Санкт-Петербургский государственный университет информационных технологий,
механики и оптики
Факультет информационных технологий и программирования
Кафедра «Компьютерные технологии»

**Г. О. Чикишев, А. А. Сизиков**

# Моделирование движения человекоподобного робота при помощи генетических алгоритмов

Санкт-Петербург
2009

Введение ..... 3
1. Постановка задачи ..... 4
1.1. Описание среды Webots ..... 4
1.2. Описание модели движения робота ..... 4
2. Генетический алгоритм ..... 6
2.1. Общие понятия ..... 6
2.2. Использованные методы ..... 6
3. Анализ результатов ..... 8
3.1. Полученные данные ..... 8
3.2. Анализ данных ..... 9
Заключение ..... 10
Источники ..... 11
Приложение ..... 12

# Введение

В данной курсовой работе показано применение генетических алгоритмов для моделирования движения робота.

Использование виртуальной трехмерной модели позволяет спроектировать робота, способного совершать определенные движения, не конструируя его реальную модель, что экономит средства на разработке и тестировании. Поэтому получение простого и универсального метода генерации функции управления движением является актуальной задачей. Она была рассмотрена, например, в работе [1], однако описанный там подход к реализации генетического алгоритма несколько отличается от предложенного в данной работе.

В качестве среды для моделирования выбрана программа *Webots* [2], алгоритмы реализованы на языке *Java* с использованием инструментального средства *GAAP* [3].

# 1. Постановка задачи

Цель данной работы – анализ эффективности использования генетических алгоритмов для моделирования движения робота. В качестве примера был выбран шаг вперед, как основной этап движения. В качестве программного средства для симуляции была выбрана среда *Webots*. Рассмотрим эту среду и используемый в ней способ управления движением роботов.

## *1.1. Описание среды Webots*

Программное обеспечение *Webots* – это трехмерная среда симуляции, которая позволяет моделировать движение различного вида роботов: колесных, прямошагающих и летающих. Пользователи способны создавать специальное окружение для роботов и программировать их поведение. Роботы взаимодействуют со средой посредством сенсоров, данные от которых передаются управляющему контроллеру. Программа контроллера управляет поведением моторов робота и написана на языке *C++* или *Java*.

## *1.2. Описание модели движения робота*

Модель робота *Nao*, используемая в *Webots*, оснащена 22 сервомоторами. Их расположение показано на рис. 1 (сервомоторы *RWristYaw*, *LWristYaw*, *RHand* и *LHand* в симуляторе не используются).

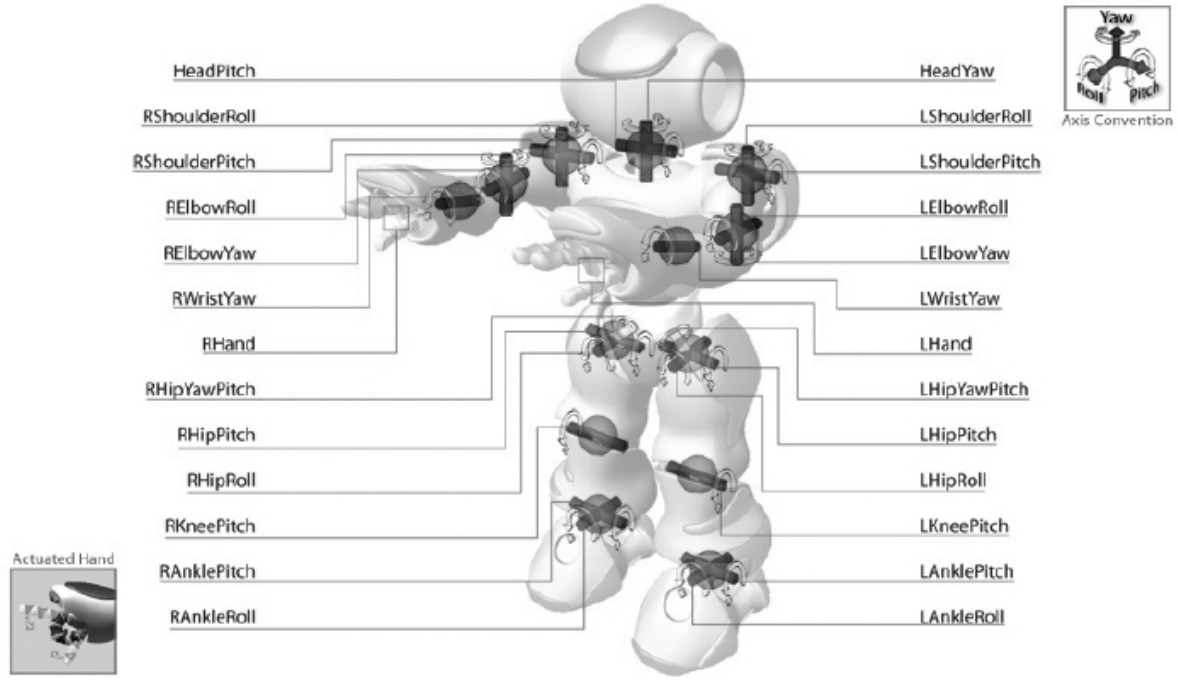


Рис. 1. Расположение сервомоторов робота

Для упрощения задачи будем считать, что только следующие 10 сервомоторов участвуют в выполнении шага: *RHipPitch, RHipRoll, RKneePitch, RAnklePitch, RAnkleRoll, LHipPitch, LHipRoll, LKneePitch, LAnklePitch, LAnkleRoll* (все они управляют ногами робота).

Кроме того, на значения углов сервомоторов накладываются ограничения, приведенные в табл. 1. Они будут использованы в качестве граничных значений на этапе применения генетического алгоритма для получения функции управления соответствующими сервомоторами робота.

Таблица 1. Ограничения на значения углов сервомоторов

| Название сервомотора | Минимальное значение, рад | Максимальное значение, рад |
|---|---|---|
| *LKneePitch* | 0.0 | 2.2689 |
| *RKneePitch* | 0.0 | 2.2689 |
| *LAnkleRoll* | -0.7854 | 0.4363 |
| *RAnkeRoll* | -0.4363 | 0.7854 |
| *LHipRoll* | -0.4363 | 0.7854 |
| *RHipRoll* | -0.7854 | 0.4363 |
| *LAnklePitch* | -1.309 | 0.7854 |
| *RAnklePitch* | -1.309 | 0.7854 |
| *LHipPitch* | -1.7453 | 0.4363 |
| *RHipPitch* | -1.7453 | 0.4363 |

# 2. Генетический алгоритм

## 2.1. Общие понятия

Генетическим алгоритмом называется последовательность из пяти операций:

- формирование начального поколения;
- тестирование популяции;
- скрещивание (кроссовер);
- мутация;
- формирование следующего поколения.

Начальное поколение составляется из фиксированного числа случайно сгенерированных особей.

Процесс тестирования заключается в вычислении для каждой особи значения фитнесс-функции – оценочной функции, которая зависит от текущей задачи и определяет приспособленность особи.

Операция скрещивания необходима для генерации нового поколения и выглядит так: на вход подается две особи, из которых по определенному алгоритму формируется две новых особи-потомка. Выбор наиболее эффективного алгоритма скрещивания (оператора кроссовера) определяется путем проведения серии экспериментов.

Мутация предназначена для внесения разнообразия в поколения. Это делается для того, чтобы особи не были слишком похожи одна на другую. Наконец, новое поколение формируется из старого и из лучших особей, получившихся в результате операций скрещивания и мутации.

Эти этапы повторяются до того момента (критерий останова), пока не будет получена (выращена) особь с достаточно высоким уровнем приспособленности или пока не станет ясно, что дальнейший рост значения фитнесс-функции невозможен.

## 2.2. Использованные методы

Отдельная особь генетического алгоритма состоит из десяти частей, соответствующих десяти сервомоторам (пять для левой ноги робота и пять для правой), каждая из которых представляет собой последовательность значений, принимаемых сервомотором последовательно в течение шага.

Если разбить время (затрачиваемое роботом на один шаг) на несколько промежутков и задать на границах значения каждого сервомотора, то остальные значения на самом промежутке могут быть аппроксимированы линейной функцией $y = ax + b$. Коэффициенты могут быть легко вычислены по общеизвестным формулам. Таким образом, управление каждым сервомотором задается числовым вектором из значений на границах промежутков, а управление каждой особью генетического алгоритма – десятью такими векторами. Задача состоит в получении особи с таким набором чисел, которые зададут функцию управления движением робота, совершающего шаг вперед.

В качестве фитнесс-функции была выбрана числовая функция, зависящая от двух параметров:

- времени, которое прошло от начала движения по заданным значениям до падения робота (если падение произошло);
- расстояния, на которое робот продвинулся вперед.

Среда *Webots* получает эти значения в реальном времени и передает их программе контроллера – в этом и состоит процесс тестирования каждой особи.

В качестве операторов скрещивания были выбраны следующие: одноточечный, арифметический, геометрический, линейный и *blx*-кроссовер [4].

# 3. Анализ результатов

## *3.1. Полученные данные*

Эксперимент состоял в последовательном запуске алгоритма генерации и тестирования особей с разными операторами скрещивания и различными параметрами (размер поколения, частота мутаций, число промежутков, на которые разбивается время шага). Наилучшие результаты были достигнуты при использовании *blx*-кроссовера (параметр *alpha* = 0.7) и арифметического кроссовера. Размер каждого поколения был равен 100 особям, вероятность мутации – 0.3, число временных промежутков – 20. Самые приспособленные особи научились делать шаг вперед и не падать. После этого процесс генерации останавливался вручную. На графиках (рис. 2, 3) представлен рост значений фитнесс-функций в двух экспериментах с разными кроссоверами, которые проведены при данных условиях.

Эти графики были получены с помощью инструментария *GAAP* и отображают рост значения функции приспособленности в зависимости от номера поколения. Красная, синяя и зеленая кривые связаны соответственно с максимальным, средним и минимальным значениями фитнесс-функции. Эксперимент в обоих случаях был прекращен в тот момент, когда среднее значение функции практически перестало изменяться от поколения к поколению. Для генетического алгоритма это значит, что скрещивание существующих особей или их мутация почти всегда дает менее приспособленную особь – новое поколение будет практически идентично предыдущему.

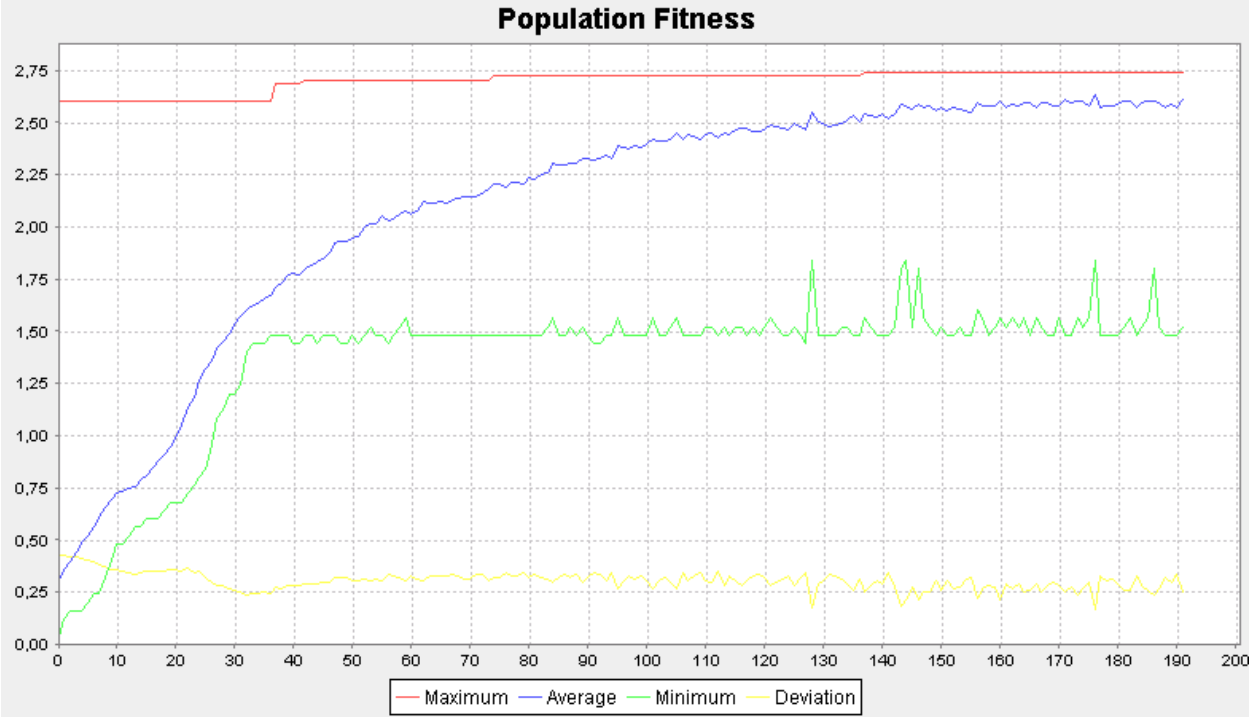


Рис.2. Арифметический кроссовер

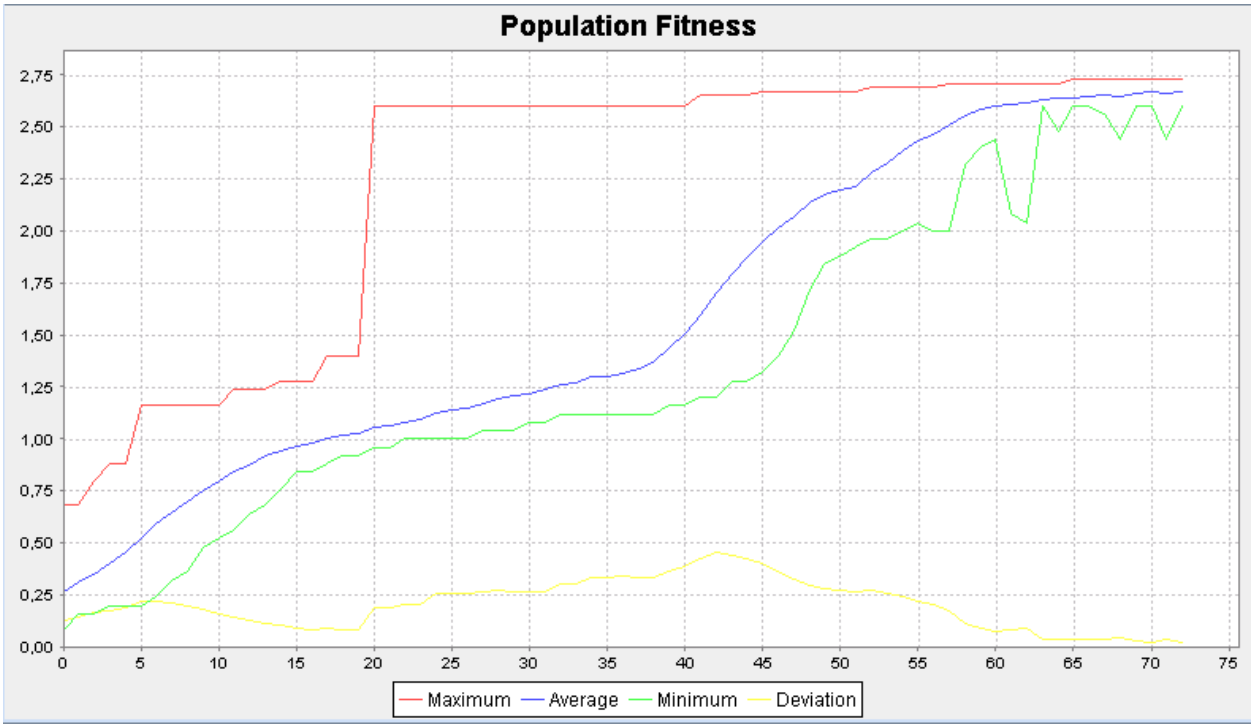


Рис.3. *blx*-кроссовер

## 3.2. Анализ данных

Результаты показали, что поставленная задача решается использованием арифметического и *blx*-кроссоверов. Прочие алгоритмы скрещивания, перечисленные в разд. 2.2, не дали удовлетворительного результата. Можно заметить, что при использовании *blx*-кроссовера результат достигается гораздо раньше (70 поколений против 140). Это связано с тем, что при арифметическом скрещивании особи-потомки оказываются более «похожи» на родителей, чем в случае *blx*-кроссовера. Таким образом, во втором случае каждое следующее поколение сильнее отличается от предыдущего, а, следовательно, шанс получить более приспособленную особь выше, чем при использовании арифметического кроссовера, при котором новые особи не слишком заметно отличаются от своих родителей. Этим и объясняется скачкообразный рост максимального значения фитнесс-функции на графике *blx*-кроссовера. Рассмотренный недостаток разнообразия арифметического скрещивания можно компенсировать увеличением размера поколения (однако в этом случае увеличивается время тестирования).

Необходимо заметить, что начальная аппроксимация зависимости работы сервомотора от времени линейными функциями могла быть заменена, например, квадратичными или кубическими сплайнами или гармоническими функциями вида $y = a*sin(w*t) + b*cos(w*t)$ [1, 5]. При использовании подхода, изложенного в работе [1], движения робота являются более плавными, чем при применении подхода, примененного в данной работе, поскольку гармонические функции не имеют перегибов, как кусочно-линейные. Однако наличие большего числа коэффициентов, которыми определяется функция и которые участвуют в генетическом алгоритме, затрудняет анализ влияния выбора одного или другого оператора скрещивания или мутации на итог эксперимента. Поэтому выбор более эффективного способа аппроксимации функции работы сервомоторов не является очевидным.

# Заключение

Результаты данной работы продемонстрировали возможность использования генетических алгоритмов в задачах моделирования движения прямоходящих роботов.

Предложенный подход с использованием генетического алгоритма можно улучшить путем выбора иного метода представления каждой особи (разд.3.2) или применением других операторов скрещивания и/или мутации.

# Источники

1. *Чеботарева Ю. К.* Применение генетических алгоритмов для генерации функций, описывающих движение, на примере шага вперед человекоподобного робота / Сборник статей третьей Всероссийской научной конференции «Нечеткие системы и мягкие вычисления», том II. Волгоград: Волгоград. гос. техн. университет. 2009, с.79–88.
2. http://www.cyberbotics.com/
3. http://code.google.com/p/gaap/
4. *Паклин Н. А.* Непрерывные генетические алгоритмы
http://www.basegroup.ru/library/optimization/real_coded_ga/
5. *Inada H., Ishii K.* A Bipedal Walk Using Central Pattern Generation, Brain Science and Engineering, Kyushu Institute of Technology, 2004.

# Приложение

Ниже приведены исходные коды генетического алгоритма, которые были использованы вместе с инструментальным средством *GAAP*. Исходный код *GAAP* можно найти на сайте [3].

## SplineTwo.java

```
package org.gaap.examples.webots.commons;

public class SplineTwo {
    private static final double EPS = 0.00001;
    private double a;
    private double b;
    public double x1;
    public double x2;

    public SplineTwo(double a, double b, double x1, double x2) {
        this.a = a;
        this.b = b;
        this.x1 = x1;
        this.x2 = x2;
    }

    public void setX1(double x1) {
        this.x1 = x1;
    }

    public void setX2(double x2) {
        this.x2 = x2;
    }

    public SplineTwo() {
    }

    public double getA() {
        return a;
    }

    public double getB() {
        return b;
    }

    public void setA(double a) {
        this.a = a;
    }

    public void setB(double b) {
        this.b = b;
    }

    public double getX1() {
```

```
        return x1;
    }

    public double getX2() {
        return x2;
    }

    public double getValue(double x) {
        return a * x + b;
    }


    public boolean isBetween(double value) {
        return (x1 <= value + EPS && value - EPS <= x2);
    }

    public void calc(double x1, double x2, double x1Value,
double x2Value) {
        this.x1 = x1;
        this.x2 = x2;
        setA(calcA(x1, x2, x1Value, x2Value));
        setB(calcB(x1, x2, x1Value, x2Value));
    }

    public static double calcB(double x1, double x2, double
x1Value, double x2Value) {
        return x2Value - (x2Value - x1Value)*x2/(x2 - x1);
    }

    public static double calcA(double x1, double x2, double
x1Value, double x2Value) {
        return (x2Value - x1Value)/(x2 - x1);
    }
}
```

**SplineFunctionTwo.java**

```
package org.gaap.examples.webots.commons;

public class SplineFunctionTwo implements Function {

    public SplineTwo spline[];

    public SplineFunctionTwo() {
        spline = new SplineTwo[Constants.SPLINE_COUNT];
    }

    public double getValue(double x) {
        int j = 0;
        while (!(spline[j].x1 <= x && x < spline[j].x2) && (j +
1 != spline.length)) {
            j++;
        }
```

```
        return spline[j].getValue(x);
    }
}
```

## Motion.java

```
package org.gaap.examples.webots.commons;

import java.io.PrintStream;
import java.util.*;

public class Motion {
    public SplineFunctionTwo[] servos;

    public Motion() {
        servos = new SplineFunctionTwo[Constants.SERVOS_COUNT];
        for (int i = 0; i < servos.length; i++) {
            servos[i] = new SplineFunctionTwo();
        }
    }

    public double getServoValue(int servo, long time) {
        return servos[servo].getValue(((double) time) / 1000);
    }

    public void save(PrintStream out) {
        long time = 0;
        int pos = 1;

        out.print("#WEBOTS_MOTION,V1.0,");
        for (int i=0; i < SoftServos.ACTIVE_SERVOS.length; i++)
{
            out.print("," +
SoftServos.ACTIVE_SERVOS[i].getName());
        }
        for (int i=0; i<SoftServos.INACTIVE_SERVOS.length; i++)
{
            out.print("," +
SoftServos.INACTIVE_SERVOS[i].getName());
        }
        out.println();
        while (time <= Constants.MOTION_TIME) {
            out.print(Constants.TIME_FORMAT.format(new
Date(time)));
            out.print(",Pose" + pos);

            for (int servo = 0; servo < Constants.SERVOS_COUNT;
servo++) {
                Formatter fmt = new Formatter(Locale.ENGLISH);
                out.print("," + fmt.format("%1.4f",
getServoValue(servo, time)));
            }
```

```
            for (int i = 0; i <
SoftServos.INACTIVE_SERVOS.length; i++) {
                Formatter fmt = new Formatter(Locale.ENGLISH);
                out.print("," + fmt.format("%1.4f",
SoftServos.INACTIVE_SERVOS[i].getValue()));

            }
            out.println();

            pos++;
            time += 40;
        }
    }
}
```

**BlendCrossover.java**

```
package org.gaap.examples.webots.chikishev;

import org.gaap.ga.impl.operators.CrossoverAdapter;
import org.gaap.utils.Utils;
import org.gaap.examples.webots.commons.SplineTwo;
import org.gaap.examples.webots.commons.SplineFunctionTwo;
import org.gaap.examples.webots.commons.Motion;
import org.gaap.examples.webots.commons.Constants;
import org.gaap.examples.webots.mandrikov.MotionMutation;

import java.util.Random;

public class BlendCrossover extends CrossoverAdapter<Motion> {
    MotionMutation m = new MotionMutation();

    @Override
    public Motion[] crossover(Motion mother, Motion father) {
        Motion child1 = new Motion();
        Motion child2 = new Motion();

        Random random = Utils.getRandom();
        double m = random.nextDouble();



        for (int i = 0; i < Constants.SERVOS_COUNT; i++) {
            child1.servos[i] = new SplineFunctionTwo();
            child2.servos[i] = new SplineFunctionTwo();


            for (int j = 0; j < Constants.SPLINE_COUNT; j++) {
                SplineTwo spline1 = new SplineTwo();
                SplineTwo spline2 = new SplineTwo();

                double x1 = mother.servos[i].spline[j].x1;
                double x2 = mother.servos[i].spline[j].x2;
```

```
                spline1.calc(
                        x1,
                        x2,
                        (j == 0 ?
mother.servos[i].spline[j].getValue(x1) :
child1.servos[i].spline[j - 1].getValue(x1)),

generateNew(mother.servos[i].spline[j].getValue(x2),
father.servos[i].spline[j].getValue(x2), m)
                );
                spline2.calc(
                        x1,
                        x2,
                        (j == 0 ?
mother.servos[i].spline[j].getValue(x1) :
child2.servos[i].spline[j - 1].getValue(x1)),

generateNew(father.servos[i].spline[j].getValue(x2),
mother.servos[i].spline[j].getValue(x2), m)
                );

                child1.servos[i].spline[j] = spline1;
                child2.servos[i].spline[j] = spline2;
            }
        }

        return new Motion[]{child1, child2};
    }

    private double generateNew(double value1, double value2,
double m) {
        double al = 0.7;
        double min = Math.min(value1, value2);
        double max = Math.max(value1, value2);
        double dif = max - min;
        return (max + 2 * dif * al - min) * m + min - dif * al;
    }
}
```

## MotionMutation.java

```
package org.gaap.examples.webots.chikishev;

import org.gaap.ga.impl.operators.MutationAdapter;
import org.gaap.utils.Utils;
import org.gaap.examples.webots.commons.SplineTwo;
import org.gaap.examples.webots.commons.SplineFunctionTwo;
import org.gaap.examples.webots.commons.Motion;
import org.gaap.examples.webots.commons.Constants;

import java.util.Random;
```

```
public class MotionMutation extends MutationAdapter<Motion> {
    private static final double MUTATION_P = 0.2;

    @Override
    public Motion mutate(Motion ind) {
        Motion temp = new Motion();

        Random random = Utils.getRandom();

        for (int i = 0; i < Constants.SERVOS_COUNT; i++) {
            temp.servos[i] = new SplineFunctionTwo();
            for (int j = 0; j < Constants.SPLINE_COUNT; j++) {
                SplineTwo spline = ind.servos[i].spline[j];
                SplineTwo newSpline = new SplineTwo();
                newSpline.setA(spline.getA());
                newSpline.setB(spline.getB());
                newSpline.setX1(spline.getX1());
                newSpline.setX2(spline.getX2());
                temp.servos[i].spline[j] = newSpline;

            }
        }
        for (int aa = 0; aa < 10; aa++){
        int i = random.nextInt(Constants.SERVOS_COUNT);
        int j = random.nextInt(Constants.SPLINE_COUNT);
        SplineTwo spline = temp.servos[i].spline[j];

        double x1 = spline.getX1();
        double x2 = spline.getX2();
        double x1Value = spline.getValue(x1) + (0.5 -
random.nextDouble())/3;
        double x2Value = spline.getValue(x2) + (0.5 -
random.nextDouble())/3;
        spline.calc(x1, x2, x1Value, x2Value);

        }
        return temp;
    }
}
```

## Main.java

```
package org.gaap.examples.webots.chikishev;

import org.apache.log4j.Logger;
import org.gaap.examples.webots.commons.*;
import org.gaap.ga.Ga;
import org.gaap.ga.Offspring;
import org.gaap.ga.Operator;
import org.gaap.ga.Selector;
import org.gaap.ga.impl.*;
import org.gaap.ga.impl.selectors.RouletteSelector;
import org.gaap.ga.utils.LeveledListener;
```

```
import org.gaap.utils.Utils;

import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {
    private static String outputPath = "C:\\Program
Files\\Webots\\projects\\contests\\nao_robocup\\controllers\\ga_
motions\\";

    private static final Logger LOG =
Logger.getLogger(Main.class);

    public static void main(String[] args) {

        Ga ga = initGa();

        ga.addListener(new LeveledListener<Ga>() {
            @Override
            public void fired(Ga ga) {
                DefaultGa g = (DefaultGa) ga;
                DefaultPopulation oldPopulation =
((DefaultPopulation) ga.getPopulation());
               /* if (oldPopulation.getAge() == 50) {
                    List<DefaultIndividual> oldP =
oldPopulation.getIndividuals();
                    Collections.sort(oldP,
DefaultIndividualComparator.getInstance());
                    Motion[] start = new
Motion[Constants.POPULATION_SIZE];
                    for (int i = 0; i < start.length *
Constants.BIG_MUTATION; i++) {
                        start[i] =
createIndividual(Constants.SPLINE_COUNT);
                    }
                    int count = (int) (start.length *
Constants.BIG_MUTATION);
                    int j = start.length - 1;
                    for (int i = count; i < start.length; i++) {
                        start[i] = oldP.get(j).getChromosome();
                        j--;
                    }
                    Offspring offspring = new Offspring(start);

g.setPopulation(g.getBreeder().breed(offspring, null));
                } */
            }

            @Override
            public int getLevel() {
```

```
                return Ga.LEVEL_MODIFY;
            }
        });

        ga.addListener(new LeveledListener<Ga>() {
            public void fired(Ga ga) {
                DefaultPopulation population =
((DefaultPopulation) ga.getPopulation());
                double fitness = population.getMaximumFitness();

                int age = population.getAge();
                LOG.info("Age="+age+", Max fitness=" + fitness);

                if (age % 10 == 0) {
                    try {
                        FileOutputStream out = new
FileOutputStream(outputPath + age + ".motion");
                        PrintStream p = new PrintStream(out);
                        ((Motion)
population.getBestIndividual().getChromosome()).save(p);
                        p.flush();
                        p.close();
                    } catch (Exception e) {
                        LOG.error(e.getMessage(), e);
                    }
                }
            }

            public int getLevel() {
                return Ga.LEVEL_READ;
            }
        });

        RunGa.run(ga);
    }

    private static Ga initGa() {
        DefaultGa ga = new DefaultGa();

        List<Selector> selectors = new ArrayList<Selector>();
        selectors.add(new RouletteSelector(0.7));
        selectors.add(new RouletteSelector(0.3));
        ga.setSelectors(selectors);

        List<Operator> operators = new ArrayList<Operator>();
        operators.add(new BlendCrossover());
        operators.add(new MotionMutation());
        ga.setOperators(operators);

        DefaultBreeder breeder = new DefaultBreeder(new
MotionTester(), 0.1);

        Motion[] start = new Motion[Constants.POPULATION_SIZE];
```

```
        for (int i = 0; i < start.length; i++) {
            start[i] = createIndividual(Constants.SPLINE_COUNT);
        }

        Offspring offspring = new Offspring(start);
        ga.setPopulation(breeder.breed(offspring, null));
        ga.setBreeder(breeder);

        return ga;
    }

    public static Motion createIndividual(int splineCount) {
        Motion result = new Motion();

        for (int num = 0; num < Constants.SERVOS_COUNT; num++) {
            ServoInfo servoInfo =
org.gaap.examples.webots.chebotareva.
SoftServos.ACTIVE_SERVOS[num];

            double x1Value;
            double x2Value = 0;

            double min = servoInfo.getMin();
            double max = servoInfo.getMax();
            double value = servoInfo.getValue();

            result.servos[num].spline = new
SplineTwo[splineCount];

            for (int i = 0; i < splineCount; i++) {
                double x1 = i * ((double) Constants.MOTION_TIME)
/ splineCount / Constants.SECOND;
                double x2 = (i + 1) * ((double)
Constants.MOTION_TIME) / splineCount / Constants.SECOND;

                if (i == 0) {
                    x1Value = value;
                    x2Value = min +
Utils.getRandom().nextDouble() * (max - min);

                } else {
                    x1Value = x2Value;

                    if (i == splineCount - 1) {
                        x2Value = value;

                    } else {
                        x2Value = min +
Utils.getRandom().nextDouble() * (max - min);

                    }
                }
```

```
                SplineTwo spline = new SplineTwo(0,
servoInfo.getValue(), x1, x2);
                spline.calc(x1, x2, x1Value, x2Value);

                double splineValue=spline.getValue((x1 + x2)/2);
                result.servos[num].spline[i] = spline;
            }
        }

        return result;
    }
}
```