VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Pengju Gong

Automatic PID Tuning Based on Genetic Algorithm for Botnia Soccer Robots

Technology and Communication

2012

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

FOREWORD

I would like to thank all the people who helped me and inspired me during my study.

At the beginning, I would like to give my honest thanks to my thesis's supervisor, Mr. Yang Liu. He not only instructs me the academic knowledge, but also offers me a lot of precious opportunities. He gave me the opportunity to study in Vaasa University of Applied Sciences and to be a member of Botnia Soccer Robot Team. Those opportunities help me to grow from everyman to an embedded system engineer.

Secondly, I would like to thank associate professor Jun Shu, who is an associate professor in Hubei University of Technology. He is fully skillful and patient, even the very basic questions, he also explained to me meticulous.

I am indebted as well to all the members of our Botnia robot teams past and present. Dong Liu, Jiangtao Zhang, Bing Feng, Yuan Gao, I learned not only skills but also how to be a real engineer from you.

In the end, thanks to all the staff of Vaasa University of Applied Sciences, thanks for all the helps from you.

Here are my deepest thanks again for all of you I mentioned above.

Vaasa, 5/7/2012

Gong Pengju

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Pengju Gong
Title	Automatic PID Tuning Based on Genetic Algorithm for Botnia Soccer Robots
Year	2012
Language	English
Pages	61 + 1 Appendices
Name of Supervisor	Yang Liu

Current PID parameters of botnia soccer robot are experiential predefined. Those PID parameters are fixed on different field surface, which make the PID controller cannot perform very well. To get appropriate PID parameters based on the specific field, the genetic algorithm was used to tune the PID parameters automatically.

In this thesis, a genetic algorithm program used to tune botnia soccer robot PID parameters has been designed. Meanwhile, this design has been simulated with Matlab and implemented with C language in ARM platform. Furthermore, an improved PID controller was designed and simulated to overcome current PID controller's disadvantages.

The simulation results are satisfactory, and the testing results are acceptable for practical environment. Several problems were found in the testing and the performance of this program can be improved in the future.

Keyword Genetic Algorithm, PID, Robotics

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF APPENDICES

LIST OF ABBREVIATION

GA : Genetic Algorithm
SGA : Simple Genetic Algorithm
PID: Proportional–Integral–Derivative
SR6 : 6^{th} generation soccer robot
ARM: Advanced RISC Machine
FPGA: Field-Programmable Gate Array

1 INTRODUCTION

1.1 Background

The RoboCup competition is one of most famous robot competitions in the world. It uses the robots as footballers to simulate the real football game and requires every team is able to “steadily, rapidly, accurately” control their soccer robots. Because only the robots are controlled precisely, they have capability to complete a series of complex movements and team cooperation. Such as robot quick startup, fast steering, precise passing, accurate navigation. Otherwise all the data calculated by strategy server would be meaningless, for instance, strategy server has figured out precise pass route and passes time, but the robot cannot arrive at the predetermined coordinates, or cannot arrive on time, this means we will lose possession of the ball.

For now, most RoboCup teams are using PID controller as the robot motor controller, which means appropriate PID parameters need to be found to achieve precise control of the robots, but on different field surface, the appropriate PID parameters are different. Calculating corresponding PID parameter separately and manually setting those parameters to robots is very laborious or even impossible 60 PID parameters need to be set for one filed (every robot has four motors which need twelve PID parameters and every team has five robots).The ideal situation is that robots are able to automatically calculate and adjust their PID parameters according to specific field conditions.

Automatic PID tuning is significative not only for RoboCup competition but also for real life. Normally, mobile robots will work in different environments, which require robots to automatically adjust their PID parameters (if they are using PID controller) according to the different environments, otherwise, the mobile robot has no practicality.

Precise motor control is so important that Botnia Soccer Robot team keeps thinking about how to get better PID parameters and how to automatically tune the PID parameters of the robot. Traditional PID calculations requires the accurate model-

ing of the robot, but detailed modeling lead to a rapid increase in computation while too rough modeling is not reliable. The main advantages of the genetic algorithm are an unnecessary robot modeling to calculate the PID parameters and its fully proven global optimization capability in the decades of research. Although robot motor control is a real-time system, the calculating of PID parameters does not need to be real-time. Once the PID parameters are calculated in certain circumstances, it can be kept using until the environment is changed, for example, in RoboCup competition, we can let robots calculate their PID parameters according to competitive field before the competition, and then use this set of parameters in the competition.

1.1 Objective

The purpose of this thesis was to design a genetic algorithm program that is suitable for botnia soccer robot. This genetic algorithm program is used to tuning PID parameters of botnia soccer robot to obtain a better performance motor controller. The requirements of this program are as following:

- Good capability of global optimization
- Time consuming is within one hour, and should be as short as possible.

In order to achieve those requirements, below should be notice:

- A set of appropriate genetic operators
- A set of appropriate genetic parameters

2 BOTNIA SOCCER ROBOT

2.1 RoboCup

RoboCup is an international scientific initiative with the goal to advance the state of the art of intelligent robots. When established in 1997, the original mission was to field a team of robots capable of winning against the human soccer World Cup champions by 2050. While that mission remains, RoboCup has since expanded into other relevant application domains based on the needs of modern society /1/.

2.2 Botnia Soccer Robot Team

Botnia soccer robot team is one of the most competitive teams in the RoboCup competition, until now, Botnia team has designed six generations soccer robot and has been in the RoboCup world championship since 2006 and ranked top 10 in Small-Size League. Botnia team was competed with top universities in the world such as Harvard, MIT, Carnegie Mellon, Georgia Tech, etc. and beat some of them.

2.3 Botnia Soccer Robot Overview

The introduced robot is the 6th generation robot, so called "SR6". **Figure** shows a picture of SR6.

SR6 robot consists of the electronic system, mechanical component, motor component and battery component.

Motor component contains four wheels and one dribble motor. The wheels are using single omni directional wheel structure. This structure can make robot to move in 360 degree, and also make the control easier and more accurate.

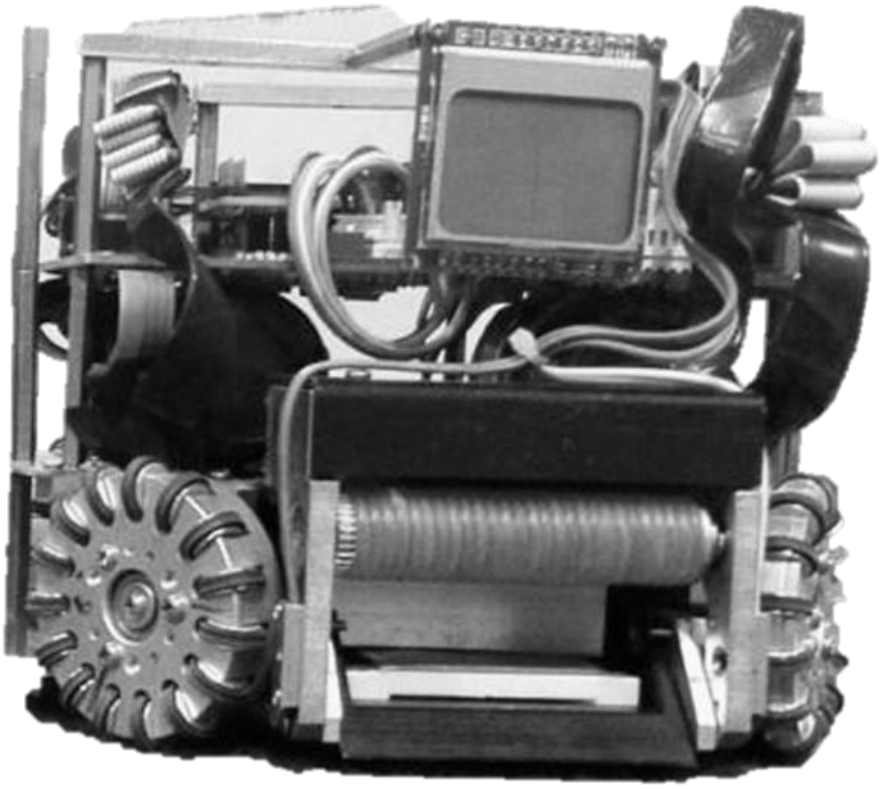

Figure . SR6 soccer robot

2.4 Robot Motors Layout

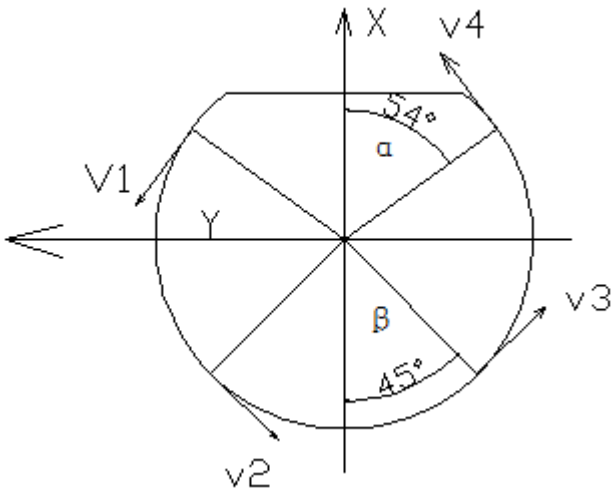

Figure . SR6 soccer robot motors layout /2/

There are three robot velocity parameters are received from strategy server, v_r, v_x and v_y, which respectively represent the rotation velocity and x, y translational velocity based on robot coordinate system. The robot coordinate system regards the robot center as origin, the robot kicking direction as the X-axis positive direction, and the left as the Y-axis positive direction. In **Figure** , v_1、 v_2、 v_3、 v_4 represent respectively linear velocity of four motors based on the robot coordinate system, and the direction of the arrow shows the rotation positive direction. α is the half of front wheels angle and β is the half of back wheel angle.

From above motors layout information, the following formulas can be obtained /2/:

$$v1 = -v_x \sin\alpha + v_y \cos\alpha + v_r \qquad ()$$

$$v2 = -v_x \sin\beta - v_y \cos\beta + v_r \qquad ()$$

$$v3 = +v_x \sin\beta - v_y \cos\beta + v_r \qquad ()$$

$$v4 = +v_x \sin\alpha + v_y \cos\alpha + v_r \qquad ()$$

Convert above formulas to matrix:

$$\begin{bmatrix} v1 \\ v2 \\ v3 \\ v4 \end{bmatrix} = \begin{bmatrix} -\sin\alpha & \cos\alpha & 1 \\ -\sin\beta & -\cos\beta & 1 \\ \sin\beta & -\cos\beta & 1 \\ \sin\alpha & \cos\alpha & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_r \end{bmatrix} \qquad ()$$

If we need calculate Vx and Vy from V1, V2, V3 and V4, the following formulas can be used.

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} -\frac{1}{4\sin\alpha} & -\frac{1}{4\sin\beta} & \frac{1}{4\sin\beta} & \frac{1}{4\sin\alpha} \\ \frac{1}{(\cos\alpha+\cos\beta)} & -\frac{1}{(\cos\alpha+\cos\beta)} & -\frac{1}{(\cos\alpha+\cos\beta)} & \frac{1}{(\cos\alpha+\cos\beta)} \end{bmatrix} \begin{bmatrix} v1 \\ v2 \\ v3 \\ v4 \end{bmatrix}$$

()

2.5 Robot Electronic System

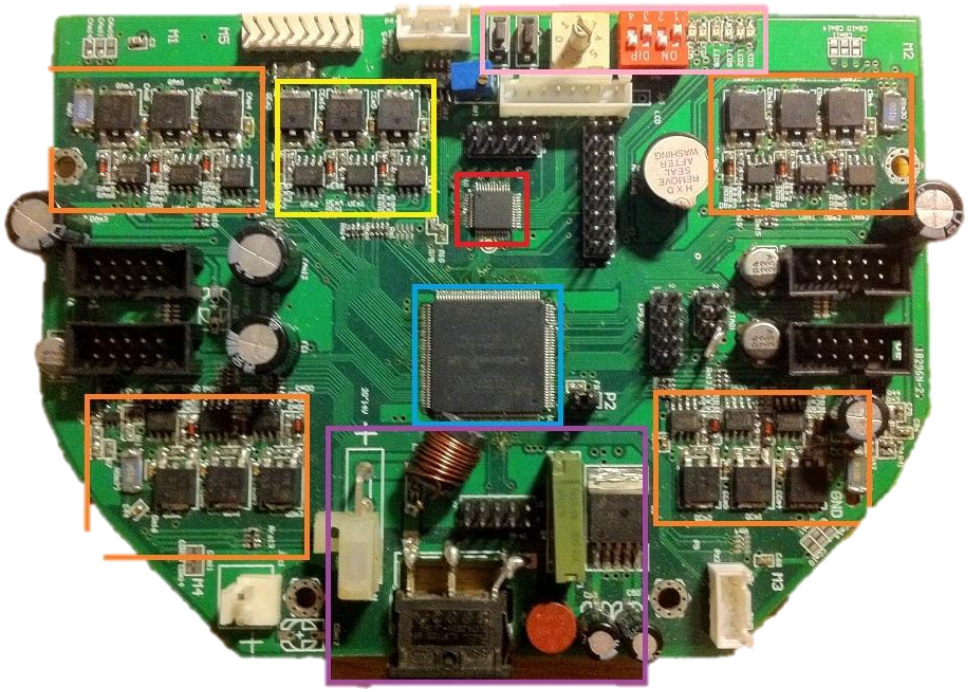

Figure . SR6 motherboard overview

Figure shows the motherboard of SR6, its main components and features are listed below:

Red Frame: The ARM (STM32) chip is the CPU of robot, which is responsible for receiving commands from strategy server and control FPGA according to those commands through SPI bus.

Blue Frame: The FPGA (cyclone III) is responsible for controlling robot sensors and motors depended on commands from ARM chip. The FPGA contains 3-loop feedback PID (position, velocity and current) controller, current protection etc. functions.

Orange Frames: Four MOSFET driver modules, which are in charge of driving the robot.

Yellow frame: The dribble motor driver module that is responsible for dribbling.

Purple Frame: The power module. SR6 is powered by a four core battery.

Pink Frame: Mode selection panel, used to select different mode of robot.

Embedded system also contains LCD display, wireless module etc.

3 PID CONTROLLER

3.1 Overview

The PID controller is the one of most popular feedback controller in the world. It contains three types of terms, the proportional term, integral term and derivative term. The aim of PID controller is to continuous decrease the difference between expected value and real value. **Figure** shows PID block diagram in following:

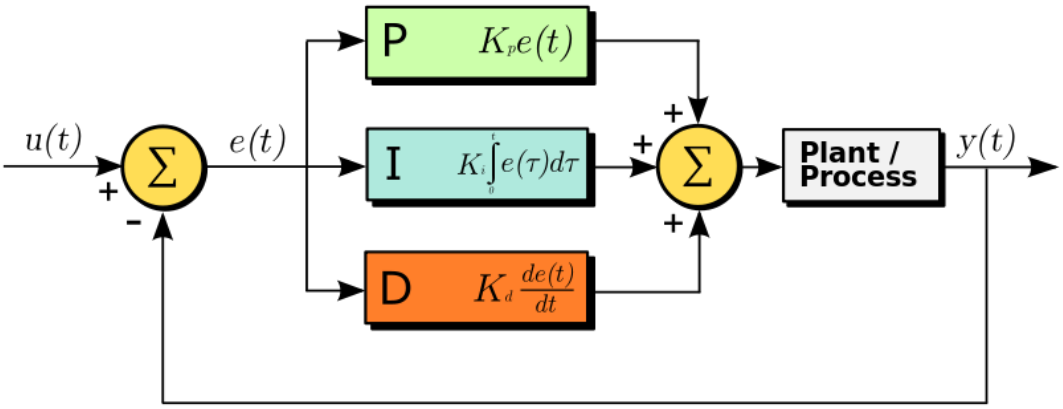

Figure . Block diagram of PID controller /3/

$u(t)$ is an expected value, $y(t)$ is the actual output value of the system, $e(t)$ is the error value, which means the difference between expected value and actual value, and $c(t)$ is the control value, which can be got by below formulas :

$$e(t) = u(t) - y(t) \qquad ()$$

$$c(t) = K_p[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d \frac{de(t)}{dt}] \qquad ()$$

k_p is the proportional factor

T_i is the integral time, $k_p * \frac{1}{T_i}$ is the integral factor

T_d is the derivative time, $k_p * T_d$ is the derivative factor

3.2 Proportional

The proportional term expression is $K_p * e(t)$

The role of proportional term is to respond to the error immediately. Once the errors appear, proportional term wills immediately take action to decrease the error. The strength of the proportional control depends on the proportional factor K_p, the greater proportional factor, the stronger control action, and then the faster transition process, but if the proportional control is too stronger, the system will be more prone to oscillatory and destabilized.

3.3 Integral

The integral term expression is $K_p * \frac{1}{T_i} \int_0^t e(t)dt$

From the mathematical expression of the integral term can know, the integral term will keep increasing until the error = 0, therefore, integral term is used to eliminate steady state error.

Although integral term can eliminate the steady state error, it will reduce the response speed of the system. Increasing T_i will slow down the PID process but can reduce the overshoot, and improve system stability. Decreasing T_i will make the integral action stronger, then the system may produce oscillations, but adjusting period will be shorter.

3.4 Derivative

The derivative term expression is $K_p * T_d \frac{de(t)}{dt}$

The actual control system is always required to accelerate the adjusting process. When error happened, it not only needs to make an immediate response (the effect of proportional term), but also need to predict the error according to the trend of error. In order to achieve this target, derivative term is used.

The derivative term effect is determined by the derivative time constant, Td, the greater Td, the stronger inhibition of error changing. Derivative term has an obviously great effect on system stability.

The effect of the derivative term is to prevent the changing of error. It controls the system based on the trend of changing (rate of changing). If error changing be faster, the output of the derivative controller will increase, and then, the error can be corrected before it becomes larger. Derivative term helps the system to reduce the overshoot, overcome the oscillation, and make the system stabilized.

But the derivative term is very sensitive to noise existed in systems. Generally, do not have derivative term, or filtering the input signal first if you want to use derivative term in the system.

3.5 Botnia Robot PID Controller

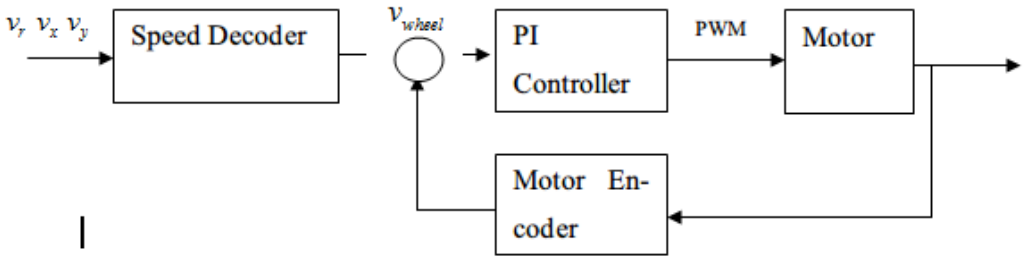

Figure . Botnia robot PID controller

Figure shows the block diagram of botnia robot PID controller. The derivative term was not used because it is easily affected by noise, and unfortunately, robot system is full of kinds of noise, for example, the motors will generate substantive noise. In this way, the parameters number can be simplify from 12 to 8 in each

robot, reducing the complexity of the optimized parameters. The control formula is listed in following:

$$c(k) = K_p * e_k + K_i \sum_{j=0}^{k} e_j + K_d(e_k - e_{k-1}) \quad ()$$

Where

$$K_i = K_p * \frac{T}{T_i}$$

$$K_d = K_p * \frac{T_d}{T}$$

The disadvantage of this algorithm is $K_i \sum_{j=0}^{k} e_j$ term will slow down speed adjusting. For example, when robots need quick startup, fast changing direction or stop immediately, this term will reduce the performance of robots.

Another disadvantage is the output of this algorithm is absolute position of the motor. If the calculation fails, the output $c(k)$ will significantly change, this will cause a substantial position change of motor, which is not allowed in RoboCup competition.

3.6 Improved Botnia Robot PID Controller

To solve above problems, those formula can be evolved as following:

$$c(k-1) = K_p[e_{k-1} + \frac{T}{T_i}\sum_{j=0}^{k-1} e_j + T_d \frac{e_{k-1} - e_{k-2}}{T}] \quad ()$$

$$\Delta c(k) = c(k) - c(k-1) => K_p[e_k - e_{k-1} + \frac{T}{T_i} e_k + T_d \frac{e_k - 2e_{k-1} + e_{k-2}}{T}]$$

$$=> K_p[1 + \frac{T}{T_i} + \frac{T_d}{T}]e_k - K_p(1 + \frac{2T_d}{T})e_{k-1} + K_p \frac{T_d}{T} e_{k-2}$$

$$=> Ae_k - Be_{k-1} + Ce_{k-2} \quad ()$$

Where

$$A = K_p[1 + \frac{T}{T_i} + \frac{T_d}{T}] = K_p + K_i + K_d$$

$$B = K_p(1 + \frac{2T_d}{T}) = K_p + 2K_d$$

$$C = K_p \frac{T_d}{T} = K_d$$

And

$$c(k) = c(k-1) + \Delta c(k) \quad ()$$

For PI controller, following formula can be got:

$$c(k) = c(k-1) + (K_p + K_i)e_k - K_p e_{k-1} \qquad ()$$

The $K_i \sum_{j=0}^{k} e_j$ term has been removed in formula 13, the delay problem are solved. When relative position of motor is required, then just need formula 11 to calculate $\Delta c(k)$, the absolution position problem can be solved.

This improved PI controller has been simulated in Matlab, but not be implemented yet, this will be a part of future work.

The following code is the Matlab simulation code of improved PID controller.

```
function [ VWheelCtrlOut,ek] = Robo_Pid_Main( VWheel-
Hope,VWheelReal,VWheelCtrlOut,ek1 )
global KpKi;

for i=1:1:4
    A = KpKi(i,1) + KpKi(i,2);  % A = kp+ki+kd
    B = KpKi(i,1);              % B = kp+2*kd
    %C = 0 ;                   % C = Kd  Kd=0 -> C =0
    ek=VWheelHope(i)-VWheelReal(i);
    VWheelCtrlOut(i)=VWheelCtrlOut(i)+A*ek-
B*ek1(i); %A*ek+B*ek1+C*ek2
end
end
```

4 GENETIC ALGORITHM

4.1 Overview

Genetic Algorithm is one of heuristic random search methods that inspired from Darwin's evolution theory. It was invented by Professor J. Holland of the United States in 1975 /4/.The basic idea of genetic algorithm is “naturally select, fitter survive” . It is a kind of mathematical simulation of real natural evolutionary process, and it is often applied as an approach to solve global optimization problems.

The advantage of genetic algorithm is that genetic algorithm itself do not need know anything about the problem, it needs only evaluate each chromosome generated by the algorithm, encodes the solution of the problem into a chromosome, and select the chromosomes based on fitness, so that the good adaptability chromosomes have more chance to reproduce offspring.

Normally, genetic algorithm begins with a group of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are used to form new solutions (offspring) are selected according to their fitness - the more suitable, the more chances they have to reproduce.

A typical genetic algorithm requires:

- A genetic representation method of all the possible solutions
- A appropriate fitness evaluation function to evaluate the fitness of all solutions
- A set of appropriate genetic operator chosen according to the project-specified situation

The genetic algorithm does not have certain procedure, which means the procedure of genetic algorithm can be changed according to different situation. This is a

challenge of genetic algorithm, the genetic operators and parameters have to be selected according to the specific project and there do not have a clearly criterion can tell us how to select genetic operators and parameters.

And unfortunately, the simple genetic algorithm (SGA) invented by Professor J. Hollan is not convergent, so elitism is often used. That means, at least one best solution is copied without changes to the new population, in this way; the best solution can survive to the end of algorithm.

Nowadays, the genetic algorithm has been widely used in combination optimization, machine learning, signal processing, adaptive control and artificial intelligence. It is a key technology in modern intelligent computing.

Figure shows the flow chart of genetic algorithm used in this article.

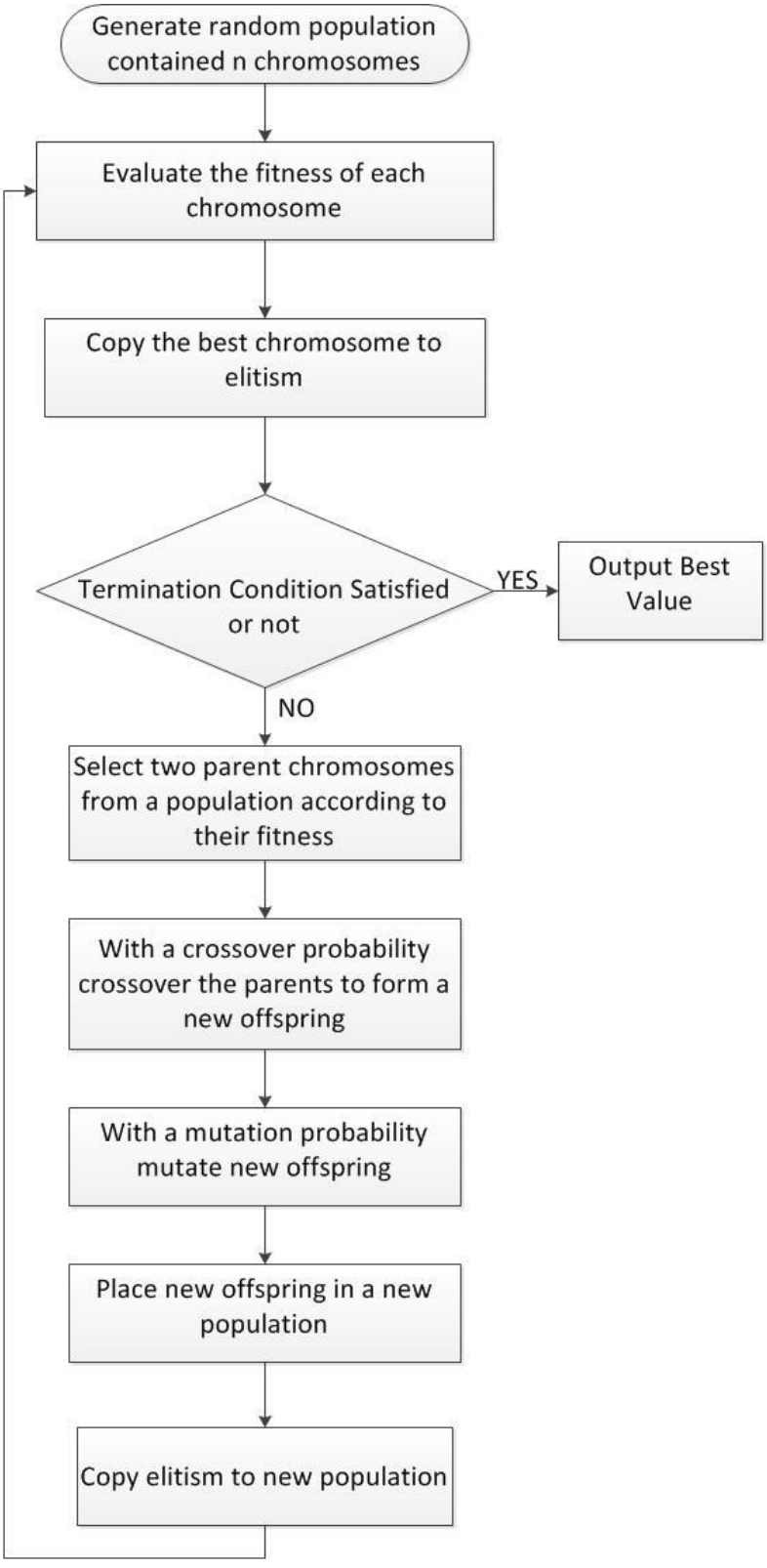

Figure . Flow chart of genetic algorithm

4.2 Terminology

Chromosome

The chromosome is also called individuals, which are used to represent the solutions in state space.

Population

The population is a group of individuals. Numbers of individuals make up the population, and the number of individuals in the population is called the population size.

Fitness

In the theory of evolution, fitness indicates a particular individual adaptive capacity to the environment. The higher fitness means the greater probability to be selected.

Elite

It is the best individual in each generation, or say, the individual that owns highest fitness in current population.

4.3 Genetic Representation

Genetic representation is the first challenge when the genetic algorithm is applied and it is a critical step in the genetic algorithm design. The representation method decides which operators will be used and how to design the fitness evaluation function, largely determines the performance of genetic algorithm.

So far people have many different representation methods. In general, these encoding methods can be divided into three categories: binary encoding method, floating encoding method and symbol encoding method. Normally, the binary encoding is the first choice because following reasons:

- Decoding operation is faster than others
- Crossover and mutation operation is easy to achieve with binary encoding method

The binary encoding method is used in botnia robot genetic algorithm and all the discussion in this article is based on the binary encoding method.

The general binary encoding means convert the possible solutions to a string of binary. Each solution contains at least one variable, for instance, in PID controller, normally, each solution contains three variables, Kp, Ki and Kd. Assume the length of solution is fixed 16 bit, 0-4 bit can be assigned to Kd, 5-9 bit to Ki, 10-15 bit to Kp. If the individual is 12332 (001100 00001 01100), then Kd is 01100, Ki is 00001, Kp is 001100, but integer is not enough to present solution, decimal value is needed to present more accurate numbers, such as Ki, the value of Ki should from 0 to 1, not just 0 or 1, to solve this problem, Ki can be divided by 31(the maximum value of Ki is 11111, which means 31 in decimal form), then the value from 0 to 1 can be achieve. If Ki is required from 0 to 2, let Ki*2/31, then the value of Ki will be from 0 to 2.

The botnia robot encoding method will be introduced in details in simulation and implementation chapter.

4.4 Fitness Evaluation

The genetic fitness function that also called object function is used to calculate the fitness of individuals. The fitness function is always problem dependent, for different project, the reasonable fitness function need to be selected depend on analysis the specify problem. If this is designed wrongly, the algorithm will either converge on an inappropriate solution, or will have difficulty converging at all /5/.

In other words, the fitness function plays a very important role in genetic algorithm to obtain the best solutions within a large search space. Good fitness functions will help genetic algorithm to explore the search space more effectively and efficiently. Bad fitness functions, on the other hand, can easily make genetic algorithm get trapped in a local optimum solution and lose the discovery power /6/.

4.5 Genetic Operator

Selection, Crossover and Mutation are called Genetic operators, they are the vital components of genetic algorithm, which can decide the performance of genetic algorithm.

But for specific situation, it is possible to not only use those three operators, but also use other operators like regrouping to improve the genetic algorithm global optimization capability.

4.5.1 Selection

Selection operator is used to choose individuals from the current generation's population according to the fitness of individuals.

The local optimum problem has to be noticed in selection operator. Selection operator usually devotes a lot of effort to maintaining the diversity of the population to prevent premature convergence. Generally, the harder the problem, the more such local optimum there are /7/. Do not only select the best individuals because selecting other individuals in addition to the best ones maintains the diversity of the population, the solutions are spread further out in the search space, and if a

part of the population is stuck in a local optimum, a different part of the population can still make progress.

Several often used selecting methods are listed in the following:

Roulette - A selection operator in which the chance of a individual getting selected is proportional to its fitness. This is where the concept of survival of the fittest comes into play.

Tournament - A selection operator which uses roulette selection N times to produce a tournament subset of individuals. The best individuals in this subset are then chosen as the selected individuals. This method of selection applies addition selective pressure over plain roulette selection.

Random - A selection operator randomly selects an individual from current population.

4.5.2 Crossover

The central role of biological evolution is the reorganization of the biological genetic, so the key operation of the genetic algorithm is the crossover operator. Crossover operator will randomly choose points every time to splice parent genes.

There are several ways to achieve the crossover:

Single-Point Crossover

Select a certain single point of both parent individuals. Binary string from beginning of individual to the crossover point is copied from one parent; the rest is copied from another parent.

For example,

Before crossover:

Father 00000|01110000000010000

Mother 11100|00000111111000101

After crossover:

Offspring 00000|00000111111000101

Offspring 11100|01110000000010000

Two-Point Crossover

Two crossover points are selected, binary string from beginning of individual to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent.

For example,

Before crossover:

Father 01 |0010| 11

Mother 11 |0111| 01

After crossover:

Offspring 11 |0010| 01

Offspring 01 |0111| 11

4.5.3 Mutation

Mutation operator will alter one or more gene values in an individual from its initial state. It is an aiding method to generate new individuals, which determines the local optimization ability of genetic algorithm, while maintaining the diversity of the population.

The basic bit mutation operator refers to the individual encoded strings randomly assigned one or a few genes to mutation operator. If those genes is 0, then the mutation operator turn it to 1, on other hand, if the original gene value is 1, the mutation operator turns it to 0.

For example,

Before Mutation:

00000111000 0 000010000

After Mutation:

00000111000 1 000010000

4.6 Elitism

Only selecting genes from the generated offspring to form a new population may lose a lot of information in the previous generation populations. In other words, when crossover operator and mutation operator is used to produce a new generation, it is very likely to destroy the optimal solution. In each new generation, at least one best individual are exactly copied to the new generation. In this way, the optimal solution can survive to the end of the genetic algorithm.

Another reason why the elitism is used is that the studies found the simple genetic algorithm is not convergent, but the genetic algorithm with elitism will converge for sure. But genetic algorithm with elitism is possible to convergence in local optimal solution; thoughtful genetic parameters have to be selected to avoid this problem.

4.7 Terminal Condition

The genetic loop is repeated until terminal conditions are reached. The reasonable terminal conditions can shorten the time of the genetic algorithm. In general, the terminal conditions are following terms:

- Reach the maximal generation
- Found the satisfied individual
- Could not find a better individual for many generations

4.8 Genetic Parameters

Like the genetic operators, genetic parameters also have vital effect of the performance of genetic algorithm. The unreasonable parameters will lead to the failure of genetic algorithm.

The extreme important parameters are maximum generation, population size, crossover probability and mutation probability.

4.8.1 Maximum Generation

Maximum generation is the maximum number of generation, for instance, if the maximum generation is 50, which mean the genetic loop will be repeated for 50 times. Too small maximum generation will reduce the global optimization capability and too big maximum generation will increase the time consuming of genetic algorithm. The genetic algorithm will converge after numbers of generations, and then the maximum generation can be chosen as convergent generation number.

4.8.2 Population Size

The population size is the number of individuals in the population.

It is difficult to find the optimal solution when the population size is too small, on another hand, too large population will slow down the genetic algorithm, and study was found that the bigger population size is helpless to optimize the results. Different problems may have their appropriate population size.

Normally, population size is usually 30-160. Some studies suggest that choose the population size depends on the coding method.

4.8.3 Crossover Probability

Probability of crossover operator used in genetic algorithm, Crossover probability generally take a value from 0.6 to 0.95, if this value is small, it will be very difficult to find a better individual, but if this value is big, it will be easy to destroy the high fitness individual.

4.8.4 Mutation Probability

Mutation probability, generally take a value from 0.01 to 0.03.

If the mutation probability is too small, it is difficult to produce a new gene structure, thus reduce the global optimization capability.

Mutation probability cannot be more than 0.5; otherwise genetic algorithm will become a complete random search algorithm.

5 SIMULATION AND IMPLEMENTATION

The genetic algorithm for botnia soccer robot is explained in this chapter, the algorithm is introduced first, then the Matlab code in and C code will be revealed to make the introduction more clear and understandable.

The genetic algorithm used in botnia soccer robot is based on SGA, which has been introduced in above chapter; the reason why advanced genetic algorithm did not be used is the CPU of robot is ARM chip, which is not as powerful as PC's CPU. The calculation have to be reduced to make it is possible to run genetic algorithm in robot, in other words, the genetic algorithm used in robot has to be computationally efficient. But elitism that not belongs to SGA still need to be used to ensure the genetic algorithm is convergent.

5.1 The Main Function

From the main function, the main genetic loop used in botnia soccer robot can be known.

In the beginning, initialize genetic parameters. Those genetic parameters are global variable, which can make parameters tuning faster and easier. Those values are empirical values. The selection of genetic parameters will be introduced in the following subsection.

Matlab Code:

```
MAX_GENERATION = 60;
POPULATION_SIZE = 101;
PERFECT_FITNESS = 5500;
P_MUTATION= 0.002;
P_CROSSOVER = 0.80;
```

C Code:

```
#define POPULATION_SIZE 101 //POPULATION_SIZE
#define P_CROSSOVER  3435973837
//probability of crossover 3435973837/2^32=0.8
#define P_MUTATION   8589935
//probability of mutation 8589935/2^32=0.002
#define MAX_GENERATION 60
#define BEST_FITNESS   5500
```

After initialization, we start genetic algorithm process.

Step 1

Creating initial population randomly, the population has to cover the whole state space and the individuals have to keep enough diversity. It is can be seen from **Figure** that the individuals distribute in whole state space and different with each other in the initial population.

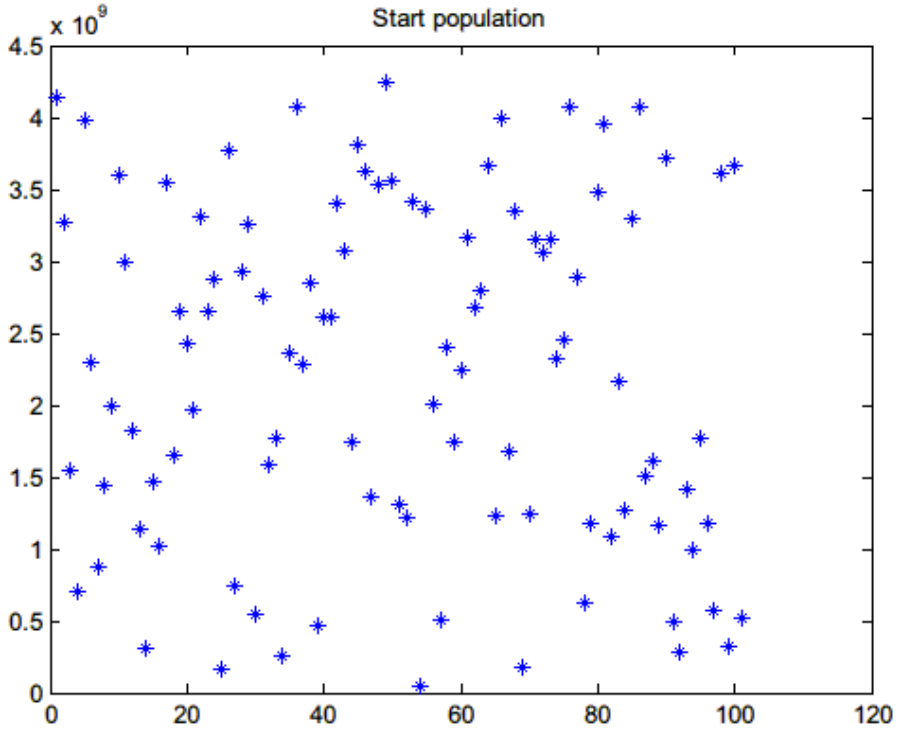

Figure . Individuals' distribution in start population

Matlab Code:

```
%random create initial population individuals range 0-2^32
population = round(rand(1,POPULATION_SIZE)*(2^32));
start_pop = population;
```

C Code:

```
      //**********create the initial population*********//
      srand(time(NULL));
      for(population_index=0;population_index<POPULATION_SIZE
;population_index++)
      {
            population[population_index]=rand();
      }
```

Step 2

Evaluate each individual. Before evaluation, individuals need to be decoded, then obtain individual's fitness by evaluate function, the details of evaluate function will be introduced in following subsection.

Matlab Code:

```
%evaluate each individual
    for i =1:1:POPULATION_SIZE
        KpKi = Robo_Ga_Decode(population(1,i));
        fitness(1,i) = Robo_Ga_Evaluate();
        fprintf(fid,'fitness=%4.3f\n',fitness(1,i));
    end
```

C Code:

```
for(population_index=0;population_index<POPULATION_SIZE;popula
tion_index++)

{Decode(population[population_index]);

//get kp,ki values

fitness[population_index] = Evaluate(execpt_wheel_v);
//kp,ki -> fitness

}
```

Step 3

Preserve the elitist, the Robo_Ga_Max function is used to get the maximum fitness from fitness array.

Matlab Code:

```
%elitist
[max_fitness, max_fitness_index]= Robo_Ga_Max(fitness(1,:));
fprintf('max_fitness=%4.3f\n',max_fitness);
fprintf(fid,'max_fitness=%4.3f\n',max_fitness);
population(1,POPULATION_SIZE)= popula-
tion(1,max_fitness_index);
```

C Code:

```
// elitist strategy elitist rate: 1.25%  elitist number: 1
max_fitness = Max(fitness);  //get max_fitness and
max_fitness_index population[0] =
population[max_fitness_index];// population[0] is the elitist
```

Step 4

Check if the ideal fitness value is got, if it is got, stop genetic process, output corresponding individual, else start population reproduction process.

Matlab Code:

```
if max_fitness>PERFECT_FITNESS
   KpKi = Robo_Ga_Decode(population(1,max_fitness_index));
   break
else
   for i=1:1:((POPULATION_SIZE-1)/2)
```

Step 5

The first step of reproduction is selection. Two parents are chosen according to its fitness. The individuals who have the higher fitness have more changes to be selected.

Matlab Code:

```
%select
f = Robo_Ga_Select(fitness);
fprintf(fid,'father index=%4.3f\n',f);
m = Robo_Ga_Select(fitness);
fprintf(fid,'mother index=%4.3f\n',m);
parent(1) = population(1,f);
fprintf(fid,'father=%4.3f\n',parent(1));
parent(2) = population(1,m);
fprintf(fid,'mother=%4.3f\n',parent(2));
```

C Code:

```
//select
f = Select(fitness);
m = Select(fitness);
parent[0] = population[f];
parent[1] = population[m];
```

Step 6

Crossover operation, Robo_Ga_Crossover function uses two parent selected by selection operator as the inputs to generate two offspring, the offspring variable is a matrix, which contains two values.

Matlab Code:

```
%crossover
offspring = Robo_Ga_Crossover(parent);
fprintf(fid,'offspring 1 =%4.3f\n',offspring(1));
fprintf(fid,'offspring 2 =%4.3f\n',offspring(2));
```

C Code:

```
//crossover
offspring[0] = *(Crossover(parent));
offspring[1] = *(Crossover(parent)+1);
```

Step 7

After crossover operation, execute mutation operation to import new gene to population.

Matlab Code:

```
%mutate
offspring(1) = Robo_Ga_Mutate(offspring(1));
offspring(2) = Robo_Ga_Mutate(offspring(2));
fprintf(fid,'offspring 1 =%4.3f\n',offspring(1));
fprintf(fid,'offspring 2 =%4.3f\n',offspring(2));
```

C Code:

```
//mutate
offspring[0] = Mutate(offspring[0]);
offspring[1] = Mutate(offspring[1]);
```

Step 8

Form the new population with offspring.

Matlab Code:

```
%reproduction
population(1,i) = offspring(1);
population(1,POPULATION_SIZE-i) = offspring(2);
```

C Code:

```
//reproduction
population[population_index] = offspring[0];
population[POPULATION_SIZE-population_index+1] = offspring[1];
```

Until now, one generation process has been finished, this process need to be repeated until the terminal conditions are reached.

When genetic terminal conditions are reached, output the final result.

Matlab Code:

```
%the end of main loop of ga
[max_fitness, max_fitness_index]= Robo_Ga_Max(fitness(1,:));
KpKi = Robo_Ga_Decode(population(1,max_fitness_index));
```

C Code:

```
//*******the end of main loop of ga******************//
Max(fitness);     //get max_fitness_index
Decode(population[max_fitness_index]); //get kp,ki values
```

In the following subsection, the corresponding functions used in above genetic loop will be introduced in details.

5.2 Genetic Representation

As mentioned in introduction of botnia soccer robot, one botnia robot has four motors, each motor requires two PID parameters, Kp and Ki, to achieve the PID control, which means, one robot need eight parameters, so eight parameters need to be encoded to one individual that length is 32 bit, thus, 4 bit for each PID parameter.

For motors of botnia robot, the lower bound of Kp and Ki is 0 and the upper bound is 2.

Following is the individual decoding function, which can give an account of encoding method.

num2str(bitget(uint32(individual),29:32)) means extract 28 to 31 bit from individual's binary string.

then divide extracted 4 bit binary string by 15, then the value from 0 to 1 can be achieve, the reason is extracted binary string is 4 bit, if num2str(bitget(uint32(individual),29:32)) = 15, divide by 15, the result will be 1, otherwise, the result will less than 1. Multiplying the result by 2.0, the value from 0 to 2 can be got.

Matlab Code:

```
function [ KpKi ] = Robo_Ga_Decode( individual )
KpKi(1,1) =
bin2dec(num2str(bitget(uint32(individual),29:32)))*2.0/15.0;%
extract 4 bit(31-28) as kp0 range 0~2 resolution 128.0/15.0 =
8.533
KpKi(1,2) =
bin2dec(num2str(bitget(uint32(individual),25:28)))*2.0/15.0;
KpKi(2,1) =
bin2dec(num2str(bitget(uint32(individual),21:24)))*2.0/15.0;
KpKi(2,2) =
bin2dec(num2str(bitget(uint32(individual),17:20)))*2.0/15.0;
KpKi(3,1) =
bin2dec(num2str(bitget(uint32(individual),13:16)))*2.0/15.0;
KpKi(3,2) =
bin2dec(num2str(bitget(uint32(individual),9:12)))*2.0/15.0;
KpKi(4,1) =
bin2dec(num2str(bitget(uint32(individual),5:8)))*2.0/15.0;
KpKi(4,2) =
bin2dec(num2str(bitget(uint32(individual),1:4)))*2.0/15.0;
end
```

C Code:

```
void Decode(u32 individual)
//chromsome decoding range: kp 0-2 ki 0-2
{
      kp[0] =  ((individual&0xF0000000)>>28)*128.0/15.0;
      ki[0] =  ((individual&0x0F000000)>>24)*128.0/15.0;
      kp[1] =  ((individual&0x00F00000)>>20)*128.0/15.0;
      ki[1] =  ((individual&0x000F0000)>>16)*128.0/15.0;
      kp[2] =  ((individual&0x0000F000)>>12)*128.0/15.0;
      ki[2] =  ((individual&0x00000F00)>>8)*128.0/15.0;
      kp[3] =  ((individual&0x000000F0)>>4)*128.0/15.0;
      ki[3] =  ((individual&0x0000000F)>>0)*128.0/15.0;
}
```

5.3 Fitness Evaluation

When an automatic control system is evaluated, there are three primary figures of merit is very important: overshoot, settling time, steady-state error. Among them, the overshoot is the system stability indicator, settling time is the system efficiency indicator and steady-state error is the system accuracy indicator, the smaller value of these three parameters, the better performance of the automatic control system.

For individual motor, the actual speed of the motor need to be read from the robot motor encoder per 0.01s. According to equation 7, the e(t) can be got. For each PID process, 250 times sampling will be achieve because after 250 times sampling, the PID process is stabilized. ∑e(t) is the sum of the value of overshoot, settling time, steady state error. Square ∑e(t) to make following calculation easier. For instance, the area of the curve in the following figure is the ∑e(t).

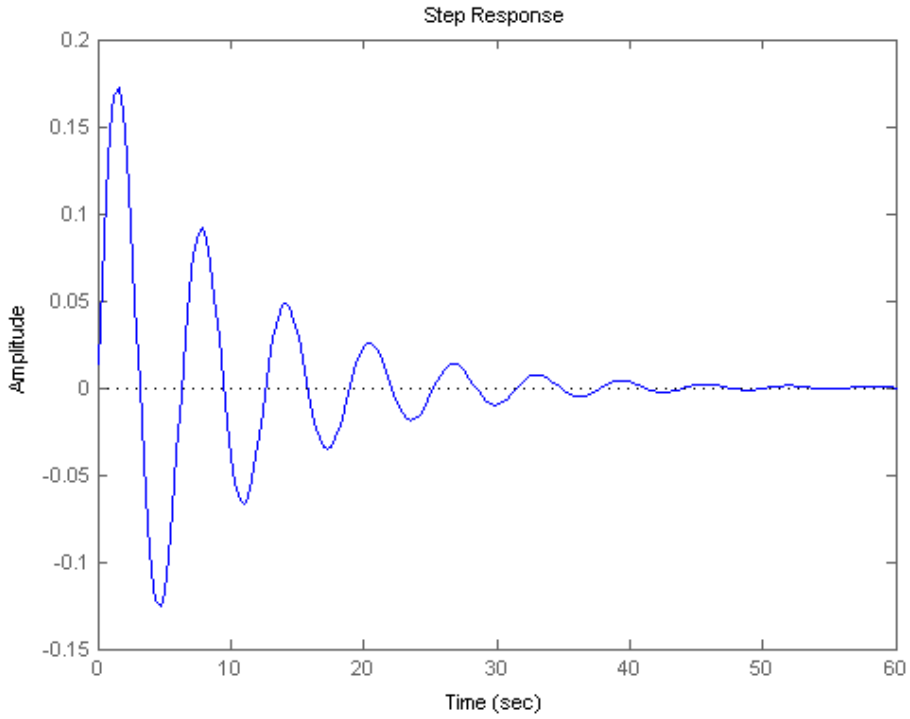

Figure . Evaluation illustration

For the entire robot, due to the interaction between the robot four wheels, even if the single motor parameters have been optimal, the entire robot may not be able to achieve the best performance. Therefore, the fitness evaluation function must evaluate four wheels together for PID tuning. The desired speed of the four motor and the robot is different, the desired motor speeds should be convert from desired robot speed by equation 6, and then calculate the difference of the actual speed and desired speed of the four motors separately, after that, accumulate four motors differences to get the ∑e(t) of robot. This ∑e(t) can reflect the control performance of robot motor control system. The smaller ∑e(t) means the better control

performance. Fitness is the reciprocal of ∑e(t), that means greater fitness reflect the controller performance is better.

Matlab Code:

```
for i=2:1:PIDIndex
[VWheelCtrlOut(i,:),ek(i,:)]
=Robo_Pid_Main(VWheelHope,VWheelFeedBack(i-
1,:),VWheelCtrlOut(i-1,:),ek(i-1,:));

VWheelReal(i,:)=Robo_WheelCtrlOutToRealV(VWheelReal(i-
1,:),VWheelCtrlOut(i,:));

VWheelFeedBack(i,1)=VWheelReal(i,1)*1;
VWheelFeedBack(i,2)=VWheelReal(i,2)*1;
VWheelFeedBack(i,3)=VWheelReal(i,3)*1;
VWheelFeedBack(i,4)=VWheelReal(i,4)*1;

VRobotReal(i,:)=Robo_WheelVToRobotV(VWheelReal(i,:));
end

   Eval=0;
   EvalE=[0,0,0,0];
   for i=1:1:PIDIndex
        e=VWheelHope-VWheelFeedBack(i,:);
        for j=1:1:4
            EvalE(j)=EvalE(j)+e(j)*e(j);
        end
   end
   Eval=Eval+(EvalE(1)+EvalE(2)+EvalE(3)+EvalE(4))/PIDIndex;
   %KpKi
   Eval=1000000/Eval;
end
```

C Code:

```
u32 Evaluate(long* execpt_wheel_v)//input kp[4] ki[4]
{
float fitness = 0.0;
u8 eva_num = 200;
//maybe after 200 times, the pid can be stable
u8 i;
float err;
for(i= 0;i<eva_num;i++)
{
u8 index;
for(index=0;index<4;index++)  //4 movement motors
{
MotorSetPidParameter(index,0 ,kp[index]);
//send kp parameter to each motor
MotorSetPidParameter(index,1 ,ki[index]);
//send ki parameter to each motor
iDelay(175);  //1750=1ms 175=0.1ms
err = execpt_wheel_v[index] - RD_MotorSpeed(index);
fitness += err*err;
}
}
fitness = fitness/eva_num;
fitness = 1000000/fitness;

return fitness;
}
```

5.4 Genetic Operator

5.4.1 Selection

Roulette algorithm is used in selection operator. Assume the fitness of every individual is represented by a pie chart and this pie chart is used as a gambling wheel. **Figure** shows this gambling wheel. In this gambling wheel, each piece represents an individual in population. The piece size is proportional to the fitness of the individual, the higher fitness, the bigger piece size in this pie chart. To select an individual, what you need to do is to throw a ball into the gambling wheels, then rotate it, after the wheel stops, checking the ball stop in which piece, then corresponding individual can be selected.

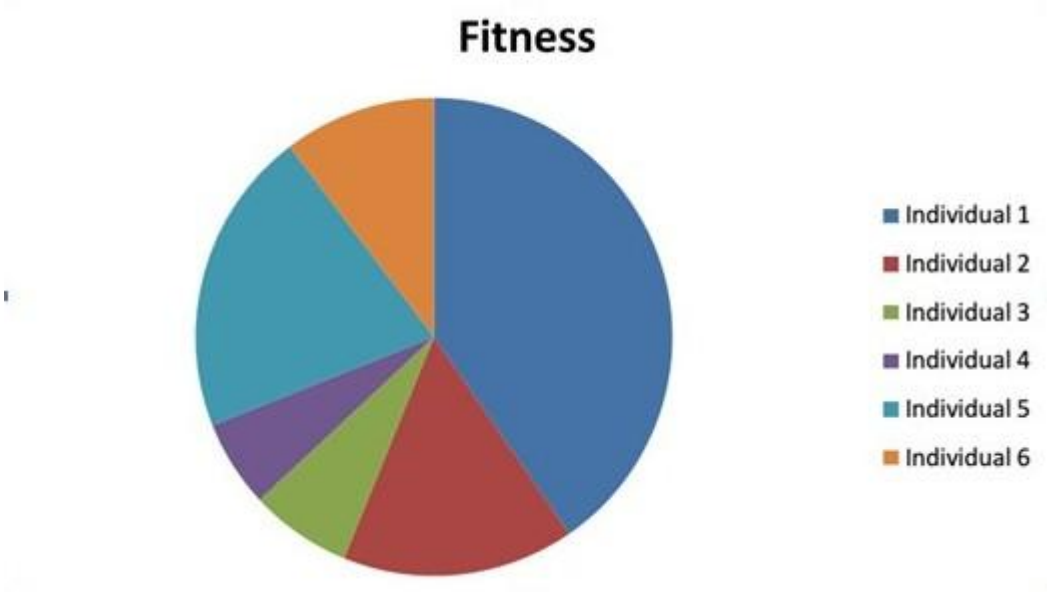

Figure . Fitness roulette

How could the roulette be implemented in programming? Randomly select a number as threshold according to the population, then start to accumulate the fitness of every individual. When the threshold is reached, output the index of corresponding individual, otherwise, keep accumulating fitness until the threshold is reached. With roulette algorithm, the fitter individuals have more chance to be selected, and the worse individuals also have chance to be selected. The worse individuals need have chance to be selected since event worst individual, there still may be good gene. If only the best individuals in the population are selected, the genetic algorithm is very likely to get stuck on a local optimum. Our purpose is to get the global optimum, once the genetic algorithm gets stuck on a local optimum, and then it is very hard to get out.

Matlab Code:

```
function [ index ] = Robo_Ga_Select( fitness )
global POPULATION_SIZE;

temp = 0;
i= 1;
threshold = 400000*rand(); %400000 general fitness is
between 2000 to 3000
while 1
    temp = temp+fitness(1,i);
    if temp > threshold
        index = i;
        break
    else
        i=i+1;
    end

    if temp < threshold && i == (POPULATION_SIZE-2)
        i = 1;
    end
end
```

C Code:

```
u8 Select(u32* fitness) //roulette wheel selection
{
      u32 filtered_fitness = 0;
      u32 temp=0;
      u8 i = 0;
      u32 threshold = 0;
      seed++;
      srand(seed);
      threshold = rand()%1500000; //threshold: 0-400000
      while(1)
      {
      filtered_fitness = *(fitness+i);
      if(filtered_fitness>8000&&filtered_fitness<20000)
      {
            temp += *(fitness+i);
//accumulate fitness values
      }
      else
            temp += 9000;
      if (temp>threshold)
      {
            return (i);
      }
      else
            i++;
      if (temp < threshold && i == (POPULATION_SIZE-1))
                  i = 0;
      }
}
```

5.4.2 Crossover

The single-point crossover is used in genetic algorithm for botnia robot.

A gene locus is randomly chosen, offspring 1 gets the genes before this locus of father and the genes after this locus of mother. Offspring 2 gets the genes before this locus of its mother and the genes after this locus of its father.

Matlab Code:

```
function [ offspring ] = Robo_Ga_Crossover(parent)
global P_CROSSOVER;
global fid;
global cro_counter;

j = 0;
while j==0
   j=rand;
end
if (rand < P_CROSSOVER)
    offspring(1) = parent(1)*j + parent(2)*(1-j);
    offspring(2) = parent(1)*(1-j) + parent(2)*j;
else
    offspring(1)= parent(1);
    offspring(2)= parent(2);
    fprintf(fid,'omg! crossover did not happan!\n');
end
end
```

C Code:

```
u32* Crossover(u32 *parent) //single-point crossover
{
	u8 i = 0;
	u32 father = 0;
	u32 mother = 0;
	u32 offspring[2];
	father = parent[0];
	mother = parent[1];
	offspring[2] = 0;
	i = 0;
	seed++;
	srand(seed);
	if(rand()<P_CROSSOVER)
	{
		i = rand()%32;    //0-32 random choose crossover
point
		offspring[0] = father&(0xFFFFFFFF<<i);
		offspring[0] |= mother&(0xFFFFFFFF>>(32-i));
		offspring[1] = mother&(0xFFFFFFFF<<i);
		offspring[1] |= father&(0xFFFFFFFF>>(32-i));
	}
	else
	{
		offspring[0] = father;
		offspring[1] = mother;
	}
	return offspring;
}
```

5.4.3 Mutation

The single-point mutation is used in genetic algorithm for botnia robot.

A gene locus in individual is randomly selected, XOR the value in this locus to generate a new individual.

Matlab Code:

```
function [ newoffspring ] = Robo_Ga_Mutate( oldoffspring )
global P_MUTATION;
global fid;
global mut_counter;

j = 0;
while j == 0
j=round(rand()*32);

end
if (rand < P_MUTATION)
    newoffspring = xor(1,bitget(uint32(oldoffspring),j));
    fprintf(fid,'lol,Mutate!\n');
else
    newoffspring = oldoffspring;

end
```

C Code:

```
u32 Mutate(u32 offspring)     //chromosome mutate
{
      u8 i = rand()%32; //random choose mutation point



      if(rand()<P_MUTATION)
            {
                  offspring ^= (1<<i);          //0->1 or
1->0

//                if(0==(offspring&(1<<i)))
//                {
//                      offspring|=(1<<i);      //0->1
//                }
//                else offspring&=~(1<<i);      //1->0

            }

      return offspring;
}
```

5.5 Genetic Parameters

More than one hundred times testing were made to select the genetic parameters, a part of testing results are listed in following tables, which typically shows the influence each parameters on genetic algorithm.

The maximum fitness is used to be the indicator of genetic algorithm performance since the purpose of genetic algorithm is to get a greater fitness.

5.5.1 Maximum Generation

The maximum generation parameter was tested under following conditions:

```
POPULATION_SIZE = 101;
PERFECT_FITNESS = 5500;
P_MUTATION= 0.002;
P_CROSSOVER = 0.80;
```

Each maximum generation parameter was tested for three times. The 30, 40, 50 and 60 are been chosen as population size and some typical results are listed for each maximum generation parameter in the following table.

Table Maximum Generation Comparison

Maximum Generation	Maximum Fitness			
	1	2	3	Average
30	4005.810	4288.340	4154.165	4149.438
40	3911.637	4218.150	3730.095	3953.294
50	5172.546	5149.048	5064.781	5128.792
60	5117.902	5372.736	5012.553	5167.730

From **Table 1** can found that normally after 50 generations, the fitness does not increase any more, that means, the genetic algorithm for botnia robot will converge in 50 generations, so 50 is chosen as the value of maximum generation.

5.5.2 Population Size

The population size parameter was tested under following conditions:

```
MAX_GENERATION = 50;
PERFECT_FITNESS = 5500;
P_MUTATION= 0.002;
P_CROSSOVER = 0.80;
```

Each population size parameter was tested for three times. The 61, 81, 91, 101, and 121 are chosen as population size and typical results are listed for each population size in the following table.

Table Population Size Comparison

Population Size	Maximum Fitness			
	1	2	3	Average
61	3146.725	3263.189	3106.155	3172.023
81	4741.174	4607.987	4941.537	4763.566
91	5318.148	4781.858	4881.958	4993.988
101	5172.546	5149.048	5064.781	5128.792
121	4820.625	5137.821	4964.057	4974.168

Table can obtain two conclusions:

- When population size less than 91, the genetic algorithm will meet local optimum problem, the maximum fitness did not increase in following generations.
- When population size is greater than 101, the performance of genetic algorithm does not rise with the incensement of population size. More individuals will consume more time but will not improve the performance of genetic algorithm.

Through the comparison of each population size, I chose 101 as population size.

5.5.3 Crossover Probability

The population size parameter was tested under following conditions:

```
POPULATION_SIZE = 101;
MAX_GENERATION = 50;
PERFECT_FITNESS = 5500;
P_MUTATION= 0.002;
```

Table Crossover Probability Comparison

Crossover Probability	Maximum Fitness			
	1	2	3	Average
0.7	5075.598	3053.558	4932.342	4353.833
0.75	5253.152	5021.422	4323.432	4866.002
0.8	5172.546	5149.048	5064.781	5128.792

In the second testing of crossover probability = 0.7, the maximum fitness only have 3053.558, the population's fitness did not increase in the genetic process, that means the genetic algorithm fallen into local optimum in this testing, the reason is the value of crossover probability is too small lead to solutions converge very soon.

I chose 0.8 as the value of crossover probability, because in numbers of testing, this value can avoid genetic algorithm falling into the local optimum.

5.5.4 Mutation Probability

The population size parameter was tested under following conditions:

```
POPULATION_SIZE = 101;
MAX_GENERATION = 50;
PERFECT_FITNESS = 5500;
P_CROSSOVER = 0.80;
```

Table Mutation Probability Comparison

Mutation Probability	Maximum Fitness			
	1	2	3	Average
0.001	4541.758	5048.739	4454.836	4681.778
0.0015	5065.538	5264.798	5113.118	5147.818

0.002	5172.546	5149.048	5064.781	5147.818

Although the results in **Table** are not very different with each other, the 0.002 is still selected as the mutation probability to improve the global optimization ability of genetic algorithm.

5.6 Simulation Result

The population size parameter testing is under following conditions:

```
POPULATION_SIZE = 101;
MAX_GENERATION = 50;
PERFECT_FITNESS = 5500;
P_CROSSOVER = 0.80;
P_MUTATION= 0.002;
```

The below diagrams show one of best results, which get satisfied fitness in 16th generation, the summary of final result is listed in the following (those results are saved automatically by Matlab program):

max_fitness=5966.967

population (1,POPULATION_SIZE)=604801152.233

Kp Ki Results

0.533, 0.000 (motor 1)

0.133, 0.133(motor 2)

0.267, 0.400(motor 3)

0.133, 0.000(motor 4)

Figure shows the average fitness in each generation. **Figure** shows best fitness in each generation. **Figure** shows four wheel's speed curve. And **Figure** shows the robot speed curve.

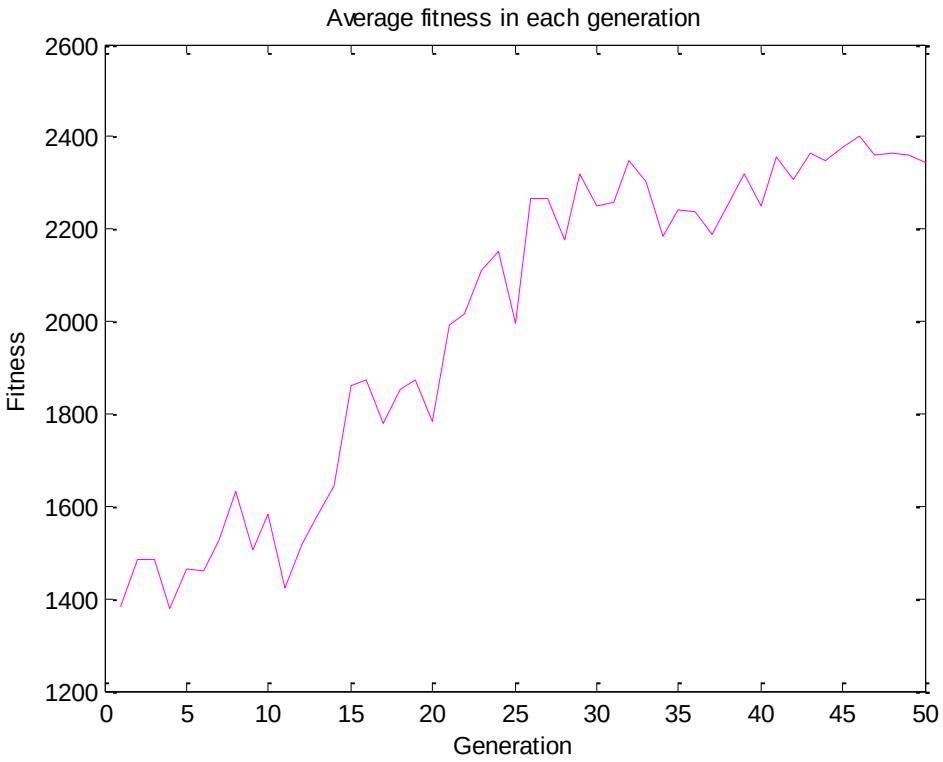

Figure . Average fitness in each generation

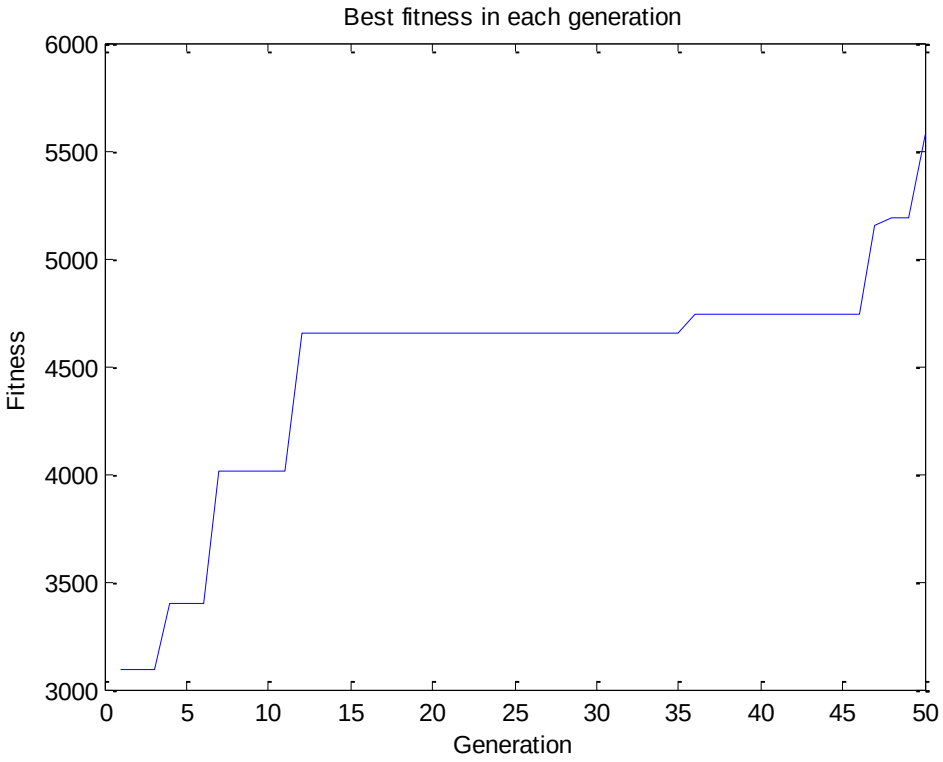

Figure . Best fitness in each generation

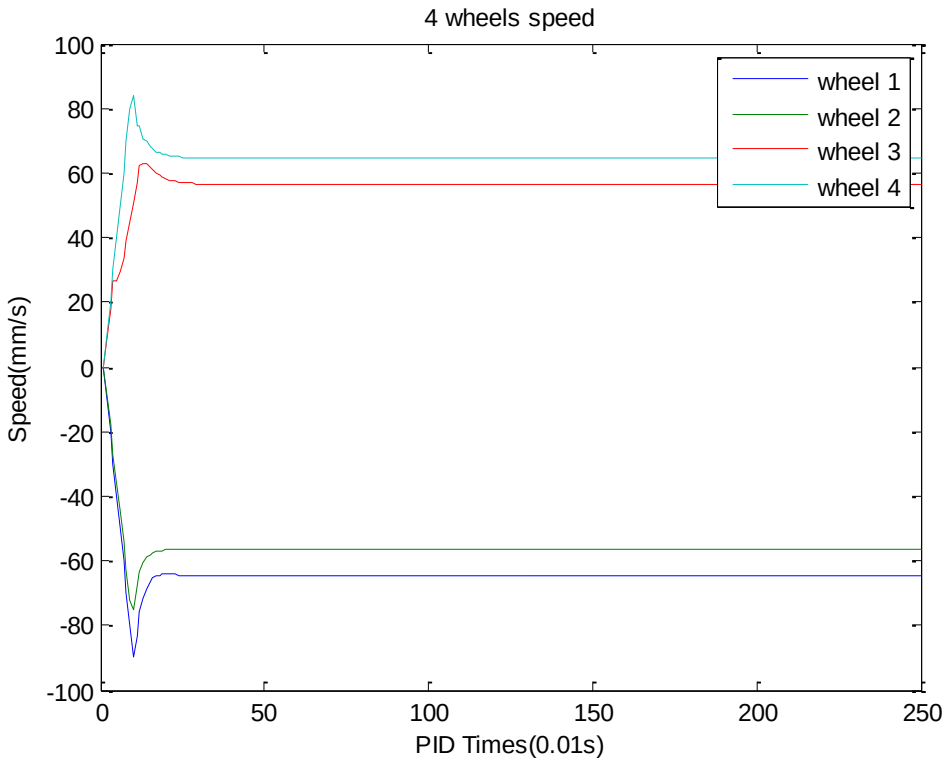

Figure . Four wheels speed

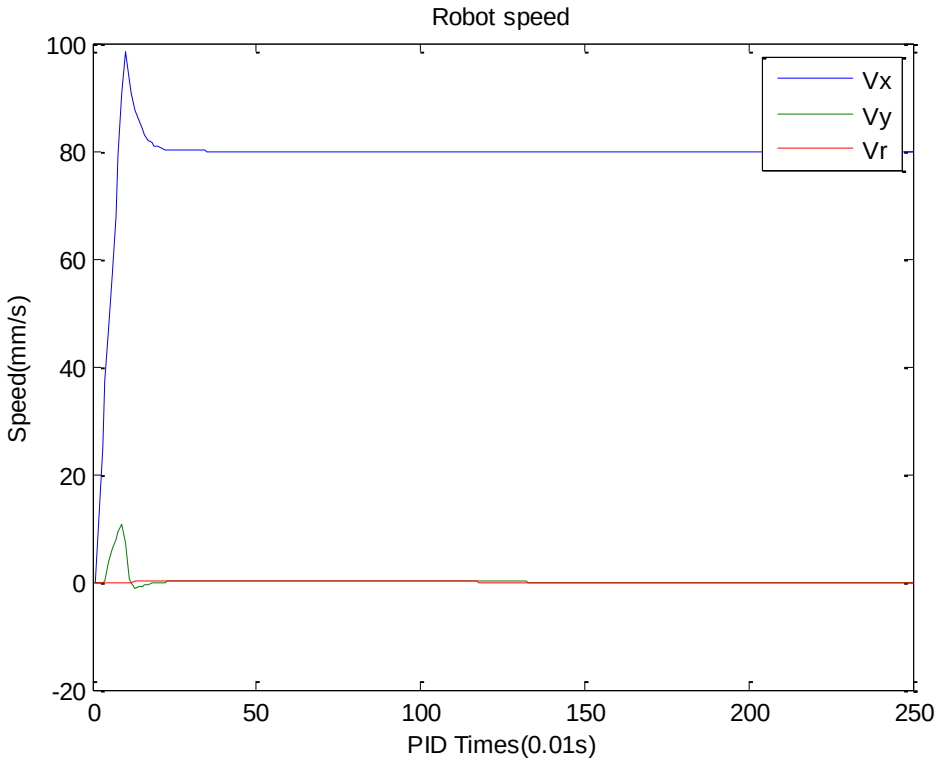

Figure . Robot speed

Figure and **Figure** shows the simulation results of the PID controller without genetic algorithm result in the below to compare with above results.

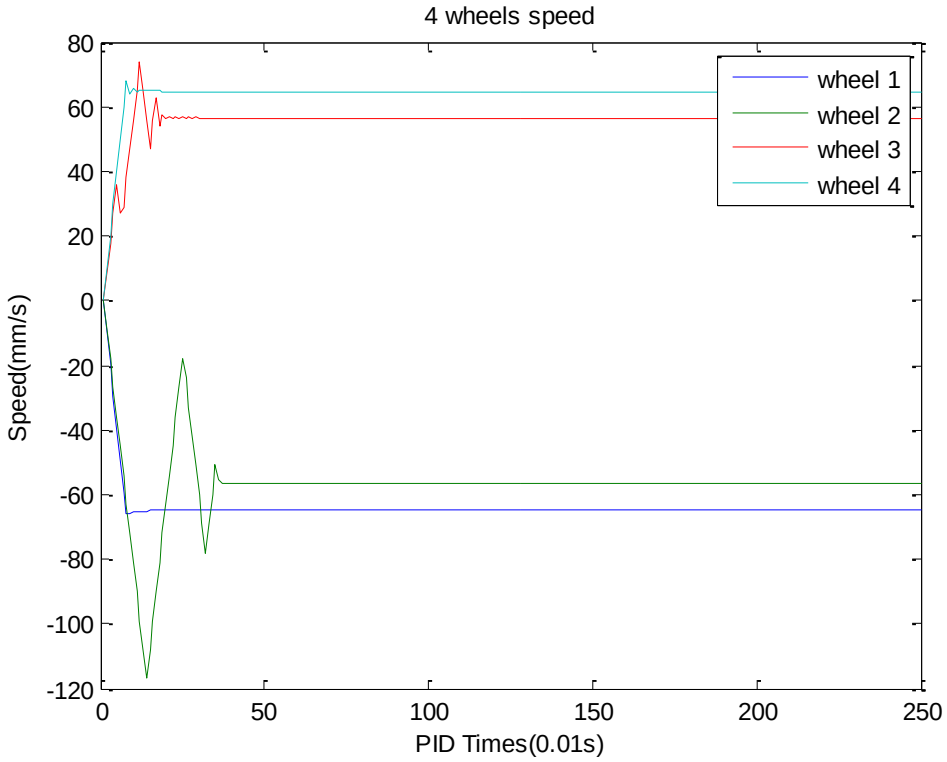

Figure . Wheels speed result of PID without GA

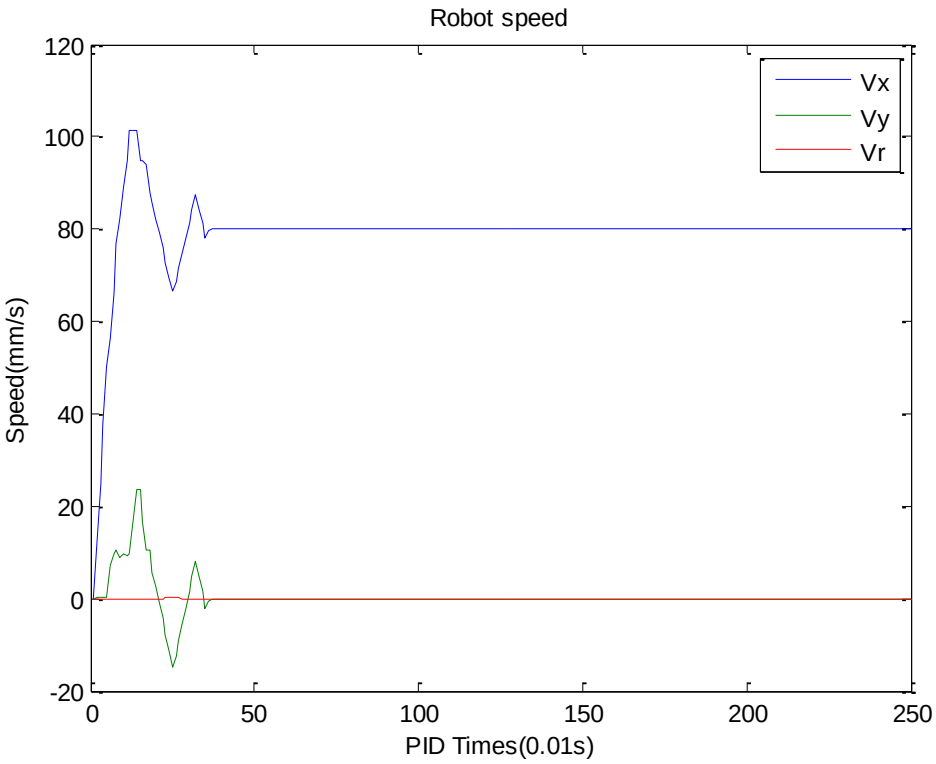

Figure . Robot speed result of PID without GA

The performance of PID controller with genetic algorithm is significantly better than the PID controller without genetic algorithm. The value of overshoot and setting time of former is obviously less than the latter. The genetic algorithm successful offers reasonable PID parameters to PID controller according to specific situation.

6 TESTING

In the practical testing, all the results can be observed through the debugger of IDE. The functions of genetic algorithm program work as predefined, but there are three problems were found, which are listed below:

- Speed of motors become uncontrolled with unfavorable PID parameters
- The evaluation process is very time consuming
- Some values in the memory of ARM are unreasonable.

The uncontrolled speed problem is that the velocity of wheels is not convergence or the overshoot of speed is unacceptable when the PID parameters are unfavorable. This problem will lead to the speed of motor become uncontrolled and motors run at breakneck speed. To solve this problem, the speed protection was set, once the speed of wheels are over the threshold, the wheels are stopped immediately. One thing should be noticed, MotorSetspeed function could not be used to stop the motors now, because the motors are out of control under the unfavorable PID parameters, in this situation, the motors can be stopped through set the kp and ki to zero.

C Code:

```
//speed protect
if(realv>100)
{
//MotorSetspeed(0,0);//no effect
MotorSetPidParameter(0,0,0);
MotorSetPidParameter(0,1,0);
MotorSetPidParameter(1,0,0);
MotorSetPidParameter(1,1,0);
MotorSetPidParameter(2,0,0);
MotorSetPidParameter(2,1,0);
MotorSetPidParameter(3,0,0);
MotorSetPidParameter(3,1,0);
break;
}
```

Although the speed limitation was set to protect motors, it takes around 2s before the speed is limited. The reason is not clear yet for now, but the solutions will be tried to solve this problem.

Another problem is the time consuming problem. The evaluation should occur only after the speed of motor is steady. Otherwise the next evaluation result was affected by pervious evaluation process. It takes around 2s to wait for the speed become steady, which means, each evaluation takes around 2s. For each generation, there are 101 evaluations, so the evaluation will be very time consuming. A solution should be found to reduce the time of evaluation. The answer is not obtained yet, but it will be a future work.

The third problem is some values in the memory of ARM are unreasonable. **Figure** show the watcher result of fitness array, the fitness array is initialized to 0, all values in the array should be 0, but the value of fitness [15] and fitness [30] is unreasonable. The reason of this problem is not very clear, might be the noise lead to this problem. Some value filters are used in program to filter those unreasonable values.

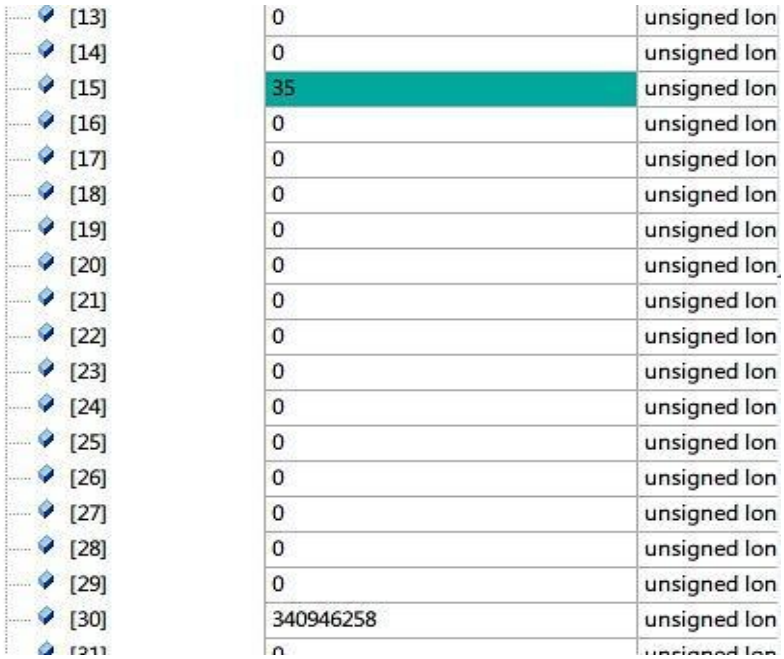

Figure . The watcher result of fitness array

7 CONCLUSION

In the thesis, a genetic algorithm program used to tune botnia soccer robot PID parameters has been designed. Meanwhile, this design has been simulated with Matlab and implemented with C language in ARM platform. Furthermore, an improved PID controller also was designed and simulated to overcome current PID controller's disadvantages.

Most of simulation tastings can obtain ideal PID parameters, which make the overshoot is within 30mm, settling time is within 0.3s, and there is no steady state error, so, the simulation results are satisfactory,

In practical testing, although it is time-consuming, it is unnecessary to tuning PID parameters very fast, so this problem can be skipped temporarily. Another problem is that the velocity of wheels is not convergence when the PID parameters are unfavorable. Although the speed limitation was set to protect motors, it takes around 2s before the speed is limited. So, the results in practical environment are not perfect but still acceptable.

In the future, the genetic algorithm program can be moved to strategy sever, because the computational capability of strategy sever is much better than botnia robot and the vision data also can be used in genetic algorithm program to aid PID tuning if the genetic algorithm is running in strategy sever. And the improved PID controller will be implemented to instead of current PID controller.

REFERENCES

/1/ RoboCup Call for Participation. Accessed1.3.2012.

http://www.ai.rug.nl/robocupathome/cfp2012.html

/2/ Luo Zhong, Li Yang, Jun Shu, 2012, unpublished. Accessed 1.3.2012.
On the Combined Automatic Four Wheel PI Parameter Setting Algorithm for Rob oCup

/3/ PID controller – Wikipedia. Accessed1.4.2012.

http://en.wikipedia.org/wiki/PID_controller

/4/ Genetic algorithm – Wikipedia. Accessed1.4.2012.

http://en.wikipedia.org/wiki/Genetic_algorithm

/5/ Fitness function – Wikipedia
Accessed13.4.2012.http://en.wikipedia.org/wiki/Fitness_(genetic_algorithm)

/6/ Harris Wu, The Effects of Fitness Functions on Genetic Programming-Based Ranking Discovery For Web Search. Accessed19.4.2012. The Effects of Fitness Functions on Genetic Programming-Based Ranking Discovery For Web Search.pdf>

/7/ Selection (genetic algorithm) – Wikipedia. Accessed1.4.2012.

http://en.wikipedia.org/wiki/Selection_(genetic_algorithm)

/8/ Jin-Sung Kim, Jin-Hwan Kim, Ji-Mo Park, Sung-Man Park, Won-Yong Choe

And Hoon Heo, Auto Tuning PID Controller based on Improved Genetic Algorithm for Reverse Osmosis Plant. World Academy of Science, Engineering and Technology. Accessed23.4.2012.

http://www.waset.org/journals/waset/v47/v47-70.pdf>

/9/ Artificial intelligence - Genetic Algorithm selection and crossover - Stack Overflow. Accessed20.4.2012.

http://stackoverflow.com/questions/8106111/genetic-algorithm-selection-and-crossover

The Embedded Software Development Environment of Botnia Robot

FPGA development IDE: Quartus II 9.0sp2 Web Edition

ARM development IDE and toolchain Information:

IDE-Version: uision V4.23.00.0

Toolchain: MDK-ARM Standard Version: 4.23

Toolchain Path: C:\Keil\ARM\BIN40

C Compiler: Armcc.Exe V4.1.0.894

Assembler: Armasm.Exe V4.1.0.894

Linker/Locator: ArmLink.Exe V4.1.0.894

Librarian: ArmAr.Exe V4.1.0.894

Hex Converter: FromElf.Exe V4.1.0.894

CPU DLL: SARMCM3.DLL V4.23

Dialog DLL: DARMSTM.DLL V1.63.0.0

Target DLL: Segger\JL2CM3.dll

Dialog DLL: TARMSTM.DLL V1.60