Implementation of Path Planning using Genetic Algorithms on Mobile Robots

Hagen Burchardt, Ralf Salomon

***Abstract*—This paper presents the implementation of a genetic algorithm based path planning on RoboCup's small-size league robots. The algorithm is adjusted to the resource constraints of micro controllers that are used in embedded environments. Because path planning on mobile robots is a continuous process, the path planning runs until the robot arrives its destination. Hereby, the path is updated to environmental changes, such as moving obstacles.**

I. Introduction

The RoboCup initiative [1] aims to have a team of humanoid robots that play soccer against the human world champion in 2050. One of the leagues is the Small-Size league. The robots of this league have a diameter of only 18 cm. Due to their size, they have limited calculation power. The robots are remotely controlled by a PC which analyzes the image that is captured by two cameras, located 4 meters above the field.

Many teams [11][12][13] use a central server to calculate the strategy and the robots behaviors. The central control PC sends only the wheel speeds to the robots. However, this implementation leaves only little room for improvement and research. Therefore research groups started to move more intelligence onto the robot. As the robots have no own vision system, they need external information. The central server sends a vectored view of the field to the robots. With this information and their internal sensors, the robots are able to navigate on the field and to fulfill simple tasks, such as moving to a destination position and kicking the ball, without further assistance of the server.

As the robots now calculate their movements, they need to be able to determine a collision free path. This problem class, which is a sub class of routing algorithms, is called path planning. Path planning algorithms find the shortest route between two points and along the way avoid obstacles.

Small-Size League robots use micro controller sized CPUs. Calculation power and memory are limited resources. Discrete path planning algorithms, such as potential fields, grid based algorithms, splines, and tangent finding [2][3][9], either need too much CPU performance or do not meet the memory constraints. Genetic algorithms are a possible solution to overcome the limitations of classical algorithms. They are able to cover a large search space and use a relatively low amount of memory and CPU resources. However, they do not always find the global optimum, which is the shortest path in this case. This paper demonstrates that genetic algorithms are also able to adapt a found solution to a continuously changing environment.

Hagen Burchardt and Ralf Salomon are with the Institute of Applied Microelectronics and Computer Engineering, College of Computer Science and Electrical Engineering, University of Rostock, 18119 Rostock-Warnemünde, Germany (phone: +49 381 498 7251; fax: +49 381 498 7252; email: {hagen.burchardt, ralf salomon}@uni-rostock.de

II. Problem Description

A. RoboCup Small-Size League Rules

In RoboCup's small-size league, two teams of five robots each play soccer on a five by four meters green carpet. An image recognition software extracts the robots' position from an image provided by two cameras that are located four meters above the floor. The robots have a maximum diameter of 18 cm and play with an orange golf ball. Nearly all teams use an omnidirectional drive, which allows the robot to move in any direction without prior rotation. According to the rules, the robots should avoid any direct contact to each other. This is especially true for the goal keeper robot, which is not allowed to be touched by any other robot [10].

B. Hardware Constraints of the Robot

Due to their small size, small-size league robots cannot be equipped with powerful CPUs. Micro controllers with a calculation speed of about 5 MIPs control all main functions of the robot. Their tasks range from wireless communication and wheel speed control to simple calculations and decisions. Usually, 16-bit micro controllers are used as they provide an optimum between calculation performance and energy consumption. Although these controllers are quite powerful for their size, they have several restrictions. The integrated memory often has a size of less then 64 KB Flash and 4 KB RAM. Nearly no micro controller has an integrated FPU[1], and therefore, all floating point operations are emulated. Due to this emulation, floating point operations on micro controllers should be avoided. The robots that are used in the experiments are equipped with a Motorola HC-12 micro controller [4], which has a clock speed of 16 MHz. Its integrated hardware modules take over several CPU intensive tasks, such as PWM[2] generation, pulse accumulation, and serial communication which frees CPU time for applications.

The robot uses a three-wheeled omnidirectional drive. Figure 1 shows a photo of the robot and its drive. Each wheel is driven by a separate motor. Wheel and motor are coupled by a drive belt. Small sub-wheels are mounted on the rim of each wheel. This system allows the robot to drive in any direction without prior rotation.

[1]Floating Point Unit, a mathematical Co-Processor
[2]pulse width modulation

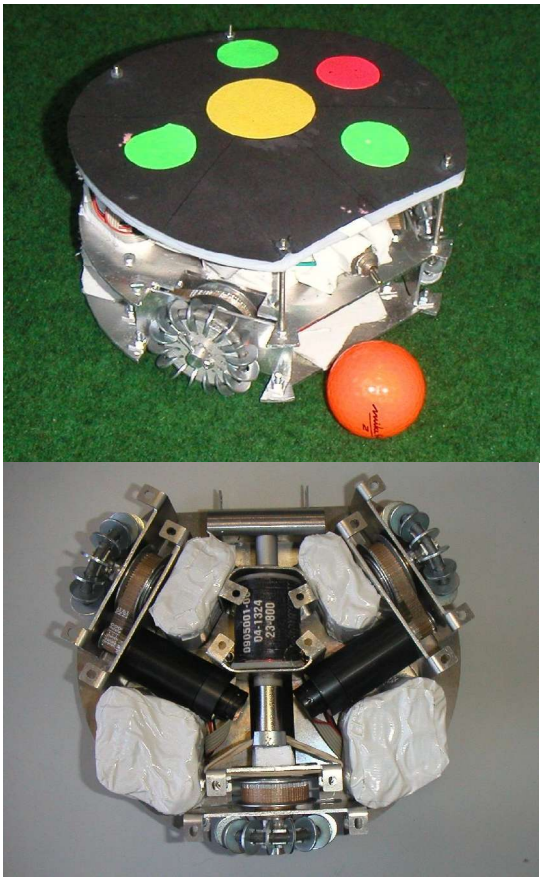


Fig. 1 Mechanics of a small-size league robot

C. Path Planning

The purpose of path planning algorithms is to find a collision free route that satisfies certain optimization parameters between two points. In dynamic environments, a found solution needs to be re-evaluated and updated to environmental changes.

In case of RoboCup, all robots on the field are obstacles. Due to the global camera view, the positions of all robots and hereby all obstacles are known by the robot.

III. Approach Description

Genetic algorithms use evolutionary methods to find an optimal solution. The solution space is formed by parameters. Possible solutions are represented as individuals of a population. Each gene of an individual represents a parameter. A complete set of genes forms an individual. A new generation is formed by selecting the best individuals from the parent generation and applying evolutionary methods like recombination and mutation. After a new generation is generated, each offspring is tested with a fitness function. From all offspring, and in case of (μ+λ)-strategy also from the parents, the μ best individuals are chosen as the parents of the next generation. μ usually denotes the number of parents while λ is the number of generated children for the next generation.

A. Gene Encoding

To apply genetic algorithms to the problem of path planning, the path needs to be encoded into genes. An individual represents a possible path. The path is stored in way points. The start and the destination point of the path are not part of an individual. As the needed number of way points is not known, it is variable. As a result, the gene length is variable.

Several approaches aim to map the coordinates of the way points into a one-dimensional representation. As an effect, the length of the gene is shorter which results in a smaller solution space. XOY as proposed in [5] is a very short encoding scheme. Here, the path is encoded as equidistant distances of the direct line between start and destination. However, this approach only works when a nearly direct route exists. Therefore limiting the solution space by mapping it into one dimension is not an option. As shown in figure 2, each way point is stored in its x and y coordinates as integer values.

Length	x_1	y_1	x_2	y_2	x_3	y_3

Fig. 2 Gene Encoding of an Individual

As the obstacles are relatively small against the size of the field and their number cannot exceed ten because there are only five robots of each team on the field. This leaves enough room for navigation, three way points between start and end position are sufficient to find a route. Therefore, the maximal number of way points is set to three.

B. Fitness Function

The fitness function is important for the algorithm's stability, because an inadequate function may lead to either stuck at local minima or oscillations around an optimum. Fitness functions are usually constructed by accumulation of weighted evaluation functions. In case of path planning, needed evaluation functions are the path length and a collision avoidance term.

When choosing the representation of the obstacles, it needs to be considered that the calculation is done on the robot. Therefore, the memory footprint is a very important factor.

There are two different approaches to encode the obstacles on the field. The field can be represented in an two-dimensional array in which cells are marked that are occupied by an obstacle. The array method allows for an easy search algorithm and does not need much calculation performance but is memory consuming. Algorithms utilizing field representation arrays can be found in [6] and [7].

Another way to represent the field is the use of coordinates. Each obstacle is stored with its coordinates and its size. This only for obstacles of any shape. Vectored storing of obstacles provides a higher accuracy and a lower

memory consumption but also rises the calculation effort.

In the problem at hand of circular shape, all obstacles have the same diameter. Considering the low memory space of micro controllers, the vectored approach is the better choice.

The error function consists of the path length and the collision penalty where $path_i$ denotes the length of the sub path, d_i the distance between path and the obstacle center in case the obstacle is hit, r_o the radius of the obstacle, and $c_{penalty}$ a penalty constant. The penalty for hitting an obstacle depends on the distance to its center. The deeper the path is in the obstacle, the higher the penalty should be. Consequently, the fitness raises when the error function lowers.

$$f = \sum_{i=1}^{4} path_i + \sum_{i=0}^{n_{collision}} c_{penalty} \cdot max(0, r_o - d_i) \qquad (1)$$

The collision penalty needs to have a larger influence than a long route. Therefore, $c_{penalty}$ is set to twice the length of the field. Consequently, when the error function has a higher value than twice the field length, no collision free route has been found.

C. Evolutionary Operations

Evolutionary algorithms find a problem solution by generating new individuals using evolutionary operators. The operators split into two main classes. Crossover operators exchange genes of two individuals, while the mutation operators modify genes of individuals by altering the values of genes. Both classes help to keep the population diverse.

Zheng et al. [8] proposed six mutation operators, which are specially designed for the problem field of path planning. These operators range from modification of one gene over exchange operators to insertion and deletion of way points.

As well genetic as evolutionary operators can influence the number of way points in the path and thereby the length of the gene.

IV. Implementation

When implementing an evolutionary algorithm the limits of the micro controller, on which the algorithm is executed must be considered. Due to the low available memory resources, the population size needs to be relatively small, which has a big influence on the diversity of the population.

The robot is moving in a dynamic environment. The robot cannot start to move without a valid path. This means that the calculation must fulfill real time constraints. As mentioned earlier, floating-point operations are expensive on micro controllers. The coordinates of all objects on the field are given in integers. Therefore the whole calculation is done in integer arithmetic.

A. Distance Calculation

The distance between two points is very easy to calculate.

$$l = |\vec{a}| = \sqrt{a_x^2 + a_y^2} \qquad (2)$$

When calculating this on a microcontroller, the square root operation needs particular attention. The fasted way to calculate this operation is the Heron algorithm [14], as

```
unsigned int isqrt(unsigned long int a)
{
  unsigned long int l;
  int i;

  l = 1 + (a>>2);
  for (i=0;i<14;i++)
    l = (a/l+l)>>1;
  return l;
}
```

Fig. 3 ANSI C implementation of the Heron Algorithm in C

shown in figure 3. 14 iterations cover the used coordinate range of 0 to 1023 pixels.

Test for collision between a sub path and an obstacle is a two-step process. The first check determines whether the obstacle is inside the rectangular shape that the sub-path spans in the orthogonal coordinate system. Hereby, borders of the size of the robot and the obstacle need to be added. The second test calculates the distance between path and obstacle. In the Small-Size league, it is sufficient to approximate all obstacles as circular shapes. The distance between a point or any circular shaped object on the field and the path line is calculated by the vector cross product. The cross product represents the area that is enclosed by the two vectors. To get the distance, the vector cross product is divided by the length of the path vector ($\vec{b}$). To get the real distance between the robot and the obstacle, the robot radius and the obstacle's radius need to be subtracted.

$$dist = \frac{|\vec{a} \times \vec{b}|}{|\vec{b}|} = \frac{|\vec{a}||\vec{b}|\sin(\vec{a}, \vec{b})}{|\vec{b}|} = \frac{|a_x b_y - a_y b_x|}{\sqrt{b_x^2 + b_y^2}} \qquad (3)$$

Those vector operations only need some basic mathematical operators and do not need much calculation time. They are therefore a good solution for micro controller based robots.

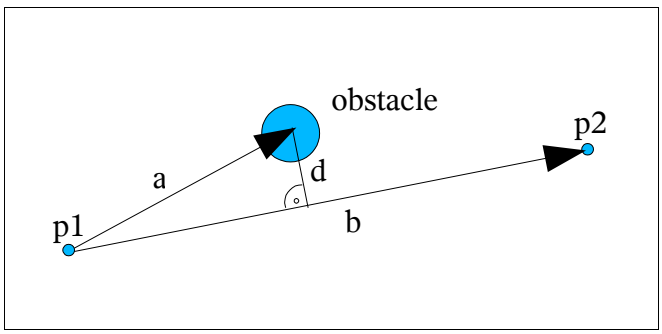


Fig. 4 Calculation of distance between path and obstacle

B. Continuous Calculation

Robots are not static devices. They move around, and their environment and with it the obstacle positions change. Even the destination position of the robot may change. Therefore, the path finding algorithm needs to run during the entire course from the start position to the destination. Due to this reasons, path finding on a robot is a continuing process. On the other hand, The robot does not need to know the best route before it starts driving; a found collision free route is sufficient.

The calculation is done in the main loop of the robot's control program. In the same loop, the data frame is evaluated, and the wheel speeds are calculated. The time between two received data frames is 35 ms. Due to the other tasks that need to be finished in the main loop, the evaluation time for path planning is limited to 20 ms. As the experiments will show, these constraints allow only for the evaluation of one complete evolution step during every control loop cycle. As mentioned above, the found route does not need to be perfect to start moving. Therefore, the robot does never need to wait longer then four cycles until it can start moving.

C. Measuring Time

As mentioned above, the path planning is only allowed to consume a certain amount of time. Therefore, the time spent for path planning needs to be measured. Fortunately, micro controllers provide easy to use 16 Bit hardware timers. The clock of the used micro controller runs at a speed of 16 MHz. Due to the needs of other software components, such as the PID-controller, this clock is divided by four. This results in a timer which has a resolution of 250ns and a duration of about 16,4ms. To measure a time, the timer value after the process to measure is subtracted by the timer value at the start of the measured process. Measuring longer periods of time is possible by splitting the process into two timer runs.

V. Experiments

In this section, the path finding algorithm is tested in various scenarios. All experiments are done directly on the robot. The time, the robot needs to find a valid solution and the time the robot needs to find the best solution is measured. A valid solution is a collision free route from the start point to the destination point. In addition, the route length must not be longer than twice the distance between the start and the destination.

A. Measuring the Calculation Time

In this first step, the time needed to evaluate a population is measured. The parameters vary from 1 to 3 for μ and 10 to 30 for λ. μ is denoting the parent population size while λ is denoting the number of children. The scenario includes four obstacles along the path. For this measurement a plus strategy is used. All times in Table I are averaged measurements with a maximal error of 0.9 ms. The timings vary because the randomly chosen genetic operators need different time to calculate.

The result indicates that it is possible to use up to 30 offspring in one generation. However, due to variations in calculation speed, it is saver to use only 20 offspring.

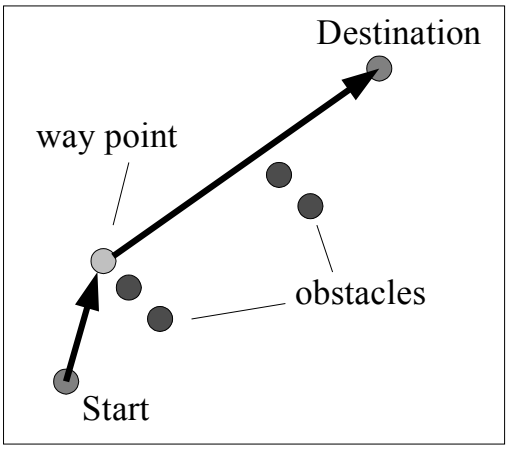
Table I

Calculation time for one generation depending on μ and λ.

μ	λ=10	λ=20	λ=30
1	5.5 ms	11.2 ms	15.5 ms
2	6.5 ms	14.8 ms	20.7 ms
3	7.2 ms	14.4 ms	20.5 ms

B. Finding a Path in a Static Environment

The next experiments will find the optimal values for μ and λ. The robot's task is to find the shortest path in two scenarios. Figure 5 shows the first scenario consists of four obstacles that are placed in pairs. The robot does not fit



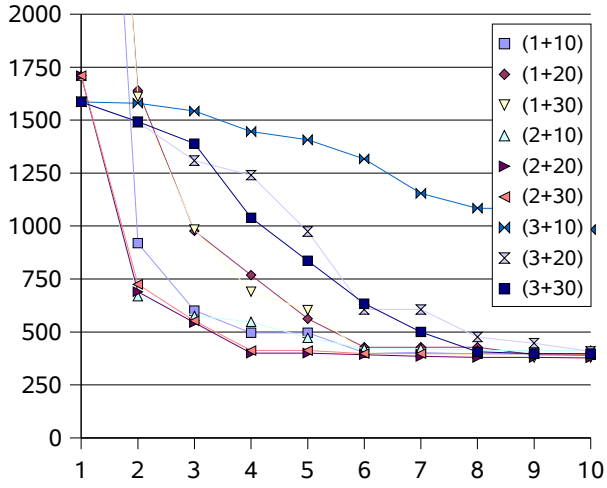


Fig. 5 Scenario 1 with four obstacles. The diagram shows the error function depending on the generation.

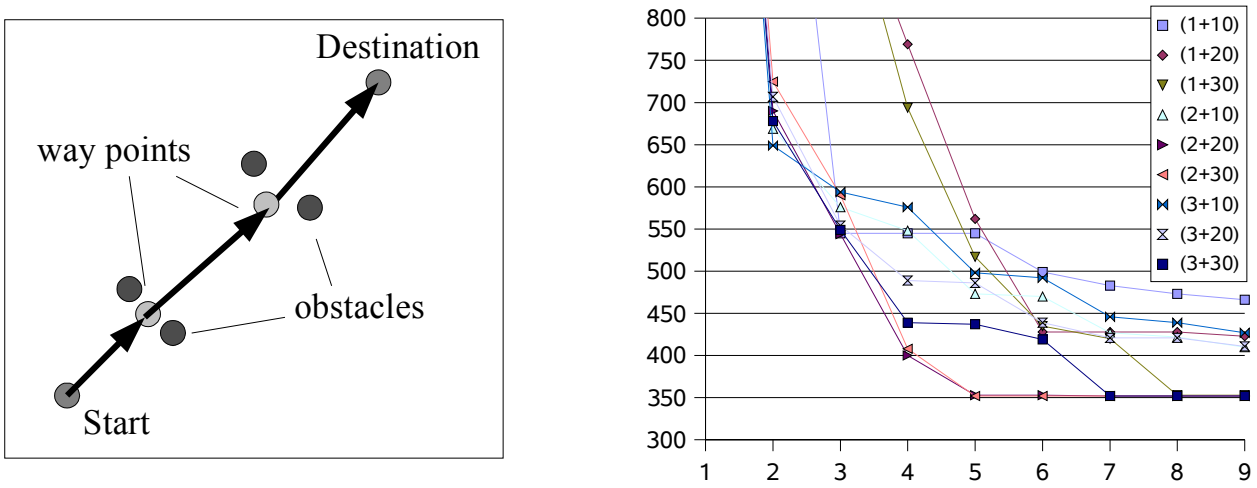


Fig. 6 Scenario 2 with four obstacles grouped as gates. The diagram shows the error function depending on the generation.

though the obstacles. The path planning algorithm needs to find a way around them. The distance between start and destination is 353 pixels. The shortest path has a length of 375 pixels.

The mutation probability is 60%, the probability of path lengthening or shortening is 20% respectively. In case of mutation, the modification a gauss curve of maximal 16 pixels width is multiplied with the current step size.

The results show that for this scenario, all tested (2+λ) strategies found a valid solution after only two generations. After two additional generations, the solution of the (2+20) and (2+30) strategy is already nearby the optimal solution. It should be noted that the (3+10) strategy performs worse than the other strategies, because three out of thirteen individuals are selected and therefore too many bad candidates join the next generation.

In the second scenario, two obstacles are placed besides the optimal path like gates. As Figure 6 shows, the optimal path goes through the gates.

The diagram in figure 6 shows that non of the (μ+10) strategies found the optimal path in the experiment. Again, the (2+20) and (2+30) strategies perform very well and find a valid solution after only three generations. The nearly best path is found after only five generations. This scenario also shows that (2+λ) strategies are in this case faster in finding a solution than (1+λ) strategies.

In both experiments the robot found a valid path after only two generations. Due to the time needed to calculate one generation, the robot needs to wait two cycles before it can start driving. While the robot drives, the path is improved with each iteration. The results also show that a (2+20) strategy is a good compromise between calculation speed and memory consumption.

C. Dynamic Environments

In real-world scenarios, the obstacles as well as the robot are moving. The last experiment shows how the robot behaves in a changing environment. Unlike the other experiments which were done directly on the robot, this experiment was simulated, because it is a lot effort to synchronize the robot's and obstacle's movement with physical hardware. The algorithm code is exactly the same.

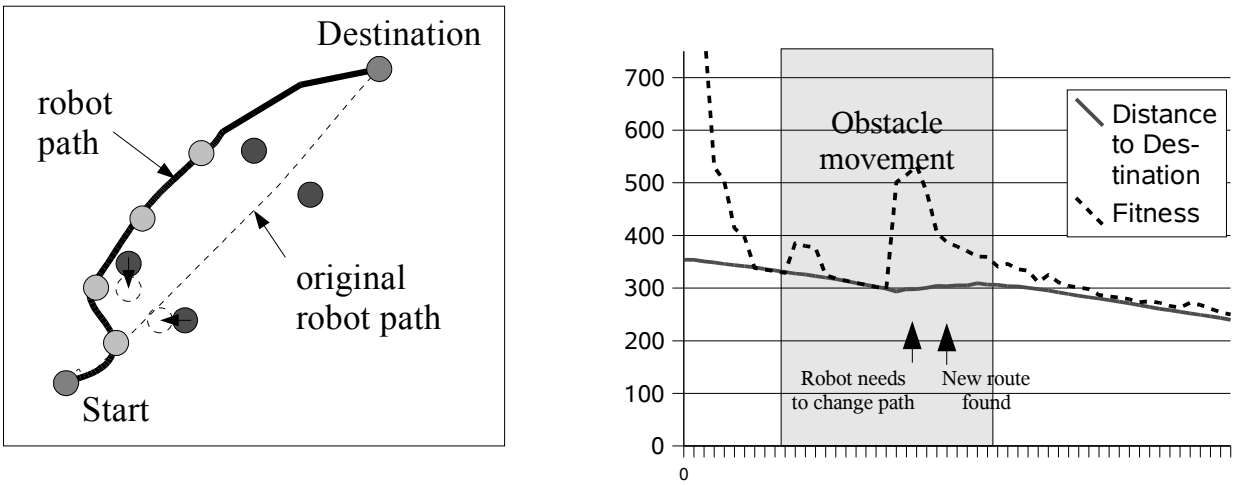


Fig. 7 Robot movement in a dynamic environment

The dynamic test scenario is similar to the static scenarios. Actually, scenario 2 is transferred into scenario 1 by moving the obstacles when the robot is driving. The movement of the obstacles starts at time step 10 and finishes at time step 30. The robot drives with a speed of 5 pixels per time step. At the beginning, the obstacles are positioned in a way that the robot has enough space between them. In their end position, the robot needs to drive around them.

The graph in Figure 2 shows that until the obstacles start to move, the error function has the same value as the direct distance to the destination. As soon as the obstacle starts to move, the robot is adjusting its path. At time step 22, the distance between both obstacles is smaller than the robot size. At this point, the fitness function raises by factor of two. The algorithm finds a new route within four time steps.

For this experiment, a (2+20)-strategy was used. Because the fitness function changes when the obstacles or the robot move, found solutions need to be re-calculated in each step. Otherwise, the robot will not change its path as a found solution remains valid.

VI. Conclusion

This paper presented the implementation of path planning based on evolutionary algorithms on a RoboCup small-size league robot. In particular, it demonstrated the implementation of genetic algorithms on the robot's micro controller. The optimal evolutionary algorithm was evaluated in the experiments. Furthermore, this paper showed that the implementation meets the real-time constraints of the robot environment. The algorithm's capability to find a path from source to destination and to adapt it to environmental changes was shown in realistic scenarios.

Further research will focus on the implementation of corporate path planning and prediction of obstacle movement. Both will lead to improved reaction time to obstacle movement.

References

[1] http://www.robocup.org

[2] J.F. Canny, “The Complexity of Robot Motion Planning”, MIT Press, Cambridge, MA, 1988.

[3] S. Udupa, “Collision Detection and Avoidance in Computer Controlled Manipulators”, PhD thesis, Dept. of Electrical Engineering, California Institute of Technology, 1977.

[4] http://www.freescale.com

[5] X. Du, H.-h. Chen, W.-k. Gu, “Neural network and genetic algorithm based global path planning in a static environment”, Journal of Zhejiang University SCIENCE, 2005 6A (6):549-554

[6] B.-T. Zhang, S.-H. Kim, “An Evolutionary Method for Active Learning of Mobile Robot Path Planning”, Proc. 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA'97), pp. 312-317, 1997.

[7] K.H. Sedighi, K. Ashenayi, T.W. Manikas, R. L. Wainwright, H.M. Tai, “Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm”, Proc. 2004 IEEE Congress on Evolutionary Computation (CEC2004), p. 1338-1345.

[8] C.W. Zheng, M.Y. Ding, C.P. Zhou, “Cooperative Path Planning for Multiple Air Vehicles Using a Co-evolutionary Algorithm”, Proceedings of International Conference on Machine Learning and Cybernetics 2002, Beijing, 1:219-224.

[9] J. Thomas, A. Blair, N.Barnes, “Towards an efficient optimal trajectory planner for multiple mobile robots, Proceedings of the 2003 International Conference on Intelligent Robots and Systems, 2291-2296.

[10] small-size league rules http://www.cs.cmu.edu/~brettb/robocup/rules/f180rules2006.pdf

[11] FU-Fighters team description http://robocup.mi.fu-berlin.de/docs/fufighters06TDP.pdf

[12] b-smart team description http://b-smart.informatik.uni-bremen.de/B-Smart_TDP2006.pdf

[13] Vienna Cubes team description http://cubes.technikum-wien.at/media/documents/TDP Vienna Cubes 2005.pdf

[14] T. L. Heath, A History of Greek Mathematics, 2 vol. (1921, reprinted 1993).