Додаток А. Лістинг коду програми

Лістинг коду віртуального макету:

– GA:

```
from deap import base, algorithms
from deap import creator
from deap import tools

import algelitism

import random
import matplotlib.pyplot as plt
import numpy as np

import gym

env = gym.make("MountainCar-v0")

LENGTH_CHROM = 200    # довжина хромосоми, підлеглої оптимізації

# константи генетичного алгоритму
POPULATION_SIZE = 50   # кількість індивидів в популяції
P_CROSSOVER = 0.9       # вірогідність схрещення
P_MUTATION = 0.4        # вірогідність мутації індивіда
MAX_GENERATIONS = 150    # максимальна кількість поколінь
HALL_OF_FAME_SIZE = 3

hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

RANDOM_SEED = 42
random.seed(RANDOM_SEED)

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)


toolbox = base.Toolbox()
toolbox.register("randomAction", random.randint, 0, 2)
toolbox.register("individualCreator", tools.initRepeat, creator.Individual,
toolbox.randomAction, LENGTH_CHROM)
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)
```

```
population = toolbox.populationCreator(n=POPULATION_SIZE)


def getCarScore(individual):
    FLAG_LOCATION = 0.5
    observation = env.reset()
    actionCounter = 0

    for action in individual:
        actionCounter += 1
        observation, reward, done, info = env.step(action)

        if done:
            break

    if actionCounter < LENGTH_CHROM:
        score = 0 - (LENGTH_CHROM - actionCounter) / LENGTH_CHROM
    else:
        score = abs(observation[0] - FLAG_LOCATION)

    return score,


toolbox.register("evaluate", getCarScore)
toolbox.register("select", tools.selTournament, tournsize=2)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=0, up=2,
indpb=1.0/LENGTH_CHROM)

stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("min", np.min)
stats.register("avg", np.mean)


#algelitism.eaSimpleElitism
#algorithms.eaSimple
population, logbook = algelitism.eaSimpleElitism(population, toolbox,
                                  cxpb=P_CROSSOVER,
                                  mutpb=P_MUTATION,
                                  ngen=MAX_GENERATIONS,
                                  halloffame=hof,
```

```
                    stats=stats,
                    verbose=True)

maxFitnessValues, meanFitnessValues = logbook.select("min", "avg")

best = hof.items[0]
print(best)

plt.plot(maxFitnessValues, color='red')
plt.plot(meanFitnessValues, color='green')
plt.xlabel('Покоління')
plt.ylabel('Макс/середня пристосованість')
plt.title('Залежність максимальної та середньої пристосовності від покоління')
plt.show()

observation = env.reset()

for action in best:
    env.step(action)
    env.render()

env.close()
```

– ALGELITISM:

```
from deap import tools
from deap.algorithms import varAnd


def eaSimpleElitism(population, toolbox, cxpb, mutpb, ngen, stats=None,
            halloffame=None, verbose=__debug__, callback=None):

    """Перероблений алгоритм eaSimple с элементом елітизму"""

    logbook = tools.Logbook()
    logbook.header = ['gen', 'nevals'] + (stats.fields if stats else [])

    # Evaluate the individuals with an invalid fitness
    invalid_ind = [ind for ind in population if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
```

```
for ind, fit in zip(invalid_ind, fitnesses):
    ind.fitness.values = fit

if halloffame is not None:
    halloffame.update(population)

hof_size = len(halloffame.items) if halloffame.items else 0

record = stats.compile(population) if stats else {}
logbook.record(gen=0, nevals=len(invalid_ind), **record)
if verbose:
    print(logbook.stream)

# Begin the generational process
for gen in range(1, ngen + 1):
    # Select the next generation individuals
    offspring = toolbox.select(population, len(population) - hof_size)

    # Vary the pool of individuals
    offspring = varAnd(offspring, toolbox, cxpb, mutpb)

    # Evaluate the individuals with an invalid fitness
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    offspring.extend(halloffame.items)

    # Update the hall of fame with the generated individuals
    if halloffame is not None:
        halloffame.update(offspring)

    # Replace the current population by the offspring
    population[:] = offspring

    # Append the current generation statistics to the logbook
    record = stats.compile(population) if stats else {}
    logbook.record(gen=gen, nevals=len(invalid_ind), **record)
    if verbose:
        print(logbook.stream)
```

```
        if callback:
            callback[0](*callback[1])

    return population, logbook
```

Лістинг коду фізичного макету:

– Без ГА:

```
#include <Servo.h>

        #define ECHO_PIN  A4

        #define TRIG_PIN  A5

        #define ENA 5

        #define ENB 6

        #define IN1 7

        #define IN2 8

        #define IN3 9

        #define IN4 11

        #define LED_Pin 13

        #define LineTeacking_Pin_Right  10

        #define LineTeacking_Pin_Middle 4

        #define LineTeacking_Pin_Left   2

        #define LineTeacking_Read_Right   !digitalRead(10)

        #define LineTeacking_Read_Middle  !digitalRead(4)
```

```
#define LineTeacking_Read_Left    !digitalRead(2)
#define carSpeed 200

Servo servo;
unsigned long LT_PreMillis;
int rightDistance = 0, leftDistance = 0, middleDistance = 0;

enum FUNCTIONMODE {
  IDLE,
  LineTeacking,
  ObstaclesAvoidance
} func_mode = IDLE;

enum MOTIONMODE {
  STOP,
  FORWARD,
  BACK,
  LEFT,
  RIGHT
} mov_mode = STOP;

void delays(unsigned long t) {
```

```
  delay(t);
}

int getDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  return (int)pulseIn(ECHO_PIN, HIGH) / 58;
}

void forward(bool debug = false) {
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  if (debug) Serial.println("Go forward!");
}
```

```
void back(bool debug = false) {
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  if (debug) Serial.println("Go back!");
}

void left(bool debug = false) {
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  if (debug) Serial.println("Go left!");
}

void right(bool debug = false) {
  analogWrite(ENA, carSpeed);
```

```
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  if (debug) Serial.println("Go right!");
}

void stop(bool debug = false) {
  digitalWrite(ENA, LOW);
  digitalWrite(ENB, LOW);
  if (debug) Serial.println("Stop!");
}

void line_teacking_mode() {
  if (func_mode == LineTeacking) {
    if (LineTeacking_Read_Middle) {
      forward();
      LT_PreMillis = millis();
    } else if (LineTeacking_Read_Right) {
      right();
      while (LineTeacking_Read_Right) {
```

```
        delay(1);
      }
      LT_PreMillis = millis();
    } else if (LineTeacking_Read_Left) {
      left();
      while (LineTeacking_Read_Left) {
        delay(1);
      }
      LT_PreMillis = millis();
    } else {
      if (millis() - LT_PreMillis > 150) {
        stop();
      }
    }
  }
}

void obstacles_avoidance_mode() {
  if (func_mode == ObstaclesAvoidance) {
    servo.write(90);
    delays(500);
    middleDistance = getDistance();
```

```
if (middleDistance <= 40) {
  stop();
  delays(500);
  servo.write(10);
  delays(500);
  rightDistance = getDistance();

  delays(500);
  servo.write(90);
  delays(500);
  servo.write(170);
  delays(500);
  leftDistance = getDistance();

  delays(500);
  servo.write(90);
  delays(500);
  if (rightDistance > leftDistance) {
    stop();
    delay(100);
    back();
    delay(200);
```

```
      right();
      delay(300);
     } else if (rightDistance < leftDistance) {
      stop();
      delay(100);
      back();
      delay(200);
      left();
      delay(300);
     } else if ((rightDistance <= 40) || (leftDistance <= 40)) {
      back();
      delays(180);
     } else {
      forward();
     }
    } else {
     forward();
    }
  }
}

void setup() {
```

```
  Serial.begin(9600);
  servo.attach(3, 500, 2400); // 500: 0 degree  2400: 180 degree
  servo.write(90);
  pinMode(ECHO_PIN, INPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(LineTeacking_Pin_Right, INPUT);
  pinMode(LineTeacking_Pin_Middle, INPUT);
  pinMode(LineTeacking_Pin_Left, INPUT);
}

void loop() {
  line_teacking_mode();
  obstacles_avoidance_mode();
}
```

– 3 ГА:

```
#include <Servo.h>
#define ECHO_PIN  A4
#define TRIG_PIN  A5
#define ENA 5
#define ENB 6
#define IN1 7
#define IN2 8
#define IN3 9
#define IN4 11
#define LED_Pin 13
#define LineTeacking_Pin_Right  10
#define LineTeacking_Pin_Middle 4
#define LineTeacking_Pin_Left   2
#define LineTeacking_Read_Right   !digitalRead(10)
#define LineTeacking_Read_Middle  !digitalRead(4)
#define LineTeacking_Read_Left    !digitalRead(2)
#define carSpeed 200

Servo servo;
unsigned long LT_PreMillis;
int rightDistance = 0, leftDistance = 0, middleDistance = 0;
```

```
enum FUNCTIONMODE {
  IDLE,
  LineTeacking,
  ObstaclesAvoidance
} func_mode = IDLE;

enum MOTIONMODE {
  STOP,
  FORWARD,
  BACK,
  LEFT,
  RIGHT
} mov_mode = STOP;

const int populationSize = 10;
const int genomeLength = 5;
int population[populationSize][genomeLength];
float fitness[populationSize];

void initializePopulation() {
  for (int i = 0; i < populationSize; i++) {
    for (int j = 0; j < genomeLength; j++) {
```

```
      population[i][j] = random(50, 255);
    }
  }
}

void evaluateFitness() {
  for (int i = 0; i < populationSize; i++) {
    fitness[i] = measurePerformance(population[i]);
  }
}

float measurePerformance(int genome[]) {
  float performance = 0.0;
  for (int i = 0; i < genomeLength; i++) {
    forwardWithSpeed(genome[i]);
    performance += getDistance();
    delay(genome[i]);
  }
  return performance;
}

void forwardWithSpeed(int speed) {
```

```
  analogWrite(ENA, speed);
  analogWrite(ENB, speed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}

void crossover(int parent1[], int parent2[], int child[]) {
  for (int i = 0; i < genomeLength; i++) {
    if (random(2) == 0) {
      child[i] = parent1[i];
    } else {
      child[i] = parent2[i];
    }
  }
}

void mutate(int genome[]) {
  for (int i = 0; i < genomeLength; i++) {
    if (random(10) < 1) {
      genome[i] = random(50, 255);
```

```
    }
  }
}

void geneticAlgorithm() {
  initializePopulation();
  for (int generation = 0; generation < 100; generation++) {
    evaluateFitness();

    int newPopulation[populationSize][genomeLength];
    for (int i = 0; i < populationSize / 2; i++) {
      int parent1Index = selectParent();
      int parent2Index = selectParent();
      crossover(population[parent1Index], population[parent2Index],
newPopulation[i]);
      mutate(newPopulation[i]);
    }

    for (int i = 0; i < populationSize; i++) {
      for (int j = 0; j < genomeLength; j++) {
        population[i][j] = newPopulation[i][j];
      }
    }
```

```
  }
}

int selectParent() {
  int best = random(populationSize);
  for (int i = 0; i < 3; i++) {
    int contender = random(populationSize);
    if (fitness[contender] > fitness[best]) {
      best = contender;
    }
  }
  return best;
}
void setup() {
  Serial.begin(9600);
  servo.attach(3, 500, 2400);
  servo.write(90);
  pinMode(ECHO_PIN, INPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
```

```
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(LineTeacking_Pin_Right, INPUT);
  pinMode(LineTeacking_Pin_Middle, INPUT);
  pinMode(LineTeacking_Pin_Left, INPUT);
  geneticAlgorithm();
}

void loop() {
  line_teacking_mode();
  obstacles_avoidance_mode();
}
```

Додаток Б. Демонстраційні графічні матеріали

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк.	Примітки
			Текстові документи		
А4	1	ГЮІК 010115.013 ПЗ	Пояснювальна записка	92	с
			Графічні документи		
	2		Демонстраційний матеріал у вигляді презентації	12	формат *.ppt

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮІК 010115.013 ВД			
Розроб.		Пара			Розробка системи автоматизації керування металошукачем з використанням Arduino	Літера	Арку	Аркуш
Перевір.		Гурін				Н	1	1
Н.контр.		Бронніков				Кафедра КІТАР ХНУРЕ		
Затв.		Невлюдов						