Local Movement Control with Neural Networks in the Small-Size League

Steffen Prüter, Ralf Salomon, and Frank Golatowski

University of Rostock, Faculty of Computer Science and Electrical Engineering, Institute of Applied Microelectronics and Computer Engineering, 18051 Rostock, Germany
{steffen.prueter, ralf.saloman, frank.golatowski}@uni-rostock.de

Abstract. In the RoboCup small-size league, it is common to calculate the robot's position calculated via a camera over the field as well as different kinds of artificial intelligence that's run on a PC outside the field. In this case, the robot's position must be predicted because of the various delays until the position data arrived at the robot. This paper focuses on the use of local sensors on the robot and a neural network to estimate the robot's actual position. This paper shows how local sensors can compensate for the effect of latency times and how robot's actual position can ascertained. Slip and friction effects that cannot be measured with local sensors are adjusted by a neutral network.

1 Introduction

RoboCup [1] small-size league is of particular interest, because it combines engineering tasks, such as building robot hardware and designing electronic components, with computer science applications, such as localization of objects, finding the robots' positions, and calculating the best path through obstacles. Another interesting challenge emerges from the requirement that all team members have to communicate with each other in order to develop a cooperative behavior. Research on artificial intelligence may help find the optimal solution in all of these areas.

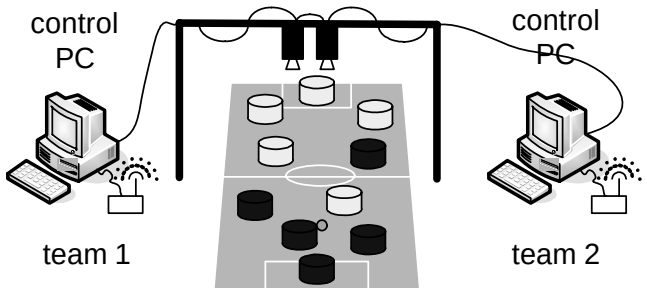

Fig. 1. The physical setup in RoboCup's small-size league.

Fig. 1 show two cameras mounted approximately four meters above the floor and observe the field of four by five meters in size on which two teams consisting of five robots play. The processing sequence starting at the camera image and ending at the robots executing their received (action) commands suffers from significant time delays, as Fig. 2 illustrates.

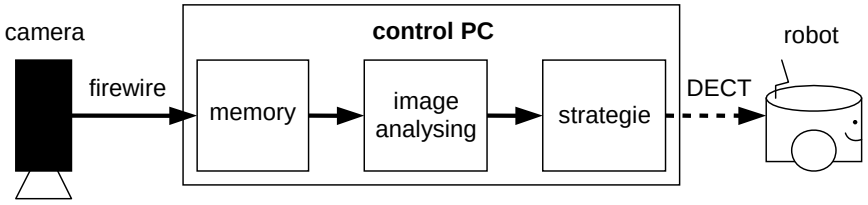

Fig. 2. The image processing system consists of five stages which all contribute to processing delays also known as latency time.

These time delays have the consequence that when receiving a command, the robot's current position does not correspond to its position in the camera image. Consequently, the actions are either inaccurate or may even lead to improper behavior in the extreme case. For example, the robot may try to kick the ball even though it is not in reach anymore.
Section 2 discusses how the position correction can be further improved on the robot itself and how local sensors can alleviate this problem to a large extent.

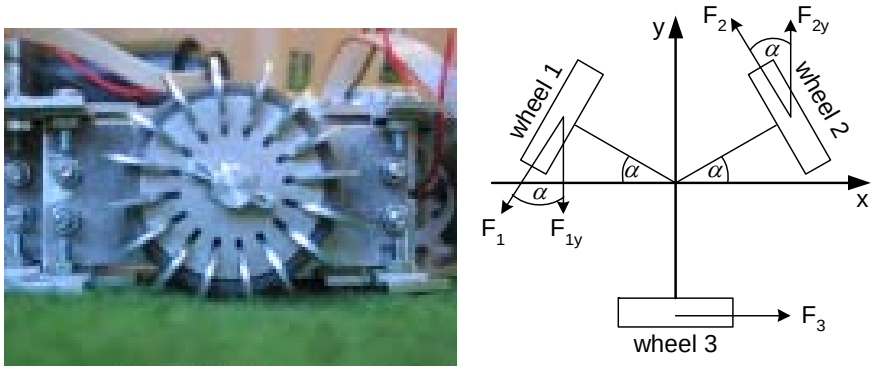

Fig. 3. An omnidirectional drive with its calculation model.

Fig. 3 shows the omnidirectional drive commonly used by most robots of the small-size league. As can be seen, an omnidirectional drive consists of three wheels, which are twisted to each other by 120 degrees. This drive has the advantage that a robot can be simultaneously doing both moving forward and spinning around its own central axis. Furthermore, the particular wheels, as shown on the left-hand-side of Fig. 3, yield high grip in the rotation direction, but almost-vanishing friction perpendicular to it. The specific orientation of all three wheels, as illustrated on the right-hand-side of Fig. 3, requires advanced controllers and exhibit higher friction than standard two-

wheel drives. The later requires sophisticated servo loops and (PID[1]) controllers [9]. Depending on the carpet and the resulting wheel-to-carpet friction, one or more wheels may slip. As a consequence, the robot leaves its recalculated moving path. To this end, a robot employs its own back-propagation network to learn its own specific slip and friction effects. Section 0 concludes this paper with a brief discussion including possible future research.

2 Local Sensors

As has been outlined in the introduction, the latency caused by the imaging-processing-and-action-generation loop leads to non-matching robot positions. As a measurable effect, the robot starts oscillating, turning around the target position, missing the ball, etc. An approach to solve the latency problem is to do the compensation calculation on the robot itself. The main advantage of this approach is that the robot's wheel encoders can be used to obtain additional information about the robot's actual behavior. However, since the wheel encoders measure only the wheel rotations, they cannot sense any slip or friction effects directly.

2.1 Latency time

RoboCup robots are real-world vehicles rather than simulated objects. Therefore, all algorithms have to account for physical effects, such as inertia and delays, and have to meet real-time constraints. Because of the real-time constraints, perfectly exact algorithms would usually require too much calculation time. Therefore, the designer has to find a good compromise between computational demands and the precision of the results. In other words, fast algorithms with a just decent precision are the method of choice here [2], [3].

As has already been mentioned in the Introduction, the latency is caused by various components including the camera's image grabber, the image compression algorithm, the serial transmission over the wire, the image processing software, and the final transmission of the commands to the robots by means of the DECT modules. Even though the system already uses the compressed YUV411 image format [5], the image processing software as well as the DECT modules are the most significant parts with a total time delay of about 200ms. For the top-level control software, which is responsible for the coordination of all team members, all time delays appear as a constant-time lag element. The consequences of the latency problem are further illustrated in Fig. 4 and Fig. 5.

[1] PID is the abbreviation of proportional-integrate-differential. For further detail, the interested reader is referred to [9]

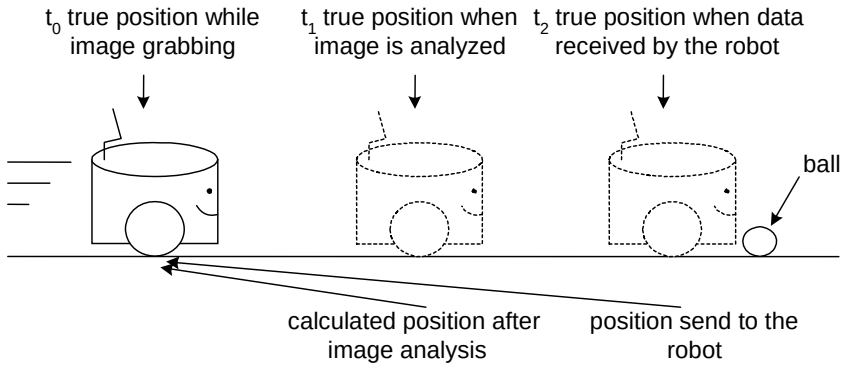

Fig. 4. Due to the latency problem, the robot receives its commands at time t_2 , which actually correspond to the image at time t_0.

Fig. 4 illustrates the various process stages and corresponding robot positions. At time t_0, the camera takes an image with the robot being on the left-hand-side. At the end of the image analysis (with the robot being at the old position), the robot has already advanced to the middle position. At time t_2, the derived action commands arrive at the robot, which has further advanced to the position t_2 to the right-hand-side. In this example, when being in front of the ball, the robots receive commands which actually belong to a point in time in which the robot was four times its body length away from the ball. Fig. 5 illustrates how the time delay between image grabbing and receiving commands leads to an oscillating behavior at dedicated target positions (marked by a cross in the figure) [6], [7], [8], [10].

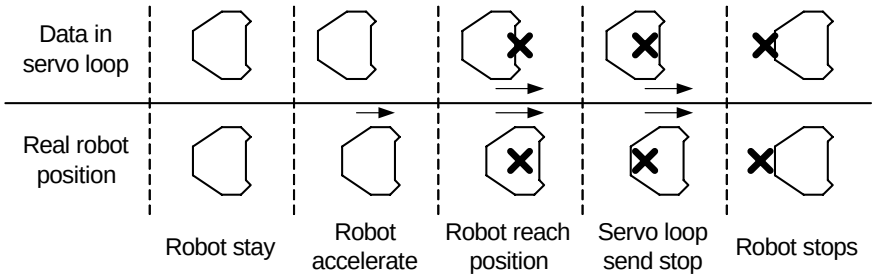

Fig. 5. Problem of stopping the robot at a desired point.

2.2 Experimental Analysis

In order to compensate for the effects discussed above, the knowledge of the exact latency time is very important. The overall latency time was determined by the following experiment:

The test software was continuously sending a sinusoidal drive signal to the robot. With this approach, the robot drives 40cm forward and than 40cm backwards. Then, the actual robot position as was seen in the image data was correlated with the control commands. As Fig. 6 shows, the duration of the latency time is seven time slots in length, which totals up to 234ms with 30 frames send by the camera.

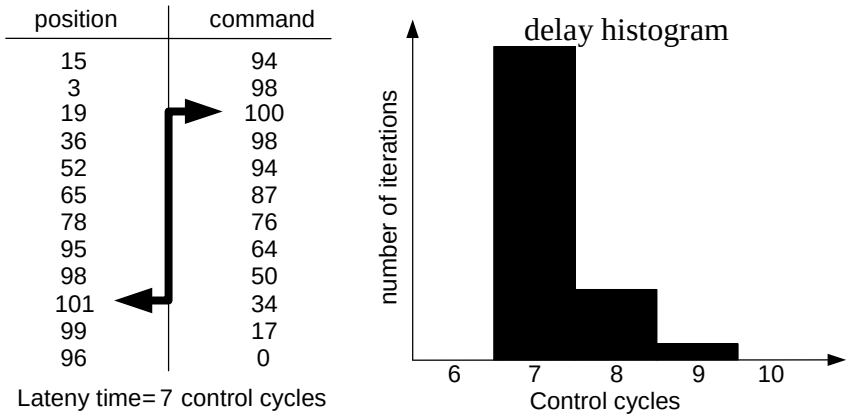

Fig. 6. Detection of the Latency time in the control loop

It might be worthwhile to mention here that for technical reasons, the time delay of the DECT modules is not constant; the jitter is in the order of up to 8ms. The values given above are averages over 100 measurements.

2.3 Increased Position Accuracy by Local Sensors

In the ideal case of slip-free motion, the robot can extrapolate its current position by combining the position delivered by the image processing system, the duration of the entire time delay, and the traveled distance as reported by the wheel encoders. In other words: in case slip does not occur, the robot can compensate for all the delays by storing previous and current wheel tick counts. This calculation is illustrated in Fig. 7.

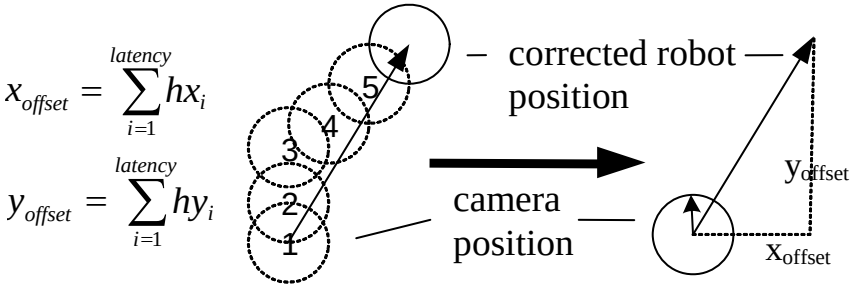

Fig. 7. Extrapolation of the robot's position using the image processing system and the robot's previous tick count.

Since the soccer robots are real-world entities, they also have to account for slip and friction, which are among other things, nonlinear and stochastic by nature. The following section employs back-propagation networks to account for those effects.

3 Embedded Back-Propagation Network

Due to the resource limitations of the robot hardware, the number of nodes and connections that the robot can store on its hardware is quite limited. From a hardware point of view, the memory available on the robot itself is *the* major constraint. In addition to the actual learning problem, this section is also addresses the challenge of finding a good compromise between the network's complexity and its processing accuracy.

A second constraint to be taken into account concerns the update mechanism of the back-propagation learning algorithm. As is well known, back-propagation temporarily stores the calculated error sums as well as all the weight changes Δw_{ij} [4]. This leads to a doubling of the memory requirements, which would exhaust the robot's onboard memory size even for moderately sized networks. As a workaround, this section stores those values on the central control PC and communicates the weight changes by means of the wireless communication facility. This separation is illustrated in Fig. 8.

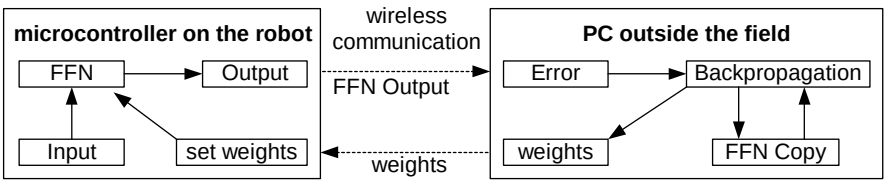

Fig. 8. Separation of the actual feed-forward network (indicated by FFN in the figure) and the back-propagation training algorithm.

3.1 Methods

As has been discussed above, the neural network has to estimate the robot position also when slip and/or friction occur. Since the coding of the present problem is but trivial, this section provides a detailed description of it. In order to avoid a combinatorial explosion, the robot is set at the origin of the coordinate system in every iteration. All other values, such as the target position and orientation, are relative to that point.

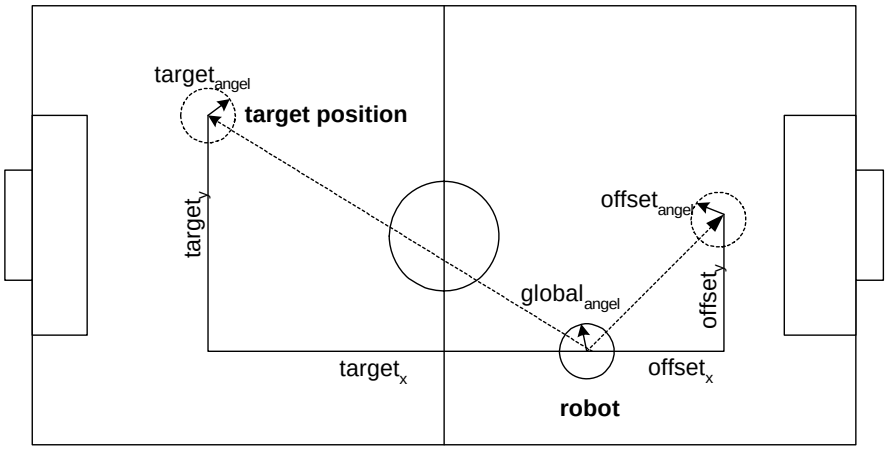

Fig. 9. And example configuration for the slip and friction compensation. For details please see text.

Fig. 9 illustrates an example configuration. This configuration considers three robot positions labeled “global”, “offset”, and “target”. The first robot corresponds to the position as provided by the image processing system. The second position, called “offset”, corresponds to the robot’s true position and hence includes the traveled distance during the time delay. The third robot symbolizes the robot’s target position. As has already been mentioned above, the neural network estimates the robot’s true positions (labeled by “offset”) from the target position, the robot’s previous position, and its traveled distances. The relative values mentioned above are scaled such that they fit into -40 to 40, and all angels are directly coded between 0 and 359 degrees. With all these values the input layer has to have seven nodes.

Due to the limited calculation capabilities of the microcontroller, all values of the neural network may be stored in integer quantities. In this format every operation on the microcontroller is done in two processing steps because of the mathematical coprocessor. For this the feed forward network calculation (FFN) on the robot must be adapted. To this end a simulation of different FFNs on a PC provide important criteria for the implementation on the robot.

Foremost different FFN structures compared and the results, they offer valuable clues to the structure, illustrated in Fig. 10. All experiments where done with 400 pre-selected training patterns and 800 test patterns. The initial learning rate was set to

$\eta = 0.1$. During the course of learning, the learning rate was increased by 2% in case of decreasing error values and decreased by 50% otherwise. It should be noted that in 10% of all experiments, back-propagation got stuck in local optima. These runs were discarded and are not further considered in this paper. Learning was terminated, if no improvement could be achieved over 100 consecutive iterations.
As you can see the one and two hidden layer networks provide an equivalent accuracy. Networks with more hidden layers no be considered due to the fact that the calculation power and time is limited. The varieties between the average errors of different node counts also low. The outcome of this is that the network structure of choice is a one hidden layer network with five nodes. This network is a good compromise between network accuracy and calculation time.

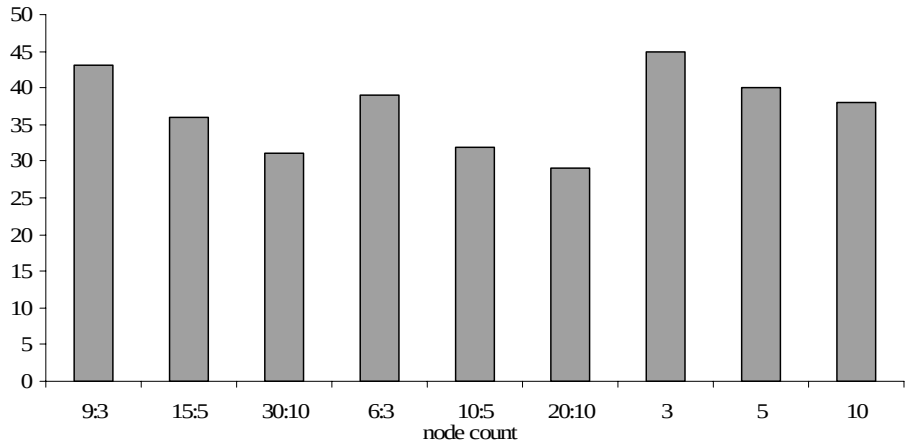

Fig. 10. Average error in mm of two and one hidden layer FFNs

The next step is the adaptation of the selected FFN on the robot. The measurements show that all resulting network weights Δw_{ij} are in the range of -10 to 10. The integer variable on the microcontroller has a range between -32,768 and 32,767. So all weights multiplied by 1,000 to fit into the integer range. The input values, i.e., global, offset, and target, position are multiplied by 100. The network input of all nodes calculated with the formula

$$net_i = \sum_i w_{ij} o_j$$

is a weighted sum of all nodes j to which it is connected by weight Δw_{ij}. Hence the node input net_i on the microcontroller is factor $1{,}000 \cdot 100 = 100{,}000$ higher than on the PC side. As an exception, the node input stored as long integer in consequence of the possible high range. The neural network learning process on the PC side calculates with floating point arithmetic to evaluate the sigmoid function and the node output

$$o_i = 1/(1 + e^{-net_i}) \ ,$$

which is also required in of microcontroller. The calculation of the results is difficult to implement because the limited calculation time on the robot. The answer to the problem is to store the network's values in a constant predefined array, because no RAM and calculation power is required for the activation function. The input values of the array are multiplied by 10 and the output values by 10,000, respectively to match the net input net_i, the sum is divided by 10,000. The simulation on the PC shows that the net input net_i has a maximum range of -10 to 10 thus the array consists of 201 values. With these modifications, the calculation of the FFN is feasible on the microcontroller.

3.2 Results

The compares of the FFN allow the implementation on a small-size league robot. Fig. 11 shows the average position error in mm without the FFN with the error of the FFNs simulated on the PC and on the robot. As can be seen the FFN provides a gain of 50% in accuracy. The error caused by the modification of the FFN on the robot is less than 8%.

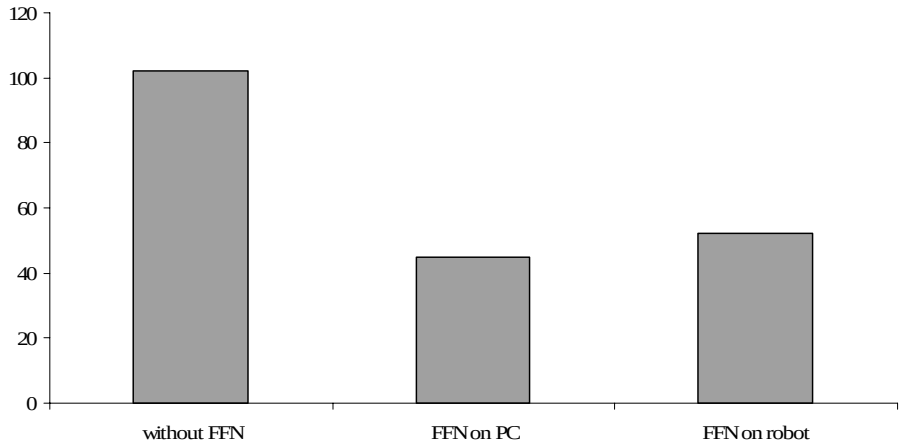

Fig. 11. Average error in mm without FFN, the simulated average error on the PC, and the average error on the robot.

A second measurement evaluates the quality of the correction by driving an 8-shaped figure. In this real world test, the robot is controlled by the camera and the PC outside, as has been suggested by others [10]. This test environment shows how precise and fast the robot can drive. The driven figure has a size of three by one meter and is cut into 64 areas. The PC outside the field checks the robot's position during the measurement and sets the new area as target position when the robot has reached the area before, so the robot cannot derivate from its way. The results shown in Table 1 exemplify that the robot's speed has significant increased on the field via the employment of FFN.

Table 1. Average time needed to drive the test figure.

	Robot	Robot with History	Robot with FFN
Time	8,2s	6,8s	5,9s

This results show that local wheel sensors implemented on the robot advance the accuracy of movement control. But wheel sensors cannot measure slip and friction effects. Back-propagation networks can reduce the positioning errors caused by these effects, but most microcontrollers and embedded devices cannot provide the required calculation power and memory. The adaptation of FFN to accomplish the hardware limitations on autonomous robots was also successful. The measurements show that only marginal variations between the common FFN and the adapted version occur. So, this advancement can be used for further implementations and other developments.
All simulations with different FFN structures and the given input parameters have an error of at least 30 mm. At this point, further research may investigate other learning and self-adaptive principals, such as Hebbian learning and the implementation of short-cuts [4], [11]. Furthermore, the input data can be expanded to all values of the history, so that the back-propagation network can also include different acceleration data in its calculation.

4 Conclusions

The focus of this paper was on the small-size league in which two teams of five robots each play soccer against each other. Since no human control is allowed, the system has to control the robots in an autonomous way. To this end, a control software analyzes images sent by two cameras and derives appropriate control commands for all team members.
Unfortunately, the image processing system exhibits various time delays at different stages, which leads to erroneous robot behavior. Section 2 has shown how local sensors compensate those effects.
The omnidirectional drives used by most research teams exhibit certain inaccuracies due to two physical effects called slip and friction. Section 3 has indicated that neural networks are able to significantly improve the robot's behavior with respect to accuracy, drift, and response.
Furthermore, the architectures presented here still require hand-crafted adjustments to some extent. In addition, the resources available on the mobile robots significantly limit the complexity of the employed networks.
First of all, future research will be addressing the problems discussed above. For this goal, the incorporation of short-cuts into the back-propagation networks seems a promising option. Another important aspect will be the development of complex controllers that would fit into the low computational resources provided by the robot's onboard hardware.

References

[1] http://www.robocup.org (Official Website).
[2] A. Gloye, M. Simon, A. Egorova, F. Wiesel, O. Tenchio, M. Schreiber, S. Behnke, and R. Rojas: Predicting away robot control latency, Technical Report B-08-03, FU-Berlin, June 2003.
[3] J. Zagal, and J.Ruiz-de-Solar, Learning to Kick the Ball Using Back to Reality, RoboCup-2004: Robot Soccer World Cup VIII, Springer, 2004
[4] R. Rojas: Neural Networks - A Systematic Introduction, Springer-Verlag, Berlin, 1996.
[5] J. C. Russ, The Image Processing Handbook, Fourth Edition, CRC Press, 2002
[6] J.C.Alexander and J.H. Maddocks, On the kinematics of wheeled mobile robots, Autonomous Robot Vehicles, Springer Verlag, pp.5-24, 1990.
[7] A. Miene, U. Visser, and O. Herzog, Recognition and Prediction of Motion Situations Based on a Qualitative Motion Description, RoboCup-2003: Robot Soccer World Cup VII, Springer, 2003
[8] R. Balakrishna, and A. Ghosal, Two dimensional wheeled vehicle kinematics, IEEE Transaction on Robotics and Automation, vol.11, no.l, pp.126-130, 1995
[9] K. J. Astrom, T. Hagglund, PID Controllers: Theory, Design, and Tuning, International Society for Measurement and Con; 2nd edition, 1995
[10] A. Gloye, C. Göktekin , A. Egorova, O. Tenchio, and R. Rojas, Learning to Drive and Simulate Autonomous Mobile Robots, RoboCup-2004: Robot Soccer World Cup VIII, Springer, 2004
[11] T. Bäck, B. Fogel, and Z. Michalewicz, Evolutionarily Computation 1: Basic Algorithms and Operators, Bristol UK: Institute of Physics Publishing, 2000