия при помощи библиотек V-Rep Remote Api и pylab.

По названию объектов в сцене получаются дескрипторы для моторов и датчиков. Затем начинается симуляция движения робота, осуществляющаяся с помощью дескрипторов датчиков. Датчики определяют дистанцию от стены справа или препятствия перед роботом, находят разницу этой дистанции с желаемой, т.е. находят ошибку, передают это значение в ПИД-регулятор, который использует его для вычисления управляющего воздействия. Полученное управляющее воздействие передается на колеса с помощью дескрипторов моторов, тем самым осуществляя необходимое движение робота. Сам ПИД-регулятор вычисляет управляющее воздействие с помощью трех компонент: пропорциональной, интегральной и дифференциальной.

Полученные графики ошибки и управляющего воздействия:





**Код программы**

**SimpleTest.py - vrep.simxStart**

# Make sure to have the server side running in V-REP:   
# in a child script of a V-REP scene, add following command  
# to be executed just once, at simulation start:  
#  
# simRemoteApi.start(19999)  
#  
# then start simulation, and run this program.  
#  
# IMPORTANT: for each successful call to simxStart, there  
# should be a corresponding call to simxFinish at the end!  
  
try:  
 import vrep  
except:  
 print ('--------------------------------------------------------------')  
 print ('"vrep.py" could not be imported. This means very probably that')  
 print ('either "vrep.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 "vrep.py"')  
 print ('--------------------------------------------------------------')  
 print ('')  
  
import time  
  
print ('Program started')  
vrep.simxFinish(-1) # just in case, close all opened connections  
clientID=vrep.simxStart('127.0.0.1',19999,True,True,5000,5) # Connect to V-REP  
if clientID!=-1:  
 print ('Connected to remote API server')  
  
 # Now try to retrieve data in a blocking fashion (i.e. a service call):  
 res,objs=vrep.simxGetObjects(clientID,vrep.sim\_handle\_all,vrep.simx\_opmode\_blocking)  
 if res==vrep.simx\_return\_ok:  
 print ('Number of objects in the scene: ',len(objs))  
 else:  
 print ('Remote API function call returned with error code: ',res)  
  
 time.sleep(2)  
  
 # Now retrieve streaming data (i.e. in a non-blocking fashion):  
 startTime=time.time()  
 vrep.simxGetIntegerParameter(clientID,vrep.sim\_intparam\_mouse\_x,vrep.simx\_opmode\_streaming) # Initialize streaming  
 # while time.time()-startTime < 5:  
 # returnCode,data=vrep.simxGetIntegerParameter(clientID,vrep.sim\_intparam\_mouse\_x,vrep.simx\_opmode\_buffer) # Try to retrieve the streamed data  
 # if returnCode==vrep.simx\_return\_ok: # After initialization of streaming, it will take a few ms before the first value arrives, so check the return code  
 # print ('Mouse position x: ',data) # Mouse position x is actualized when the cursor is over V-REP's window  
 # time.sleep(0.005)  
  
 # Now send some data to V-REP in a non-blocking fashion:  
 vrep.simxAddStatusbarMessage(clientID,'Hello V-REP!',vrep.simx\_opmode\_oneshot)  
  
 # Before closing the connection to V-REP, make sure that the last command sent out had time to arrive. You can guarantee this with (for example):  
 vrep.simxGetPingTime(clientID)  
  
 # Now close the connection to V-REP:  
 vrep.simxFinish(clientID)  
else:  
 print ('Failed connecting to remote API server')  
print ('Program ended')

**MyRobotMove.py**

import vrep  
import sys  
import time  
import numpy as np  
import pylab as pl  
import math  
  
times = []  
errors = []  
us = []  
purpose = 0.6  
v = 1  
  
error\_old = 0  
i\_min = -0.2  
i\_max = 0.2  
i\_sum = 0  
  
def check\_error\_code(error\_code, message, need\_exit = False):  
 if error\_code != vrep.simx\_return\_ok:  
 print("ERROR: Code {}. {}".format(error\_code, message))  
 if need\_exit:  
 sys.exit()  
 return False  
 return True  
  
def set\_motor\_speed(client\_id, left\_speed, right\_speed):  
 e = vrep.simxSetJointTargetVelocity(client\_id, left\_motor\_handle, left\_speed, vrep.simx\_opmode\_oneshot\_wait)  
 check\_error\_code(e, "SetJointTargetVelocity for left motor got error code")  
 e = vrep.simxSetJointTargetVelocity(client\_id, right\_motor\_handle, right\_speed, vrep.simx\_opmode\_oneshot\_wait)  
 check\_error\_code(e, "SetJointTargetVelocity for right motor got error code")  
 print("Motor speed set to {} {}".format(left\_speed, right\_speed))  
  
def controller(error):  
 up = error  
 global error\_old, i\_min, i\_max, i\_sum  
 i\_sum += error  
 i\_sum = max(i\_sum, i\_min)  
 i\_sum = min(i\_sum, i\_max)  
 ui = 0.005 \* i\_sum  
 ud = 0.05 \* (error - error\_old)  
 error\_old = error  
 return up + ud + ui  
  
def start\_simulation(client\_id, proximity\_handle, proximity\_handle2):  
 #set\_motor\_speed(client\_id, 1, 1)  
  
 t = time.time()  
 while vrep.simxGetConnectionId(client\_id) != -1:  
 (errorCode, detectionState, detectedPoint, detectedObjectHandle,  
 detectedSurfaceNormalVector) = vrep.simxReadProximitySensor(client\_id, proximity\_handle,  
 vrep.simx\_opmode\_streaming)  
 right = detectedPoint[2] if detectionState == True else 1e6  
  
 (errorCode, detectionState2, detectedPoint2, detectedObjectHandle,  
 detectedSurfaceNormalVector) = vrep.simxReadProximitySensor(client\_id, proximity\_handle2,  
 vrep.simx\_opmode\_streaming)  
 front = detectedPoint2[2] - 0.3 if detectionState2 == True else 1e6  
 dist = min(right, front)  
 error = dist - purpose  
 u = controller(error)  
  
 if error > 0:  
 errorCode = vrep.simxSetJointTargetVelocity(client\_id, left\_motor\_handle, v + u,  
 vrep.simx\_opmode\_streaming)  
 errorCode = vrep.simxSetJointTargetVelocity(client\_id, right\_motor\_handle, v - u, vrep.simx\_opmode\_streaming)  
 print("r ", error, " ", v + u, " ", v - u, " ", u)  
 elif error < 0:  
 errorCode = vrep.simxSetJointTargetVelocity(client\_id, left\_motor\_handle, v + u,  
 vrep.simx\_opmode\_streaming)  
 errorCode = vrep.simxSetJointTargetVelocity(client\_id, right\_motor\_handle, v - u, vrep.simx\_opmode\_streaming)  
 print("l ", error, " ", v + u, " ", v - u, " ", u)  
  
 if abs(error) < 1 and abs(u) < 6:  
 times.append(time.time())  
 errors.append(error)  
 us.append(u)  
  
 print('Simulation finished')  
 pl.plot(times, errors)  
 pl.xlabel('Время')  
 pl.ylabel('Ошибка')  
 pl.savefig('error.png')  
 pl.close()  
 pl.plot(times, us)  
 pl.xlabel('Время')  
 pl.ylabel('Управляющее воздействие')  
 pl.savefig('u.png')  
 exit()  
  
print('Program started')  
vrep.simxFinish(-1)  
clientID = vrep.simxStart('127.0.0.1', 19997, True, True, 5000, 5)  
if clientID != -1:  
 print("Connected to remote server")  
else:  
 print('Connection not successful')  
 sys.exit('Could not connect')  
  
  
errorCode, left\_motor\_handle = vrep.simxGetObjectHandle(clientID, 'Pioneer\_p3dx\_leftMotor', vrep.simx\_opmode\_oneshot\_wait)  
check\_error\_code(errorCode, "Can not find left motor", True)  
errorCode, right\_motor\_handle = vrep.simxGetObjectHandle(clientID,'Pioneer\_p3dx\_rightMotor', vrep.simx\_opmode\_oneshot\_wait)  
check\_error\_code(errorCode, "Can not find right motor", True)  
  
errorCode, sensor1 = vrep.simxGetObjectHandle(clientID, 'Proximity\_sensor', vrep.simx\_opmode\_oneshot\_wait)  
check\_error\_code(errorCode, "Can not find proximity sensor", True)  
  
errorCode, sensor2 = vrep.simxGetObjectHandle(clientID, 'Proximity\_sensor2', vrep.simx\_opmode\_oneshot\_wait)  
check\_error\_code(errorCode, "Can not find proximity sensor2", True)  
  
errorCode, detectionState, detectedPoint, detectedObjectHandle, detectedSurfaceNormalVector = vrep.simxReadProximitySensor(clientID, sensor1, vrep.simx\_opmode\_streaming)  
  
errorCode, detectionState, detectedPoint, detectedObjectHandle, detectedSurfaceNormalVector = vrep.simxReadProximitySensor(clientID, sensor2, vrep.simx\_opmode\_streaming)  
  
  
sec, msec = vrep.simxGetPingTime(clientID)  
print("Ping time: %f" % (sec + msec / 1000.0))  
print("Initialization finished")  
  
  
start\_simulation(clientID, sensor1, sensor2)