APPLIED ARTIFICIAL INTELLIGENCE

Robotics

PROJECT REPORT

Author:

Arthur VANDENHOEKE

Professor :

Denis STECKELMACHER

June 11, 2021

Contents

Chapter 1

Introduction

In this robotics class, we consider the task of making a person hold his/her Raspberry Pi in front of a laptop's webcam. The Raspberry Pi, equipped with its sense hat, allows to display a single red dot on its LED matrix so that the webcam can monitor it. The goal of this project is to ensure that the red dot always remains as close as possible to the webcam's captured image. This must particularly hold if the person shakes, moves, or rotates the Raspberry Pi in front of the webcam. If the Raspberry Pi's LED matrix is placed too far from the webcam's image center, the red dot should be placed as close as possible to the latter.

If we were to solve this problem manually, we would need to have access to the Raspberry Pi's accelerometer and gyroscope to determine the speed and the direction of the motion of the red dot on the LED matrix. This would require us to use a reference frame for which the origin would be the center of the webcam's image. Access to acceleration and gyroscopic data would allow to construct a velocity vector anchored at the red dot and pointing towards the origin of this reference frame. Doing so would enable us to undo the person's motions and therefore stabilize the red dot at the center of the image. Although this approach seems technically feasible, its implementation is not straightforward.

In this course, we intend to use Reinforcement Learning (RL) as a "last-resort" solution to overcome the complexity of the previous approach. Our goal is thus to produce a *policy* that allows to move the red dot on the Raspberry Pi's LED matrix so as to be as close as possible to the center of the image. In order to achieve this goal,

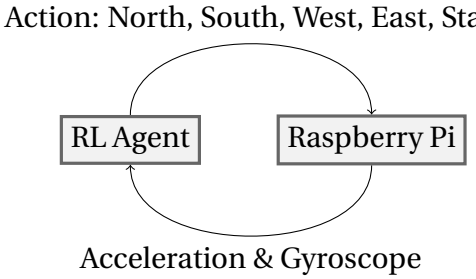

Figure 1.1: Communication between the Raspberry Pi and the RL agent. The Pi sends its acceleration and gyroscopic data over to the agent which decides what action to take in order to re-center the red dot on the Pi.

we will need the following components:

- A Raspberry Pi with a mounted sense hat (comprises the LED matrix);
- A laptop with a front webcam;
- A Reinforcement Learning agent.

In the following sections, we start by explaining how the Raspberry Pi can be set up to communicate with the laptop about the red dot's position on the webcam image. We also describe how the red dot's position can be retrieved from the raw image using image processing techniques. The last section of this chapter investigates the set up of the RL agent and its corresponding environment.

1.1 Configuration of the Raspberry Pi

In order for the RL agent to decide how to move the red dot on the Raspberry Pi 's LED matrix, the agent must have access to acceleration and gyroscopic data of the Pi. In return, the Pi must have access to the decision of the agent, given this data, in order to re-center the red dot. The actions of the agent are (1) *Move North*, (2) *Move South*, (3) *Move West*, (4) *Move East* and (5) *Stay*. Both entities thus communicate according to the scheme of figure 1.1.

To ensure a communication between both entities, we implement a Python socket server on the on both sides over the same wifi network. On the Pi, we create a python

socket server using code inspired from the project description. Section 2.1.1 of the Appendix illustrates this process. Apart from the functions for sending and receiving objects from/to the server, a Sensehat instance is created, along with a Raspberry instance. The Sensehat object is responsible for

1. displaying the LED light through its `set_pixel()` method;
2. sampling acceleration data through its `get_accelerometer_raw()` method;
3. sampling gyroscopic data through its `get_gyroscope_raw()` method.

The Raspberry instance on the other hand wraps the Sensehat instance into a class which has these acceleration and orientation measures as properties. The code of section 2.1.2 illustrates this process. In order to close the loop of figure 1.1, the Raspberry instance needs a function that translates the action prescribed by the agent into a visible result. This is exactly the purpose of the `move_led()` function of this class, which executes all five actions described above.

1.2 Webcam image processing

1.3 Reinforcement Learning agents

Chapter 2

Appendix

2.1 Code on the Raspberry Pi

2.1.1 server.py

```
import socket
from _thread import *
import pickle
import struct
from sense_hat import SenseHat
from rasp import Raspberry
import time

# Create server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('0.0.0.0', 9395))
s.listen(10)
print("Waiting for a connection")

# Configure Sensehat
sense = SenseHat()
sense.clear()
```

```
rasp = Raspberry(sense)

def send(s, data):
    data = pickle.dumps(data)
    s.sendall(struct.pack('>i', len(data)))
    s.sendall(data)

def recv(s):
    data = s.recv(4, socket.MSG_WAITALL)
    data_len = struct.unpack('>i', data)[0]
    data = s.recv(data_len, socket.MSG_WAITALL)
    return pickle.loads(data)

def threaded_client(conn):
    # Declare global variables
    global sense

    while True:
        try:
            data = recv(conn)
            # Place our red dot at desired location
            if isinstance(data, list):
                rasp.place_dot(data)

            # Execute action required by RL agent
            else:
                rasp.move_led(data)

        except:
            pass

        # Send Gyroscope and accelerator data
        reply = rasp.acceleration + rasp.orientation
```

```
        send(conn, reply)

while True:
    # Accept client
    conn, addr = s.accept()
    print('Connected to:', addr)
    start_new_thread(threaded_client, (conn, ))
```

2.1.2 rasp.py

```
import numpy as np


class Raspberry:

    def __init__(self, sense):
        self.sense = sense
        self.acceleration_ = None
        self.orientation_ = None
        self.led = [0, 0]

    @property
    def acceleration(self):
        acc = self.sense.get_accelerometer_raw()
        return [acc['x'], acc['y'], acc['z']]

    @property
    def orientation(self):
        gyro = self.sense.get_gyroscope_raw()
        return [gyro['x'], gyro['y'], gyro['z']]

    def place_dot(self, pos: np.ndarray) -> None:
        self.sense.clear()
```

```
        pos = pos[0]
        print("Resettting pixel ({}, {})".format(pos[0], pos[1]))
        self.sense.set_pixel(pos[0], pos[1], (255, 0, 0))

    def move_led(self, action: int) -> None:

        self.sense.clear()
        x, y = self.led[0], self.led[1]

        if action == 0:  # Move to the right
            self.led[0] = x-1 if x>0 else x

        if action == 1:  # Move the the left
            self.led[0] = x+1 if x<7 else x

        if action == 2:  # Move upwards
            self.led[1] = y-1 if y>0 else y

        if action == 3:  # Move down
            self.led[1] = y+1 if y<7 else y

        if action == 4:  # Stay at the same position
            self.led[0], self.led[1] = x, y

        self.sense.set_pixel(self.led[0], self.led[1], (255, 0, 0))
```