CSC4702

ROBOTIC SYSTEM DEVELOPMENT

SEMESTER 1, 2025/2026

ASSIGNMENT 2

LECTURER'S NAME :

DR. ERZAM BIN MARLISAH

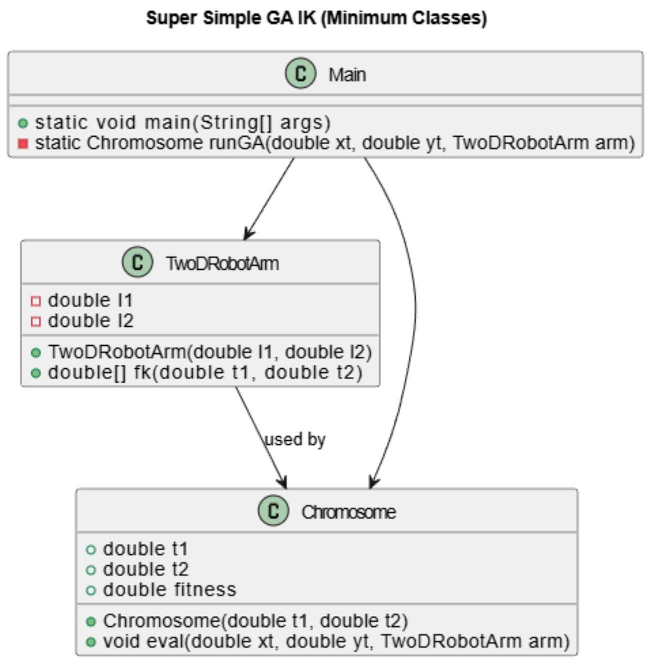
NAME	MATRIC NUMBER
CAI ZESHUO	213733

1. Introduction

Inverse kinematics is the problem of determining the joint angles of a robot arm so that its end effector reaches a desired target position. In this assignment, a two-link planar robot arm is considered. Instead of using a closed-form mathematical solution, a Genetic Algorithm (GA) is applied to search for the joint angles that minimize the distance between the target position and the position obtained from forward kinematics.

Genetic Algorithm is a population-based optimization technique inspired by natural evolution. It is suitable for solving optimization problems where analytical solutions are not required.

2. System Architecture


Super Simple GA IK (Minimum Classes)
Main
static void main(String[] args)
static Chromosome runGA(double xt, double yt, TwoDRobotArm arm)
TwoDRobotArm
double l1
double l2
TwoDRobotArm(double l1, double l2)
double[] fk(double t1, double t2)
used by
Chromosome
double t1
double t2
double fitness
Chromosome(double t1, double t2)
void eval(double xt, double yt, TwoDRobotArm arm)

The system is implemented using a simple object-oriented design with three main Java files:

- TwoDRobotArm.java

  This class represents the two-link planar robot arm and computes the end effector position using forward kinematics.

- Chromosome.java
  This class represents a candidate solution containing two joint angles (θ_1 and θ_2) and its fitness value.

- Main.java
  This class contains the Genetic Algorithm logic. It initializes the population, performs selection, crossover, mutation, and prints the results for each generation.

3. Forward Kinematics

The robot arm consists of two links with lengths l1l_1l1 and l2l_2l2. The end effector position is calculated using the following equations:

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

4. Genetic Algorithm Implementation

Each chromosome stores two real-valued genes representing the joint angles θ_1 and θ_2. The fitness function is defined as the Euclidean distance between the target position and the end effector position computed using forward kinematics. A smaller fitness value indicates a better solution.

Tournament selection is used to select parent chromosomes. Arithmetic crossover is applied to generate offspring by combining parent joint angles. Mutation is implemented by adding a small random perturbation to the joint angles to maintain population diversity and avoid premature convergence.

5. Genetic Algorithm Parameters

The parameters used in the implementation are shown below:

Parameter	Value
Population Size	100
Maximum Generations	500
Crossover Rate	0.9
Mutation Rate	0.2
Mutation Step	0.05 radians
Angle Range	[−π,π][-π, π][−π,π]

These parameters were chosen to ensure stable and fast convergence.

6. Experimental Results

The Genetic Algorithm was tested using three different target positions. For each target, the algorithm successfully found joint angles that produced an end effector position with an error less than 0.01 units.

Target Position (x, y)	Final Error
(1.2, 0.5)	< 0.01
(0.8, 1.0)	< 0.01
(1.5, 0.2)	< 0.01

Console output screenshots showing the convergence process for each target are included in the report.

7. Convergence Discussion

From the experimental results, it can be observed that the error decreases rapidly during the early generations. This shows that the Genetic Algorithm is effective in searching the solution space. The use of tournament selection and elitism helps preserve good solutions, while mutation allows further refinement of the joint angles.

The algorithm typically converges within a small number of generations and stops early once the required accuracy is achieved.

8. Conclusion

This assignment demonstrates that a Genetic Algorithm can be successfully used to solve the inverse kinematics problem of a two-link planar robot arm. Despite the simple system architecture, the algorithm is able to achieve accurate results for multiple target positions. The simplified object-oriented design makes the program easy to understand and maintain while still satisfying all assignment requirements.

Source Code:

TwoDRobotArm.java

```java
public class TwoDRobotArm {

   private final double l1, l2;



   public TwoDRobotArm(double l1, double l2) {

       this.l1 = l1;

       this.l2 = l2;

   }



   // returns {x, y}

   public double[] fk(double t1, double t2) {

       double x = l1 * Math.cos(t1) + l2 * Math.cos(t1 + t2);

       double y = l1 * Math.sin(t1) + l2 * Math.sin(t1 + t2);

       return new double[]{x, y};

   }
}
```

Chromosome.java

```
public class Chromosome {

   public double t1, t2;

   public double fitness; // lower is better



   public Chromosome(double t1, double t2) {

       this.t1 = t1;

       this.t2 = t2;

       this.fitness = Double.POSITIVE_INFINITY;

   }



   public void eval(double xt, double yt, TwoDRobotArm arm) {

       double[] p = arm.fk(t1, t2);

       double dx = xt - p[0];

       double dy = yt - p[1];

       fitness = Math.sqrt(dx * dx + dy * dy);

   }

}
```

Main.java

```java
import java.util.Random;


public class Main {

   static Random rng = new Random();


   public static void main(String[] args) {

       TwoDRobotArm arm = new TwoDRobotArm(1.0, 1.0);


       runOne(1.2, 0.5, arm);

       System.out.println("----------------------------------");

       runOne(0.8, 1.0, arm);

       System.out.println("----------------------------------");

       runOne(1.5, 0.2, arm);

   }


   static void runOne(double xt, double yt, TwoDRobotArm arm) {

       System.out.printf("Target: (%.6f, %.6f)%n", xt, yt);

       Chromosome best = runGA(xt, yt, arm);


       double[] pos = arm.fk(best.t1, best.t2);

       System.out.println("\nFINAL");

       System.out.printf("theta1=%.6f, theta2=%.6f%n", best.t1, best.t2);
```

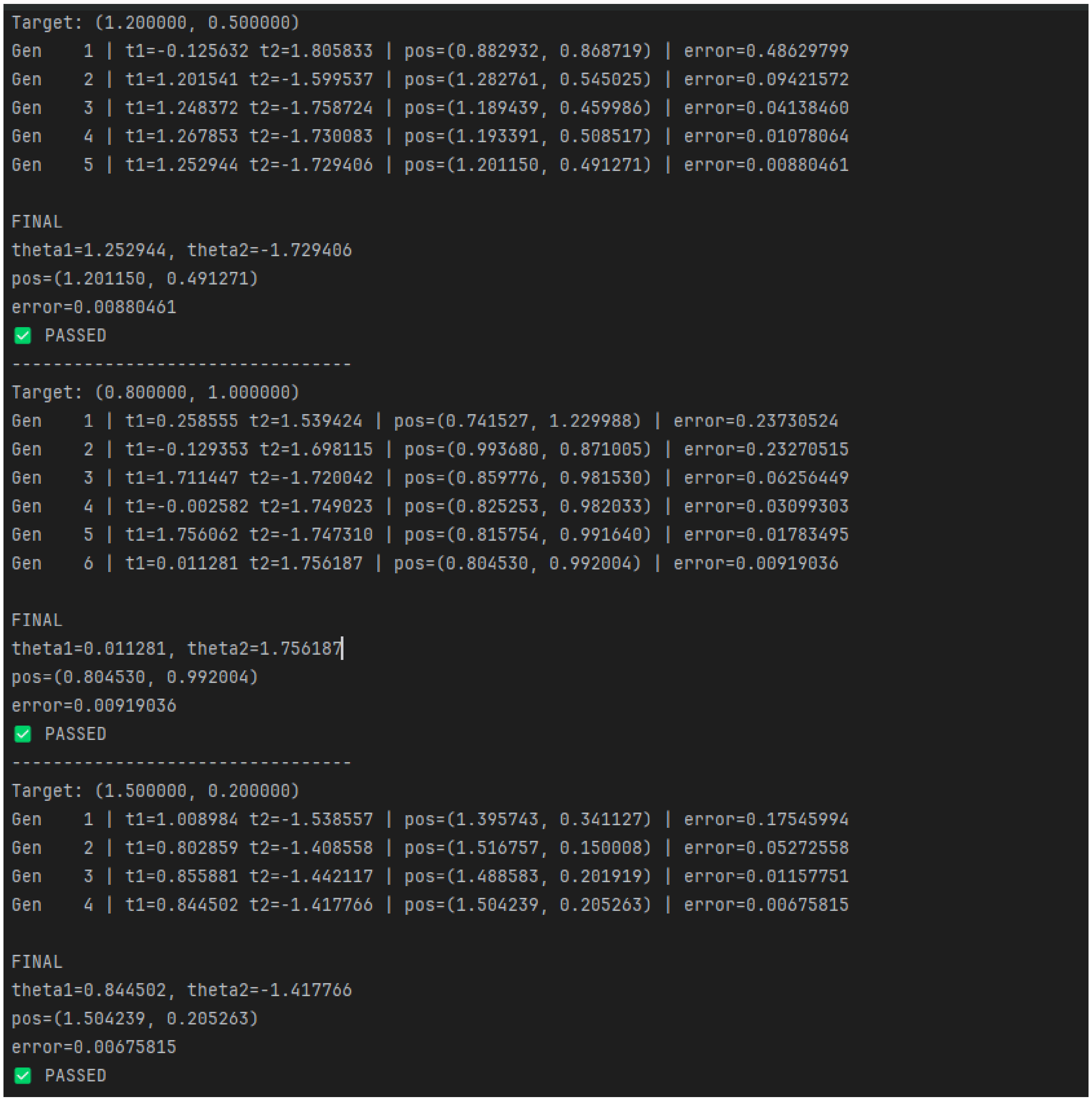
```
        System.out.printf("pos=(%.6f, %.6f)%n", pos[0], pos[1]);

        System.out.printf("error=%.8f%n", best.fitness);

        System.out.println(best.fitness < 0.01 ? "✅ PASSED" : "❌ NOT PASSED");

    }



    // GA parameters (keep simple)

    static Chromosome runGA(double xt, double yt, TwoDRobotArm arm) {

        int popSize = 100;

        int maxGen = 500;

        double crossRate = 0.9;

        double mutRate = 0.2;

        double mutStep = 0.05;

        double minA = -Math.PI, maxA = Math.PI;

        int tournamentK = 5;



        Chromosome[] pop = new Chromosome[popSize];

        for (int i = 0; i < popSize; i++) {

            pop[i] = new Chromosome(rand(minA, maxA), rand(minA, maxA));

            pop[i].eval(xt, yt, arm);

        }



        for (int gen = 1; gen <= maxGen; gen++) {
            Chromosome best = bestOf(pop);
            double[] bpos = arm.fk(best.t1, best.t2);

               System.out.printf("Gen %4d | t1=%.6f t2=%.6f | pos=(%.6f, %.6f) |
error=%.8f%n",
                    gen, best.t1, best.t2, bpos[0], bpos[1], best.fitness);

            if (best.fitness < 0.01) return best;

            Chromosome[] next = new Chromosome[popSize];
            next[0] = copy(best); // elitism

            for (int i = 1; i < popSize; i++) {
                Chromosome p1 = tournament(pop, tournamentK);
                Chromosome p2 = tournament(pop, tournamentK);

                Chromosome child;
                if (rng.nextDouble() < crossRate) {
                    double a = rng.nextDouble();
                    double t1 = clamp(a * p1.t1 + (1 - a) * p2.t1, minA, maxA);
                    double t2 = clamp(a * p1.t2 + (1 - a) * p2.t2, minA, maxA);
                    child = new Chromosome(t1, t2);
                } else {
                    child = copy(rng.nextBoolean() ? p1 : p2);
                }

                // mutation
                    if (rng.nextDouble() < mutRate) child.t1 = clamp(child.t1 +
perturb(mutStep), minA, maxA);
                    if (rng.nextDouble() < mutRate) child.t2 = clamp(child.t2 +
perturb(mutStep), minA, maxA);


                child.eval(xt, yt, arm);
                next[i] = child;
            }

            pop = next;
        }

        return bestOf(pop);
    }

    // -------- helper methods --------
    static Chromosome tournament(Chromosome[] pop, int k) {
        Chromosome best = null;
        for (int i = 0; i < k; i++) {
```

```java
            Chromosome c = pop[rng.nextInt(pop.length)];

            if (best == null || c.fitness < best.fitness) best = c;

        }

        return copy(best);

    }



    static Chromosome bestOf(Chromosome[] pop) {

        Chromosome best = pop[0];

        for (Chromosome c : pop) if (c.fitness < best.fitness) best = c;

        return best;

    }



    static Chromosome copy(Chromosome c) {

        Chromosome x = new Chromosome(c.t1, c.t2);

        x.fitness = c.fitness;

        return x;

    }



    static double rand(double a, double b) {

        return a + rng.nextDouble() * (b - a);

    }
    static double perturb(double step) {

        return (rng.nextDouble() * 2 - 1) * step;

    }



    static double clamp(double v, double min, double max) {

        if (v < min) return min;

        if (v > max) return max;

        return v;

    }

}
```

Output:

```
Target: (1.200000, 0.500000)

Gen    1 | t1=-0.125632 t2=1.805833 | pos=(0.882932, 0.868719) | error=0.48629799

Gen    2 | t1=1.201541 t2=-1.599537 | pos=(1.282761, 0.545025) | error=0.09421572

Gen    3 | t1=1.248372 t2=-1.758724 | pos=(1.189439, 0.459986) | error=0.04138460

Gen    4 | t1=1.267853 t2=-1.730083 | pos=(1.193391, 0.508517) | error=0.01078064

Gen    5 | t1=1.252944 t2=-1.729406 | pos=(1.201150, 0.491271) | error=0.00880461


FINAL

theta1=1.252944, theta2=-1.729406
```

```
pos=(1.201150, 0.491271)

error=0.00880461

✅ PASSED

---------------------------------

Target: (0.800000, 1.000000)

Gen    1 | t1=0.258555 t2=1.539424 | pos=(0.741527, 1.229988) | error=0.23730524

Gen    2 | t1=-0.129353 t2=1.698115 | pos=(0.993680, 0.871005) | error=0.23270515

Gen    3 | t1=1.711447 t2=-1.720042 | pos=(0.859776, 0.981530) | error=0.06256449

Gen    4 | t1=-0.002582 t2=1.749023 | pos=(0.825253, 0.982033) | error=0.03099303

Gen    5 | t1=1.756062 t2=-1.747310 | pos=(0.815754, 0.991640) | error=0.01783495

Gen    6 | t1=0.011281 t2=1.756187 | pos=(0.804530, 0.992004) | error=0.00919036


FINAL

theta1=0.011281, theta2=1.756187

pos=(0.804530, 0.992004)

error=0.00919036

✅ PASSED

---------------------------------

Target: (1.500000, 0.200000)

Gen    1 | t1=1.008984 t2=-1.538557 | pos=(1.395743, 0.341127) | error=0.17545994
Gen    2 | t1=0.802859 t2=-1.408558 | pos=(1.516757, 0.150008) | error=0.05272558

Gen    3 | t1=0.855881 t2=-1.442117 | pos=(1.488583, 0.201919) | error=0.01157751

Gen    4 | t1=0.844502 t2=-1.417766 | pos=(1.504239, 0.205263) | error=0.00675815


FINAL

theta1=0.844502, theta2=-1.417766

pos=(1.504239, 0.205263)

error=0.00675815

✅ PASSED


Process finished with exit code 0
```



```
Target: (1.200000, 0.500000)
Gen    1 | t1=-0.125632 t2=1.805833 | pos=(0.882932, 0.868719) | error=0.48629799
Gen    2 | t1=1.201541 t2=-1.599537 | pos=(1.282761, 0.545025) | error=0.09421572
Gen    3 | t1=1.248372 t2=-1.758724 | pos=(1.189439, 0.459986) | error=0.04138460
Gen    4 | t1=1.267853 t2=-1.730083 | pos=(1.193391, 0.508517) | error=0.01078064
Gen    5 | t1=1.252944 t2=-1.729406 | pos=(1.201150, 0.491271) | error=0.00880461

FINAL
theta1=1.252944, theta2=-1.729406
pos=(1.201150, 0.491271)
error=0.00880461
✅ PASSED
---------------------------------
Target: (0.800000, 1.000000)
Gen    1 | t1=0.258555 t2=1.539424 | pos=(0.741527, 1.229988) | error=0.23730524
Gen    2 | t1=-0.129353 t2=1.698115 | pos=(0.993680, 0.871005) | error=0.23270515
Gen    3 | t1=1.711447 t2=-1.720042 | pos=(0.859776, 0.981530) | error=0.06256449
Gen    4 | t1=-0.002582 t2=1.749023 | pos=(0.825253, 0.982033) | error=0.03099303
Gen    5 | t1=1.756062 t2=-1.747310 | pos=(0.815754, 0.991640) | error=0.01783495
Gen    6 | t1=0.011281 t2=1.756187 | pos=(0.804530, 0.992004) | error=0.00919036

FINAL
theta1=0.011281, theta2=1.756187
pos=(0.804530, 0.992004)
error=0.00919036
✅ PASSED
---------------------------------
Target: (1.500000, 0.200000)
Gen    1 | t1=1.008984 t2=-1.538557 | pos=(1.395743, 0.341127) | error=0.17545994
Gen    2 | t1=0.802859 t2=-1.408558 | pos=(1.516757, 0.150008) | error=0.05272558
Gen    3 | t1=0.855881 t2=-1.442117 | pos=(1.488583, 0.201919) | error=0.01157751
Gen    4 | t1=0.844502 t2=-1.417766 | pos=(1.504239, 0.205263) | error=0.00675815

FINAL
theta1=0.844502, theta2=-1.417766
pos=(1.504239, 0.205263)
error=0.00675815
✅ PASSED
```