Họ và tên: Nguyễn Tiến Thọ

MSSV: 3122410398

Lab02\_01\_agents

# Intelligent Agents: Reflex-Based Agents for the Vacuum-cleaner World

**Nhiệm vụ 1: Triển khai một môi trường mô phỏng**

import numpy as np

import random

def vacuum\_environment(agent\_function, room\_size=5, dirt\_probability=0.2, max\_steps=1000, verbose=False):

    """

    PEAS robot vacuum cleaner simulation environment

    Args:

        agent\_function: agent function receive (bumpers , dirty) and return action

        room\_size : size of the room (default 5 x 5 )

        dirt\_probability : probability of cell containing dust (default 0.2)

        max\_steps : robot energy (default 1000)

        verbose : show info debug (default False)

    Returns:

        performance\_measure: Number of steps required to clean an entire room

    """

    # initalize  the environment

    # True = dirty, False = clean

    room = np.random.random((room\_size, room\_size)) < dirt\_probability

    # Random initial position

    agent\_x = random.randint(0, room\_size - 1)

    agent\_y = random.randint(0, room\_size - 1)

    steps = 0

    if verbose:

        print(f"The room {room\_size}x{room\_size}, initial position: ({agent\_x}, {agent\_y})")

        print("Initial room status:")

        print\_room(room, agent\_x, agent\_y)

    # Loops

    while steps < max\_steps:

        # Check if the room is dirty

        if not np.any(room):

            if verbose:

                print(f"Finish! Cleaned the entire back room {steps} step.")

            return steps

        # Creating percepts for the agent (create bumpers (vật cản))

        bumpers = {

            "north": agent\_y == 0,

            "south": agent\_y == room\_size - 1,

            "west": agent\_x == 0,

            "east": agent\_x == room\_size - 1

        }

        dirty = room[agent\_y, agent\_x]  # There is dust in the current location

        # Call agent function

        action = agent\_function(bumpers, dirty)

        if verbose:

            print(f"Step {steps}: Position ({agent\_x}, {agent\_y}), dirty: {dirty}, action: {action}")

        if action == "suck":

            room[agent\_y, agent\_x] = False

            if verbose:

                print(f"  -> Cleaned the cell ({agent\_x}, {agent\_y})")

        elif action == "north" and not bumpers["north"]:

            agent\_y -= 1

            if verbose:

                print(f"  -> Moving North: ({agent\_x}, {agent\_y})")

        elif action == "south" and not bumpers["south"]:

            agent\_y += 1

            if verbose:

                print(f"  -> Moving South: ({agent\_x}, {agent\_y})")

        elif action == "west" and not bumpers["west"]:

            agent\_x -= 1

            if verbose:

                print(f"  -> Moving West: ({agent\_x}, {agent\_y})")

        elif action == "east" and not bumpers["east"]:

            agent\_x += 1

            if verbose:

                print(f"  -> Moving East: ({agent\_x}, {agent\_y})")

        else:

            if verbose:

                print(f"  -> Invalid action or hitting a wall: {action}")

        steps += 1

    if verbose:

        print(f"Out of energy after {steps} steps. {np.sum(room)} dirty cells left.")

    return steps

def print\_room(room, agent\_x, agent\_y):

    """Print room status with robot location"""

    for y in range(room.shape[0]):

        row = ""

        for x in range(room.shape[1]):

            if x == agent\_x and y == agent\_y: # Robot initialization position

                if room[y, x]:

                    row += "R\*"   # Robot + dirty

                else:

                    row += "R "   # Robot + clean

            else: # other location

                if room[y, x]:

                    row += "\* "  # dirty

                else:

                    row += ". "  # clean

        print(row)

    print()

**Nhiệm vụ 2: Triển khai một tác nhân phản xạ đơn giản**

# Show that your environment works with the simple randomized agent from above.

def simple\_reflex\_agent(bumpers, dirty):

    """

    Simple reflex agent:

        - If there is dust: suck immediately

        - If there is no dust: move randomly but avoid hitting walls

    Args:

        bumpers (Dictionary): Collision sensors {"north": bool, "south": bool, "east": bool, "west": bool}

        dirty (bool): Is there dust at the current position

    Returns:

        action: Selection action ("suck", "north", "south", "east", "west")

    """

    # If there is dust, vacuum it immediately

    if dirty:

        return "suck"

    # If there is no dust, move immediately but avoid hitting walls

    # Create a list of possible directions to move (without walls)

    available\_directions = []

    if not bumpers["north"]:

        available\_directions.append("north")

    if not bumpers["south"]:

        available\_directions.append("south")

    if not bumpers["east"]:

        available\_directions.append("east")

    if not bumpers["west"]:

        available\_directions.append("west")

    if available\_directions:

        return np.random.choice(available\_directions)

    # If surrounded by walls

    return "suck"

**Nhiệm vụ 3: Triển khai tác nhân phản xạ dựa trên mô hình**

import math

class ModelBasedAgent:

    """Model-based reflective agent using class"""

    def \_\_init\_\_(self, room\_size = 5):

        """Init agent with room size"""

        self.x = None

        self.y = None

        self.direction = "north"

        self.visited = np.zeros((room\_size,room\_size), dtype=bool)

        self.dirt\_map = np.zeros((room\_size,room\_size), dtype=bool)

        self.model = "exploring" # "exploring or cleaning"

        self.target\_x = None

        self.target\_y = None

        self.corner\_found = None

        self.room\_size = room\_size

    def update\_position(self, action , bumpers):

        """Update position based on actions and bumpers """

        if action == "north" and not bumpers["north"]:

            self.y -= 1

            self.direction = "north"

        if action == "south" and not bumpers["south"]:

            self.y += 1

            self.direction = "south"

        if action == "west" and not bumpers["west"]:

            self.x -= 1

            self.direction = "west"

        if action == "east" and not bumpers["east"]:

            self.x += 1

            self.direction = "east"

    def find\_nearest\_dirty\_square(self):

        """Find nearest dirty square"""

        min\_distance = float('inf')

        nearest\_x, nearest\_y = None, None

        for y in range(self.room\_size):

            for x in range(self.room\_size):

                if self.dirt\_map[y,x]:

                    distance = math.sqrt((x - self.x)\*\*2 + (y - self.y)\*\*2)

                    if min\_distance > distance :

                        min\_distance = distance

                        nearest\_x , nearest\_y = x , y

        return nearest\_x , nearest\_y

    def get\_next\_move\_towards\_target(self,targetX ,targetY):

        """Calculate the next move to reach the target"""

        dx = targetX - self.x

        dy = targetY - self.y

        if dx  == 0 and dy == 0  :

            return "suck"

        if abs(dx) > abs(dy):

            if dx > 0:

                return "east"

            else:

                return "west"

        else:

            if dy > 0:

                return "south"

            else:

                return "north"

    def reset(self):

        """Reset"""

        self.x = None

        self.y = None

        self.direction = 'north'

        self.visited = np.zeros((self.room\_size, self.room\_size), dtype=bool)

        self.dirt\_map = np.zeros((self.room\_size, self.room\_size), dtype=bool)

        self.mode = 'exploring'

        self.target\_x = None

        self.target\_y = None

        self.corner\_found = False

    def get\_action(self, bumpers , dirty):

        """

        Model-based reflective agents

        Args:

            bumpers (Dictionary): Collision sensors {"north": bool, "south": bool, "east": bool, "west": bool}

            dirty (bool): Is there dust at the current position

        Returns:

            action: Selection action

        """

        # Update location if unknown

        if self.x is None:

            # assumed to be in the northwest position

            self.x = 0

            self.y = 0

            self.corner\_found = True

        # Mark current visited location

        self.visited[self.y , self.x] = True

        # dusty location

        if dirty:

            self.dirt\_map[self.y , self.x] = True

        # Case 1: If there is dust , vacuum it

        if dirty:

            self.dirt\_map[self.y,self.x] = False

            return "suck"

        # Case 2: If mode is exploring

        if self.mode == "exploring":

            #  Check if all rooms have been explored

            if np.all(self.visited):

                self.mode = "cleaning"

            else:

                # find nearest square  unvisited

                unvisited = np.where(~self.visited) # ~ is NOT logic for list Boolean

                if len(unvisited[0]) > 0 :

                    target\_y , target\_x =  unvisited[0][0], unvisited[1][0]

                    self.target\_x = target\_x

                    self.target\_y = target\_y

                    return self.get\_next\_move\_towards\_target(target\_x, target\_y)

        # Case 3: If self.mode == "cleaning"

        if self.mode == "cleaning":

            # find nearest dirty square

            nearest\_x, nearest\_y = self.find\_nearest\_dirty\_square()

            if nearest\_x is None:

                self.target\_x = nearest\_x

                self.target\_y = nearest\_y

                return self.get\_next\_move\_towards\_target(target\_x, target\_y)

            else:

                return "suck"

        # Case 4 : Move randomly without a clear goal

        available\_directions = []

        if not bumpers["north"]:

            available\_directions.append("north")

        if not bumpers["south"]:

            available\_directions.append("south")

        if not bumpers["east"]:

            available\_directions.append("east")

        if not bumpers["west"]:

            available\_directions.append("west")

        if available\_directions:

            return np.random.choice(available\_directions)

        return "suck"

# Create instance of agent

model\_agent = ModelBasedAgent(room\_size=5)

# Wapper function

def model\_based\_reflex\_agent(bumpers, dirty):

    return model\_agent.get\_action(bumpers=bumpers , dirty= dirty)

**Nhiệm vụ 4: Nghiên cứu mô phỏng**

import matplotlib.pyplot as plt

import pandas as pd

import time

def run\_simulation\_study():

    """Simulation Study with differences between agents"""

    room\_sizes = [5,10,100]

    num\_runs = 100

    result = {

        "Room\_size" : [],

        "Randomized\_Agent": [],

        "Simple\_Reflex\_Agent": [],

        "Model-based\_Reflex\_Agent": [],

    }

    print("=== Start ===")

    for room\_size in room\_sizes:

        print(f"\n --- Checking {room\_size}x{room\_size} ---")

        print(" --- Simple randomized agent ---")

        random\_performances =[]

        for i in range(num\_runs):

            perf = vacuum\_environment(simple\_randomized\_agent, room\_size= room\_size)

            random\_performances.append(perf)

            if (i + 1) % 20 == 0 :

                print(f" Finish {i + 1}/{num\_runs} steps")

        print(" --- Simple reflex agent --- ")

        reflex\_performances = []

        for i in range(num\_runs):

            perf = vacuum\_environment(simple\_reflex\_agent ,room\_size = room\_size)

            reflex\_performances.append(perf)

            if (i + 1) % 20 == 0 :

                print(f" Finish {i + 1}/{num\_runs} steps")

        print(" --- Model-base reflex agent --- ")

        model\_base\_performances = []

        for i in range(num\_runs):

            perf = vacuum\_environment(model\_based\_reflex\_agent ,room\_size = room\_size)

            model\_base\_performances.append(perf)

            if (i + 1) % 20 == 0 :

                print(f" Finish {i + 1}/{num\_runs} steps")

        # In kết quả tóm tắt cho phòng này

        print(f"\n  Kết quả {room\_size}x{room\_size}:")

        print(f"    Randomized: {np.mean(random\_performances):.2f} ± {np.std(random\_performances):.2f}")

        print(f"    Simple Reflex: {np.mean(reflex\_performances):.2f} ± {np.std(reflex\_performances):.2f}")

        print(f"    Model-based: {np.mean(model\_base\_performances):.2f} ± {np.std(model\_base\_performances):.2f}")

        result["Room\_size"].append(f"{room\_size}x{room\_size}")

        result["Randomized\_Agent"].append(f"{np.mean(random\_performances):.2f} ± {np.std(random\_performances):.2f}")

        result["Simple\_Reflex\_Agent"].append(f"{np.mean(reflex\_performances):.2f} ± {np.std(reflex\_performances):.2f}")

        result["Model-based\_Reflex\_Agent"].append(f"{np.mean(model\_base\_performances):.2f} ± {np.std(model\_base\_performances):.2f}")

    return result

# Run

start\_time = time.time()

results = run\_simulation\_study()

end\_time = time.time()

print(f"\n=== Complete the study ===")

print(f"Execution time: {end\_time - start\_time:.2f}s")

df = pd.DataFrame(results)

df.columns = ["size" , "Randomized Agent" , "Simple Reflex Agent", "Model-based Reflex Agent"]

# Your graphs and discussion of the results goes here

def split\_value(val):

    mean , std = val.split("±")

    return float(mean.strip()) , float(std.strip())

data = {}

for agent in ["Randomized\_Agent" , "Simple\_Reflex\_Agent" , "Model-based\_Reflex\_Agent"]:

    means, stds = zip(\*[split\_value(v) for v in results[agent]])

    data[agent] = {"mean": means , "std": stds}

df\_means = pd.DataFrame({agent: data[agent]["mean"] for agent in data} , index = results['Room\_size'])

df\_stds = pd.DataFrame({agent: data[agent]["std"] for agent in data} , index = results['Room\_size'])

# Vẽ biểu đồ

ax = df\_means.plot(kind="bar", yerr=df\_stds, capsize=5, figsize=(8,5))

plt.title("So sánh hiệu suất của các tác tử hút bụi")

plt.ylabel("Số bước (mean ± std)")

plt.xlabel("Kích thước phòng")

plt.xticks(rotation=0)

plt.legend(title="Agent")

plt.tight\_layout()

plt.show()

**Nhiệm vụ 5: Tính mạnh mẽ của việc triển khai tác nhân**

Viết mô tả như thế nào để triển khai tác nhân mạnh mẽ

1. Rectangular room with unknown size (Phòng hình chữ nhật với kích thước không biết)

    - Randomized Agent: vẫn hoạt động được, vì nó chỉ di chuyển ngẫu nhiên và hút bụi khi gặp bụi. Tuy nhiên hiệu quả rất thấp, có thể tốn nhiều bước hoặc không bao giờ làm sạch toàn bộ.

    - Simple Reflex Agent: hoạt động tốt nếu quy tắc được thiết kế dựa trên bumper (cảm biến tường), vì tác tử có thể tránh va chạm và tiếp tục di chuyển. Nhưng nó không có khả năng “nhớ” nên dễ bỏ sót các khu vực.

    - Model-based Reflex Agent: hiệu quả hơn vì có mô hình nội bộ để đánh dấu các ô đã thăm. Nếu không biết kích thước ban đầu, agent có thể dần khám phá biên của phòng nhờ bumper và mở rộng bản đồ bên trong.

2. Irregular-shaped cleaning area (e.g., hallway connecting two rooms) (Khu vực dọn dẹp hình dạng không đều)

    - Randomized Agent: có thể tình cờ khám phá hết, nhưng mất rất nhiều thời gian và không đảm bảo.

    - Simple Reflex Agent: dễ bị mắc kẹt hoặc không quay lại các khu vực hẹp, vì không có trí nhớ.

    - Model-based Reflex Agent: có khả năng xây dựng bản đồ không gian đã thăm, nên sẽ tìm được đường quay lại và dọn sạch được cả khu vực hành lang hay các nhánh phòng phức tạp.

3. Room with obstacles (phòng với nhiều trở ngại)

    - Randomized Agent: chỉ phản ứng khi bumper báo tường hoặc vật cản, nên có thể bị kẹt ở một khu vực hẹp.

    - Simple Reflex Agent: sẽ tránh chướng ngại vật nhờ bumper, nhưng có thể đi vòng lại nhiều lần hoặc bỏ sót khu vực phía sau chướng ngại.

    - Model-based Reflex Agent: sẽ ghi nhận các ô không thể đi qua trong mô hình nội bộ, từ đó tìm đường vòng qua chướng ngại để đến các ô chưa dọn. Đây là tác tử phù hợp nhất cho môi trường có vật cản.

4. Imperfect dirt sensor (10% error) (Cảm biến bụi bẩn không hoàn hảo (lỗi 10%))

    - Randomized Agent: có thể hút nhầm nơi không bẩn hoặc bỏ sót bụi, làm giảm hiệu quả tổng thể.

    - Simple Reflex Agent: dễ bị ảnh hưởng vì không có cách nào xác minh lại; nếu cảm biến báo sai thì hành động cũng sai.

    - Model-based Reflex Agent: ít bị ảnh hưởng hơn, vì có thể lưu lại thông tin nhiều lần về tình trạng bụi ở một ô. Nếu gặp bất nhất (lúc báo bẩn, lúc báo sạch), agent có thể kiểm chứng lại bằng cách quay lại ô đó.

5. Imperfect bumper sensor (10% error) (Cảm biến cản không hoàn hảo (lỗi 10%))

    - Randomized Agent: có nguy cơ thử đi xuyên qua tường hoặc chướng ngại vật khi bumper báo sai, dẫn đến trạng thái “kẹt” hoặc lặp vô hạn.

    - Simple Reflex Agent: cũng gặp vấn đề tương tự, vì phản ứng hoàn toàn phụ thuộc vào cảm biến. Nếu bumper bỏ sót tường, agent sẽ đưa ra hành động không hợp lệ.

    - Model-based Reflex Agent: có thể bền vững hơn nhờ “nhớ” vị trí tường đã gặp trước đó. Nếu một lần cảm biến báo sai, thông tin từ những lần quan sát khác sẽ giúp giảm tác động của lỗi.

# Create a Simple Reflex-Based Lunar Lander Agent

**Triển khai một tác nhân phản xạ tốt hơn**

def better\_rocket\_agent\_function(observation):

    X = observation[Obs.X.value]    # vị trí ngang

    Y = observation[Obs.Y.value]    # vị trí dọc

    VX = observation[Obs.VX.value]  # vận tốc ngang

    VY = observation[Obs.VY.value]  # vận tốc dọc

    ANGLE = observation[Obs.ANGLE.value]    # góc nghiêng

    ANGULAR\_VELOCITY = observation[Obs.ANGULAR\_VELOCITY.value]  # vận tốc quay

    LEFT\_LEG\_CONTACT = observation[Obs.LEFT\_LEG\_CONTACT.value]  # nếu chân trái chạm đất

    RIGHT\_LEG\_CONTACT = observation[Obs.RIGHT\_LEG\_CONTACT.value]    # nếu chân phải chạm đất

    # Decision Threshold

    ANGLE\_THRESHOLD = 0.1

    VX\_THRESHOLD = 0.2

    VY\_THRESHOLD = 0.5

    X\_THRESHOLD = 0.2

    if X == 0 and Y == 0 :

        return Act.NO\_OP.value

    # 1. Điều chỉnh khi bị nghiêng

    if ANGLE > ANGLE\_THRESHOLD:

        return Act.RIGHT.value

    elif ANGLE < -ANGLE\_THRESHOLD:

        return Act.LEFT.value

    # 2. Điều chỉnh vị trí ngang về 0

    elif abs(X) > X\_THRESHOLD:

        if X > 0:

            return Act.RIGHT.value

        else:

            return Act.LEFT.value

    # 3. Kiểm soát vận tốc

    elif abs(VX) > VX\_THRESHOLD and Y < 1.0:

        if VX > 0:

            return Act.RIGHT.value

        else:

            return Act.LEFT.value

    # 4. Kiểm soát vận tốc rơi

    elif VY <  -VY\_THRESHOLD:

        return Act.MAIN.value

    # 5. Không làm gì cả

    else:

        return Act.NO\_OP.value

print("=== Test ===")

rewards = run\_episode(better\_rocket\_agent\_function , 150)

rewards