CS437 Extra Credit

Prithvi Balaji, Joseph Stack

Extra Credit for Postlab 3:

How about a reinforcement learning algorithm that adjusts the reward based on RSSI readings or packet detection (you will receive 10 extra credits if you choose this approach)!

Our approach:

- We run through the program and walk around the room to simulate receiving RSSI values from different locations.
- Our RPI will receive rewards at points in the room (greater reward when the RSSI is stronger; smaller reward when RSSI is weaker)
- After the learning is finished → we use the sense hat to indicate the direction in where we should move to find the spy camera located in the room using the reward system (follow direction where there is greater reward)

The captured_packet_callback function is called whenever a packet is captured by the network interface. This function extracts MAC addresses, RSSI values, and accelerometer data (x and y) from the captured packet and sensor readings. The code initializes a packet sniffer (AsyncSniffer) to capture packets on the specified network interface (iface_n). The captured_packet_callback function is used as the callback to process each captured packet.

The code starts by initializing various parameters and variables, such as MAC addresses, network interface, the duration of the sniffing process, and Q-learning parameters like learning rate, discount factor, and exploration probability. This code comes from the PostLab3 prior before we are adding a reinforcement learning algorithm.

In our EC algorithm, we are having the state space is defined based on an 8x8 grid, where each cell represents a state which is the LED space. The action space consists of four possible actions: "move_up," "move_down," "move_left," and "move_right." The Q-table is initialized as a dictionary with states as keys and sub-dictionaries as values, where each sub-dictionary maps actions to their associated Q-values. Initially, all Q-values are set to 0.
The choose_action function is used to select an action for the agent in a state based on an epsilon-greedy strategy. With probability exploration_prob, a random action is chosen, and with probability 1 - exploration_prob, the action with the highest Q-value in the current state is selected. Rewards are assigned based on the RSSI values. The code customizes reward

values depending on the RSSI value for a specific MAC address. The higher the RSSI value, the higher the reward. After calculating the reward and selecting an action, the Q-value for the current state-action pair is updated using the Q-learning update rule. It incorporates the current Q-value, the learning rate, the reward, and the maximum Q-value of the next state.

Printing Results:
The code prints the start time of the sniffing process.
In summary, this code uses Q-learning to make decisions for an agent based on RSSI data and accelerometer readings. The agent learns to choose actions in different states by updating a Q-table, and it aims to maximize cumulative rewards by selecting actions that lead to higher RSSI values.

```
import csv
from datetime import datetime
from scapy.all import *
import time
from sense_hat import SenseHat
import os
#import pandas as pd
import matplotlib.pyplot as plt

red= (255,0,0)
orange = (255,165,0)
yellow = (255,255,0)
green = (0,255,0)

sense = SenseHat()

"""
Run monitor_mode.sh first to set up the network adapter to monitor
mode and to
set the interface to the right channel.
To get RSSI values, we need the MAC Address of the connection
of the device sending the packets.
"""


dev_mac = ""
iface_n = "wlan1"
duration = 180
```

```
sense=SenseHat()
path="/home/pi/Desktop/lab3/IMU/newdata/"
timestamp_fname=datetime.now().strftime("%H:%M:%S")
sense.set_imu_config(True,True,True)
filename=path+timestamp_fname+".csv"

max_rssi = (0,0,-1000)

def create_rssi_file():
    """Create and prepare a file for RSSI values"""
    #header = ["date", "time", "dest", "src", "rssi"]
    header = ['x','y', 'RSSI', 'Timestamp']
    with open(filename, "w", encoding="UTF8") as f:
        writer = csv.writer(f)
        writer.writerow(header)


def captured_packet_callback(pkt):
    global max_rssi
    """Save MAC addresses, time, and RSSI values to CSV file if MAC
address of src matches"""
    missed_count = 0
    cur_dict = {}

    try:

        cur_dict["mac_1"] = pkt.addr1
        cur_dict["mac_2"] = pkt.addr2
        cur_dict["rssi"] = pkt.dBm_AntSignal
    except AttributeError:
        return


    date_time =
datetime.now().strftime("%d/%m/%Y,%H:%M:%S.%f").split(",")
    date = date_time[0]
    time1 = date_time[1]

    accel=sense.get_accelerometer_raw()
    gyro=sense.get_gyroscope_raw()
    mag=sense.get_compass_raw()

    x=accel['x']
```

```
    y=accel['y']
    z=accel['z']
   # print("cur d-", cur_dict)
    #if cur_dict['mac_1'] == 'e4:5f:01:d4:9f:f9'
    new_tuple = (x, y, cur_dict['rssi'])

    color = ()
    if cur_dict['mac_1'] == "e4:5f:01:d4:9f:f9":# or
cur_dict['mac_1'] =="e4:5f:01:d4:9c:b1":
        rssi_val = abs(cur_dict['rssi'])
        if rssi_val >48:
            color = red
        elif rssi_val <=48 and rssi_val >=42:
            color = orange
        elif rssi_val <42 and rssi_val >36:
            color = yellow
        elif rssi_val <= 36:
            color = green

        print("NEW RSSI: ", rssi_val)

        for x in range(8):
            for y in range(8):
                sense.set_pixel(x,y,color)

        time.sleep(0.05)
        sense.clear()



    timestamp=datetime.now().strftime("%H:%M:%S")



if __name__ == "__main__":
    create_rssi_file()
    t = AsyncSniffer(iface=iface_n, prn=captured_packet_callback,
store=0)
    t.daemon = True
    t.start()

    start_date_time =
datetime.now().strftime("%d/%m/%Y,%H:%M:%S.%f")
```

```
    time.sleep(duration)
    t.stop()

    print("Start Time: ", start_date_time)
```

Algorithm

```
import numpy as np


learning_rate = 0.1
discount_factor = 0.9
exploration_prob = 0.2


num_states = 10
num_actions = 2


q_table = np.zeros((num_states, num_actions))

#example RSSI
rssi_values = [0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0, -0.1, -0.2]

packet_detected = [1, 0, 1, 1, 0, 1, 1, 0, 0, 1]

def choose_action(state):
    if np.random.uniform(0, 1) < exploration_prob:

        return np.random.choice(num_actions)
    else:

        return np.argmax(q_table[state, :])


def q_learning():
    for episode in range(100):
        state = 0
```

```
        for t in range(len(rssi_values)):
            action = choose_action(state)
            next_state = state + 1
            reward = rssi_values[next_state]


            if packet_detected[next_state] == 1:
                reward += 10


            q_table[state, action] = (1 - learning_rate) *
q_table[state, action] + learning_rate * (reward + discount_factor
* np.max(q_table[next_state, :]))

            state = next_state


q_learning()


print("Learned Q-table:")
print(q_table)
```

```
import csv
from datetime import datetime
from scapy.all import *
import time
from sense_hat import SenseHat
import os
import random

red = (255, 0, 0)
orange = (255, 165, 0)
yellow = (255, 255, 0)
green = (0, 255, 0)

sense = SenseHat()
```

```
dev_mac = ""
iface_n = "wlan1"
duration = 180
timestamp_fname = datetime.now().strftime("%H:%M:%S")
path = "/home/pi/Desktop/lab3/IMU/newdata/"
sense.set_imu_config(True, True, True)
filename = path + timestamp_fname + ".csv"

max_rssi = (0, 0, -1000)


learning_rate = 0.1
discount_factor = 0.9
exploration_prob = 0.1


state_space = [(x, y) for x in range(8) for y in range(8)]
action_space = ["move_up", "move_down", "move_left", "move_right"]


q_table = {}
for state in state_space:
    q_table[state] = {action: 0 for action in action_space}


def choose_action(state):
    if random.uniform(0, 1) < exploration_prob:
        return random.choice(action_space)
    else:
        return max(q_table[state], key=q_table[state].get)

def create_rssi_file():
    """Create and prepare a file for RSSI values"""
    header = ['x', 'y', 'RSSI', 'Timestamp']
    with open(filename, "w", encoding="UTF8") as f:
        writer = csv.writer(f)
        writer.writerow(header)

def update_q_table(state, action, reward, next_state):
    max_next_action_value = max(q_table[next_state].values())
    q_table[state][action] = (1 - learning_rate) *
q_table[state][action] + learning_rate * (reward + discount_factor
* max_next_action_value)

def captured_packet_callback(pkt):
    global max_rssi
    """Save MAC addresses, time, and RSSI values to CSV file if MAC
address of src matches"""
    missed_count = 0
```

```
    try:
        cur_dict = {}
        cur_dict["mac_1"] = pkt.addr1
        cur_dict["mac_2"] = pkt.addr2
        cur_dict["rssi"] = pkt.dBm_AntSignal
    except AttributeError:
        return  # Packet formatting error

    x = sense.get_accelerometer_raw()['x']
    y = sense.get_accelerometer_raw()['y']
    accel_state = (int(x * 8), int(y * 8))


    if cur_dict['mac_1'] == "e4:5f:01:d4:9f:f9":
        rssi_val = abs(cur_dict['rssi'])
        if rssi_val > 48:
            reward = 10
        elif rssi_val <= 48 and rssi_val >= 42:
            reward = 5
        elif rssi_val < 42 and rssi_val > 36:
            reward = 2
        else:
            reward = 1
    else:
        reward = 0

    action = choose_action(accel_state)
    next_x, next_y = accel_state
    if action == "move_up":
        next_y += 1
    elif action == "move_down":
        next_y -= 1
    elif action == "move_left":
        next_x -= 1
    elif action == "move_right":
        next_x += 1
    next_state = (next_x, next_y)

    update_q_table(accel_state, action, reward, next_state)

    timestamp = datetime.now().strftime("%H:%M:%S")

if __name__ == "__main":
    create_rssi_file()
    t = AsyncSniffer(iface=iface_n, prn=captured_packet_callback,
store=0)
    t.daemon = True
    t.start()

    start_date_time =
datetime.now().strftime("%d/%m/%Y,%H:%M:%S.%f")
```

```
    time.sleep(duration)
    t.stop()

    print("Start Time: ", start_date_time)
```