# Занятие 3. Реализация нейросетевого управления с использованием Python/C#

Используя систему управления мобильным роботом, построенуую на основе метода потенциальных полей, можно сгенерировать обучающее множество для настройки нейронной сети (рис. 1). Данная нейронная сеть пу сути будет аппроксимировать функционал метода потенциальных полей.

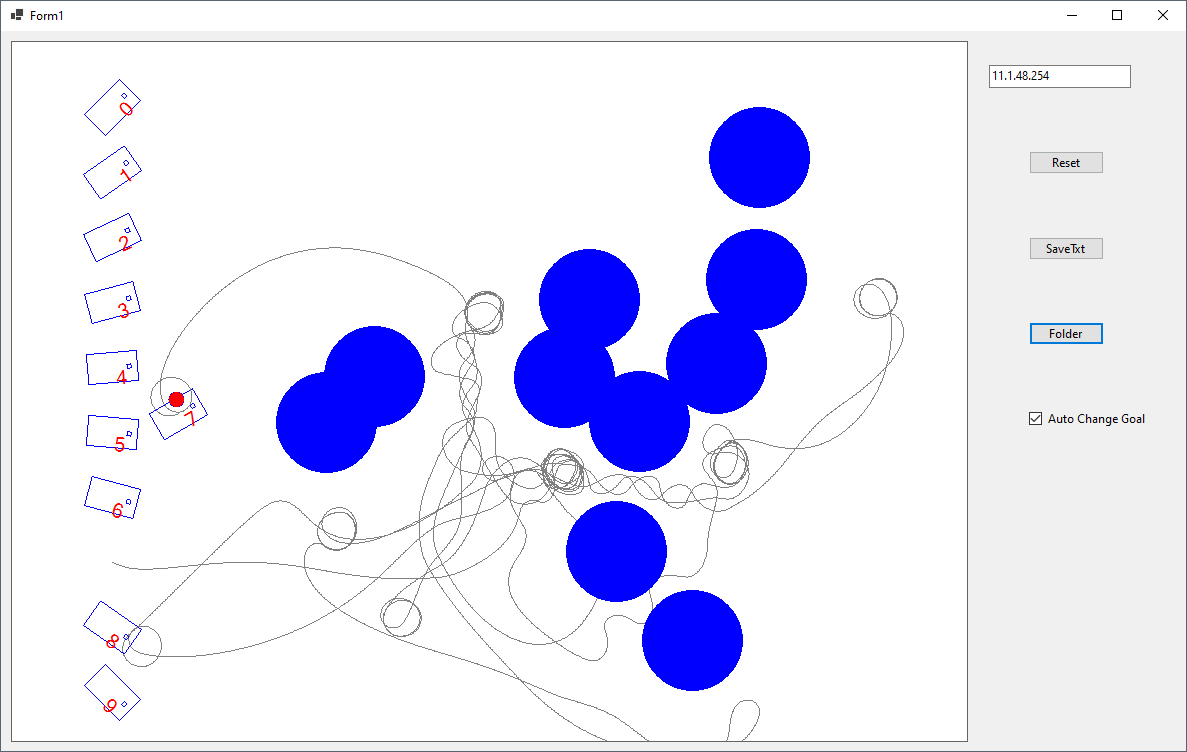


Рис. 1. Движение мобильного робота в процессе генерации обучающего множества. Траектория N-ого робота и данные о цели/препятствиях сохранаяются в файл lines(N).txt при нажатии на кнопку SaveTxt

На рис. 2 представлен датасет, полученный в процессе движения робота в среде с препятствиями

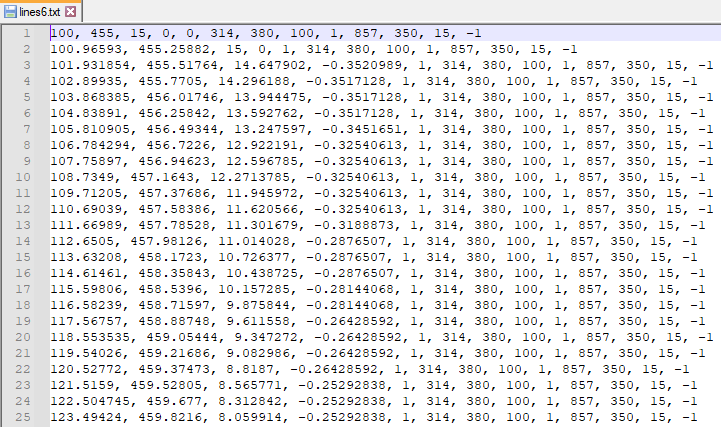


Рис. 2. Обучающее множество: входы – координаты робота x, y, angle, координаты ближайшего препятствия x2, y2, координаты цели x3, y3; выход – угловая скорость робота w

В качестве нейронной сети используется нейронная сеть с 1 скрытым слоем

#класс нейронной сети  
class Network(torch.nn.Module):  
 def \_\_init\_\_(self, hidden\_neurons):  
 super(Network, self).\_\_init\_\_()  
 # 1 слой принимающий сигнал  
 self.fc1=torch.nn.Linear(7, hidden\_neurons)  
 # 2 функция сигмоиды для второго слоя  
 self.act1=torch.nn.Tanh()  
 # 3 функция активации для выходного слоя  
 self.fc2=torch.nn.Linear(hidden\_neurons, 1)  
  
 def forward(self, x):  
 x = self.fc1(x)  
 x = self.act1(x)  
 x = self.fc2(x)  
 return x  
  
#экземпляр нейронной сети  
our\_network = Network(20)

Код обучения нейронной сети на датасет и сохранения ее на диск на языке Python выглядит следующим образом

import ast  
import torch  
import numpy as np  
  
#функция прореживания обучающей выборки  
def skip\_elements(elements, n):  
 new\_list = [ ]  
 i = 0  
 for element in elements:  
 if i%n==0:  
 c=elements[i]  
 new\_list.append(c)  
 i+=1  
 return np.array(new\_list)  
  
#функция перемешивания обучающих данных  
def shuffle\_along\_axis(a, axis):  
 idx = np.random.rand(\*a.shape).argsort(axis=axis)  
 return np.take\_along\_axis(a,idx,axis=axis)  
  
#загрузка текстового файла  
lines = open('lines6.txt','r')  
  
data = np.array([ast.literal\_eval(line.strip(',\n')) for line in lines])  
data = data.astype(np.float32)  
print(str(len(data)) + " lines")  
  
data = skip\_elements(data, 5)  
data = shuffle\_along\_axis(data, 0)  
print(str(len(data)) + " lines")  
  
cols\_inp=[0, 1, 2, 5, 6, 8, 9]  
cols\_out=[3]  
  
N=2000 #число обучающих примеров  
Nt=400 #число тестовых примеров  
  
#преобразование массива, получение входных и выходных данных  
x\_train=torch.from\_numpy(data[:N, cols\_inp])  
y\_train=torch.from\_numpy(data[:N, cols\_out]).unsqueeze(1)  
x\_test=torch.from\_numpy(data[N:N+Nt, cols\_inp])  
y\_test=torch.from\_numpy(data[N:N+Nt, cols\_out]).unsqueeze(1)  
  
#класс нейронной сети  
class Network(torch.nn.Module):  
 def \_\_init\_\_(self, hidden\_neurons):  
 super(Network, self).\_\_init\_\_()  
 # 1 слой принимающий сигнал  
 self.fc1=torch.nn.Linear(7, hidden\_neurons)  
 # 2 функция сигмоиды для второго слоя  
 self.act1=torch.nn.Tanh()  
 # 3 функция активации для выходного слоя  
 self.fc2=torch.nn.Linear(hidden\_neurons, 1)  
  
 def forward(self, x):  
 x = self.fc1(x)  
 x = self.act1(x)  
 x = self.fc2(x)  
 return x  
  
#экземпляр нейронной сети  
our\_network = Network(20)  
  
#функция потерь  
def loss(pred, target):  
 ans=(pred-target)\*\*2  
 return ans.mean()  
  
#способ оптимизации  
optimizer = torch.optim.Adam(our\_network.parameters(), 0.01)  
  
#обучение  
for i in range(200):  
 optimizer.zero\_grad()  
 y\_pred = our\_network.forward(x\_train)  
 loss\_val = loss(y\_pred, y\_train)  
 loss\_val.backward()  
 optimizer.step()  
 if(i%5==0): #в начале каждой эпохи  
 y\_pred = our\_network.forward(x\_test)  
 print((((y\_pred-y\_test)\*\*2).mean())\*\*0.5)  
 print("Loss = %f"%loss\_val)  
 #predict(our\_network, x\_test, y\_test)  
 ##plt.savefig("mygraph-%d.png"%i)  
  
#сохранение сети  
torch.save(our\_network, "our\_network.torch")  
  
#загрузка сети (в качестве примера)  
net = torch.load("our\_network.torch")  
net.eval()  
  
# должно быть близко к значению из выборки (в качестве примера)  
y=net.forward(torch.from\_numpy(np.array([104.531555, 522.11304, 25, 314, 380, 857, 350]).astype(np.float32)))  
print("Result = %f"%y)

Код системы управления роботом с возможностью переключения между методом потенциальных полей и обученной нейронной сетью выглядит следующим образом:

import math  
import requests  
import numpy as np  
from urllib.parse import urlencode  
import time  
  
  
  
#сессия HTTP - чтоб избежать многократных переподключений к серверу  
session = requests.Session()  
  
#отылка данных роботу  
def send(data):  
 str = urlencode(data)  
 r = session.get("http://192.168.0.121:8080/?" + str, timeout=1000)  
 return r.text.split("UserResult=")[1]  
  
#нормализация вектора  
def normalize(x):  
 norm = np.sqrt(x.dot(x))  
 x = x / norm if norm != 0 else x  
 return x  
  
#нормализация  
def normalize\_ang(ang):  
 while(ang<-math.pi):ang+=2\*math.pi  
 while(ang>math.pi):ang-=2\*math.pi  
 return ang  
  
T=0  
  
USE\_NET=True  
  
if USE\_NET:  
 import torch  
 # класс нейронной сети  
 class Network(torch.nn.Module):  
 def \_\_init\_\_(self, hidden\_neurons):  
 super(Network, self).\_\_init\_\_()  
 # 1 слой принимающий сигнал  
 self.fc1 = torch.nn.Linear(7, hidden\_neurons)  
 # 2 функция сигмоиды для второго слоя  
 self.act1 = torch.nn.Sigmoid()  
 # 3 функция активации для выходного слоя  
 self.fc2 = torch.nn.Linear(hidden\_neurons, 1)  
 def forward(self, x):  
 x = self.fc1(x)  
 x = self.act1(x)  
 x = self.fc2(x)  
 return x  
 # загрузка сети  
 net = torch.load("our\_network.torch")  
 net.eval()  
  
  
#управляющие данные  
data = {'id': 6, 'v': 1, 'w': 0}  
  
# f=open("dataset.txt", "w")  
  
#цикл системы управления  
while(True):  
 info=send(data)  
  
 # обработка ответа от сервера об объектах среды  
 arr = info.split('; ')  
 arr\_rob = np.fromstring(arr[0], dtype=float, sep=', ')  
 arr\_obst = np.fromstring(arr[1], dtype=float, sep=', ')  
 arr\_goal = np.fromstring(arr[2], dtype=float, sep=', ')  
  
 if USE\_NET:  
 # должно быть близко к значению из выборки  
 w\_ang = net.forward(torch.from\_numpy(np.array(  
 [arr\_rob[0], arr\_rob[1], arr\_rob[2],  
 arr\_obst[0], arr\_obst[1],  
 arr\_goal[0], arr\_goal[1]]).astype(np.float32))).item() \* 3  
 print("Result = %f" % w\_ang)  
 else:  
 #вектор соединяющий препятствие и робота, направленный в сторону робота  
 vro=np.asarray((arr\_rob[0]-arr\_obst[0], arr\_rob[1]-arr\_obst[1]))  
 Lro=np.sqrt(vro.dot(vro)) #расстояниe до препятствия  
 Fro = 100\*\*2/Lro\*\*2 #сила обратно пропорциональна квадрату расстояния  
 vro=normalize(vro)\*Fro  
  
 #вектор соединяющий цель и робота, направленный в сторону цели  
 vgr=np.asarray((arr\_goal[0]-arr\_rob[0], arr\_goal[1]-arr\_rob[1]))  
 Fgr=1 #сила - константа  
 vgr=normalize(vgr)\*Fgr  
  
 #целевое направление робота  
 v\_result = 0.5\*vgr+0.5\*vro  
 v\_result=normalize(v\_result)  
  
 #расчет управления для робота  
 beta = arr\_rob[2]/180\*math.pi #глобальный угол робота  
 gamma = math.atan2(v\_result[1], v\_result[0]) #целевой угол  
 alpha = normalize\_ang(gamma - beta)  
  
 w\_ang = alpha  
  
 data['w'] = w\_ang  
  
 print(data['w'])  
  
 T+=1  
 time.sleep(0.1)  
  
  
#запись обучающих примеров  
#file = open("dataset.txt", "w+")  
#arr = [arr\_rob[0], arr\_rob[1], arr\_goal[0], arr\_goal[1], arr\_obst[0], arr\_obst[1], data['w']]  
#file.write(str(arr));