Immortals 2011 Extended Team Description Paper

Mohammad Reza Niknejad[1], Seyed Ali Salehi Neyshabouri[2], Seyed Ali GhaziMirSaeed[3], Ehsan Kamali[4], Mohammad Hossein Fazeli[5], Yousof Piran[4], Mustafa Talaeezadeh Khouzani[6], Seyed Sadegh Mohseni Salehi Monfared[2]

[1] Department of Computer Engineering of Iran University of Science and Technology
[2] Department of IT Engineering of Sharif University
[3] Department of Mechanical Engineering of Tehran University
[4]Department of Electronic Engineering of Shahed University
[5]Department ofMechanical Engineering of Iran University of Science and Technology
[6] Department of Computer Engineering of Shahid Beheshti University

Abstract. Some achievements of Immortals team that seemed innovative and useful are explained in this paper. This paper is meant to improve these methods by sharing them with the community, and help new teams achieve the minimum requirements for participating in the competitions. At first some utilizations of GPU processing in this league are mentioned, and then a novel approach to Navigating in noisy environments is explained.

1 Introduction

"Immortals" is a robotic team consisting of university students of Sharif, Tehran, IUST, Shahed and Shahid Beheshti Universities. The team was formed in 2003 to attend junior soccer league competitions. The team successfully participated the 3rd Hellicup Robotic competitions in 2004. After placing 3rd we qualified for 2005 Osaka Robocup, taking part under the name Robonik. The team also successfully participated IranOpen Robotic competitions in 2006 & Iran Open 2007. The small size project started in summer 2007 and simple-structured robots were made by summer 2008 and have continually improved since then. After participating Robocup 2009 Graz and Robocup 2010 Singapore competitions, it was decided to gear up for the next competitions with revising the whole Mechanical system and equipping the robot with a sophisticated electronic system to extract the maximum efficiency and maneuverability from the robots. Inheriting all preceding robots' strengths while decreasing their weaknesses, a brand new generation of robots is set to participate the Robocup 2011 competitions.

2 Software

The overall software architecture is to take information of the field from SSL-Vision over network, filter this information and pass them to the AI. The AI calculates a target for each robot. Finally, it transmits the processed data to the robots, in each frame.

2.1 Calculations on GPU using CUDA

Driven by the insatiable market demand for real-time, high-definition 3D graphics, the programmable Graphic Processor Unit or GPU has evolved into a highly parallel, multithreaded, many-core processor with tremendous computational power and very high memory bandwidth. In November 2006, nVidia® introduced CUDA™, a general purpose parallel computing architecture – with a new parallel programming model and instruction set architecture – that leverages the parallel compute engine in nVidia GPUs to solve many complex computational problems in a more efficient way than on a CPU [9]. In comparison with CPU, graphic cards have much more cores each having a separate memory unit which results in reducing the flow control for a single process. CUDA comes with a software environment that allows developer to use C as a high-level programming language. It lets the programmer to get the desired results from some heavy and complex tasks much faster and allows some algorithms to be implemented in real-time which were almost abandoned due to complexity and the time required to get reasonable results. The presented method is to parallelize AI algorithms, so they could be implemented on GPU using CUDA. The algorithms descriptions and the results from both CPU and GPU are followed.

2.1.1 Field Evaluation

One of the most important aspects of artificial intelligence in a small-size league robot is the field evaluation methods. In these methods each individual point of the field is evaluated for a specific purpose, for example to find the best point for receiving a pass from a teammate possessing the ball (Fig. 1). As explained in [1] each point of the field is evaluated considering some parameters like the largest free angle toward the goal.

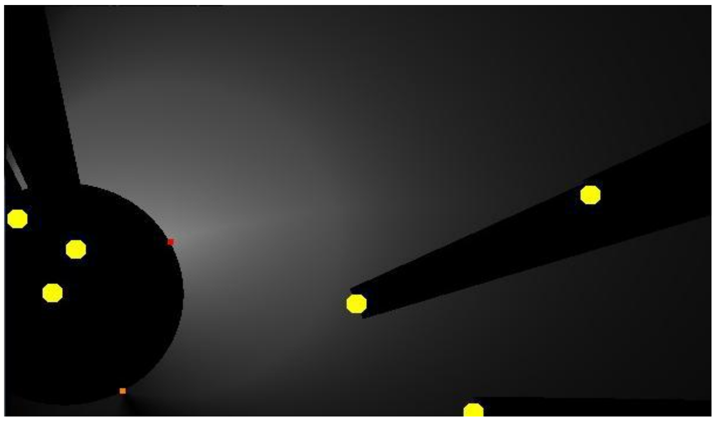

Figure 1: An example of evaluated field points for receiving pass from where the ball is located (the orange rectangle down left). Yellow circles are robots. Brighter points have more value. The red rectangle has maximum value and is the answer of the evaluation function

If the SSL field having the dimensions of 6.05*4.05m is broken down to 6050*4050 points, implementing passing evaluation method on the CPU takes about 1500ms as shown in figure 2. Such processing time is not acceptable in a field as dynamic as the field of small-size robot.

The first solution that emerges into mind is to breakdown the field into bigger parts. It is clear that decreasing the number of points in each dimension by factor of n will result in processing time to become $1/n^2$ of the original time. Thus to make the task executable in the desired time (1ms) the field should be divided into 67*100 point. Such allocation has been tested in previous robotic competitions by our team Immortals and the results were not satisfactory.

Since evaluation functions for each point of the field are calculated independently from other points, they could be implemented in parallel using CUDA. Executing passing evaluation method on the GPU with 6050*4050 allocations takes only 43ms (Fig. 2) which is a real-time evaluation of a high definition allocation (35 times faster than CPU).

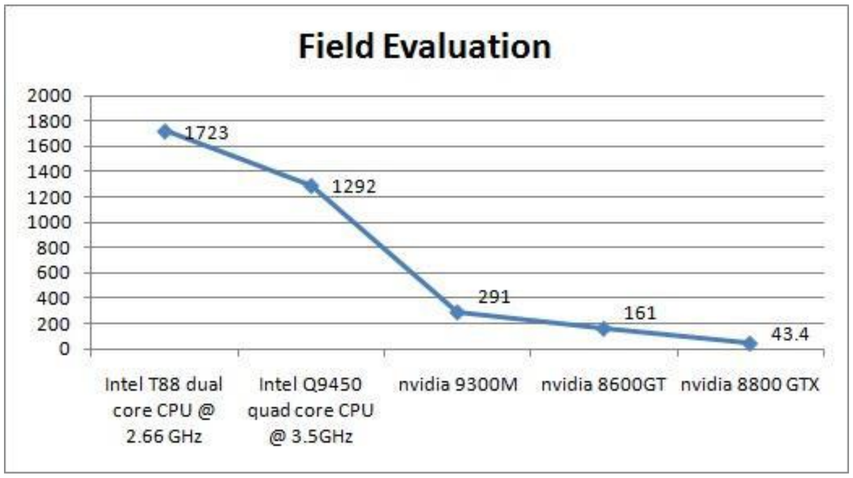

Figure 2: Field evaluation execution time on multiple CPUs and GPUs.

2.1.2 Artificial Potential Field Planning

In [7] an approach is proposed for robot path planning that consists of incrementally building a graph connecting the local minima of a potential field defined in the robot's configuration space and concurrently searching this graph until a goal configuration is attained.

For every expansion step of the graph, all neighboring points should be evaluated. This paper suggests to evaluate all points of the field simultaneously on GPU using CUDA, and to determine local minima of every step by CPU. The experiments show a huge improvement in results as shown in figure 3, especially in multi-agent cases, because same evaluation results would be reused for all agents.

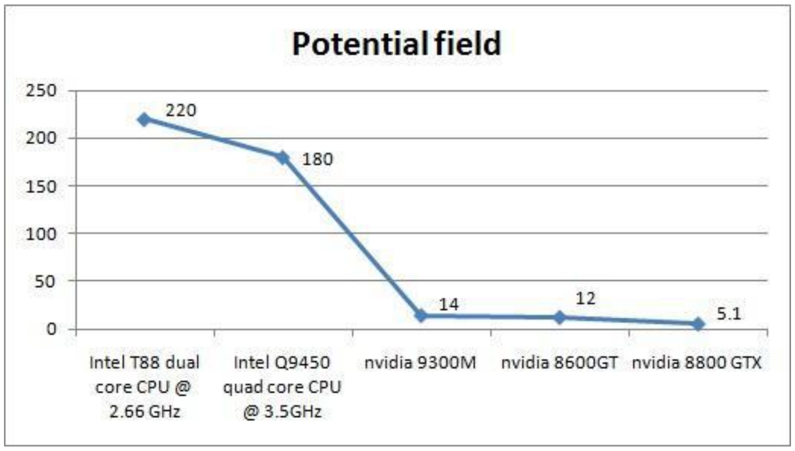

Figure 3: Potential field execution time on multiple CPUs and GPUs.

2.1.3 Physic-Based Behavioral Control

Separately implementing higher level tactics and lower level navigation control can lead to tactics which do not fully utilize the robot’s dynamic actuation abilities. It can furthermore create the problem of the navigational code breaking the constraints of the higher level tactical goals when avoiding obstacles [1]. For example when a robot is executing “shoot on goal” skill [8], it shoots the ball to the center of the largest free angle toward the goal from its position. Doing so may not be necessarily the best solution. In some cases dribbling a defender might put the robot in a much better situation, or shooting the ball toward a defender might cause the ball to deflect toward the goal. In the method presented by Zickler [1] this problem is almost solved. This method finds the best solution by simulating different ways to run a probabilistically modeled play or skill [8] by a rigid-body-simulator like nVidia PhysX™.

The main difficulty of this approach is its low speed which has not allowed it to be implemented in real-time even 2 years after its development. As mentioned in [1] it takes 1 to 30 seconds to calculate the path, depending on the size of the tree. It should be considered that the tree which takes 1 second to be computed consists of 1000 nodes, and its average success rate in an example run by the authors is less than that of the linear tactic. In fact a tree with reliable success rate should have at least 10,000 nodes taking about 4 seconds to be calculated.

One of the main reasons for its high time consumption is that the computations of physics engine requires a lot of time. This problem could be easily resolved by using PPUs (Physics Processing Unit). It would resolve half of the algorithm’s time-consumption problem [1] and the other half could be solved by using CUDA with parallel-Expanding RRT mentioned in the previous section.

This method is still under development and furthermore, the question how to re-use the previously computed result during re-planning needs to be answered.

2.2 SB-RRT

One of the relatively recently developed planners that was evolved to tackle the planning problem is RRT [10]. RRT uses random states to rapidly explore the state space. This algorithm is specially effective and efficient in fields with higher degrees of freedom. Given enough time, the probability that the planner fails to find a path, if one exists, asymptotically approaches zero.

Failure is defined as "no suitable trajectory found from the initial state to the target or goal state, if one exists". To address the challenges brought up by real-time applications, RRTs are moderated to return a feasible solution quickly, paying almost no attention to the quality of the solution. Besides, in typical implementations [11], the algorithm is not terminated as soon as the first solution is found; but, all the available computation time is used to search for an improved solution, with respect to a performance metric such as time, path length, etc. Koren Y. Borenstein et Al. proposed an approach for robot path planning that consists of incrementally building a graph connecting the local minima of a potential field defined in the robot’s configuration space and concurrently searching this graph until a goal configuration is attained [12]. So the planned path will be relatively safe, but due to the greediness nature of Potential Field approach, it may stick onto the local minima criteria and won't be able to generate a complete outcome.

Sampling based algorithms only guarantee that there is no obstacle in the planned path due to their random nature. A robot with no noise in its actuators and sensory data could move along the planned path with no collision. But in practical and real cases there is noise in both actuator and sensor functionalities and belonging a path to the Cfree does not guarantee safety of the path. A noise in the sensors’ input data may result in a wrong Cfree which may lead to a path in Cobstacle and eventually a collision. Cfree corresponds to the part of the state space which is free of obstacles, and Cobstacle is the complementary of Cfree. Noise could also result in actuators malfunction which leads to deviation from the desired path, resulting in a collision. One simple way to avoid getting such results is to increase the Cobstacle space, in a way which outcome of the planner, even with sensor and actuator noise, falls within the Cfree. The drawback of this method is that it might result in losing some existing paths in case of having some narrow passages. On the other hand, the potential field algorithm makes a reasonable distance between the planned path and obstacles, and is more likely to generate a path in real Cfree duo to its evaluating nature which chooses states with higher safety value. A value of safety, S(p) for every state, p, is defined as its distance to the nearest obstacle. In other words, it is the radius of the biggest circle with the center of p which entirely falls within the Cfree space.

The safety of a path can now be defined as:

$$S_{path} = \frac{\int_L S(p)\,dl}{L}$$

where L is the length of the path. Duo to discrete nature of the path, it is modified to:

$$S_{path} = \frac{\sum_1^n S(p_i)}{n}$$

where n is the number of waypoints, representing the path. As mentioned before, a positive value of safety is enough for the RRT to consider it as a safe state, which means with the slightest amount of noise, the real safety might be negative, but the planner consider it positive, which leads to having a collision along the path. The importance of the safety varies in different fields. For instance in fields like autonomous urban driving [18], that collision is unacceptable, it is preferred for the vehicle to standstill rather than moving and getting into an imminent collision. But in some adversarial fields like the Robocup soccer Small Size League (SSL) competitions [14], the first priority is the movement of robot to the desired destination rather than staying still in order to avoid a collision. Consequently the planner should be able to adjust the trajectory's safety level based on the field.

Safety Biased-RRT (SB-RRT) is an extension to the RRT, which suggests biasing the Rapidly-exploring Random Trees (RRTs), with the outcome of a safety evaluation, which affects the probability of choosing a random point in the sampling phase of the RRT algorithm, to increase the chance of safer outcomes. SB-RRT considers safety of the path it is planning in the expansion phase of the RRT tree with considerations in the value C, representing the cost of the chosen state.

A value of safety, S, is calculated for every state as the distance between the state and the nearest obstacle. Hence states near obstacles have a lower value of safety. On the other hand, further states from the obstacles, which are also safer, get a bigger value of safety. Another value, G, is also calculated for every state to determine its closeness to the goal:

$$G(x, g) = |g - x|$$

where x is the state being evaluated, and g is the goal state. This equation shows that G has an inversed relationship with the squared value of the distance to the goal state. The cost value, C, is calculated as the sum of these two values:

$$C = S + G$$

which represents both safety and closeness to the goal

```
Function SB_RRTPlan(env:environment,initial:state,goal:state):rrt-
tree
    var nearest,extended,target:state;
    var tree:rrt-tree;
    nearest := initial;
    rrt-tree := initial;
    while(Distance (nearest,goal) < threshold)
            target = ChooseTarget (goal);
            nearest = Nearest (tree,target);
            extended = Extend (env,nearest,target);
            if extended <> EmptyState then
                    p = UniformRandom in [0.0 .. 1.0];
        if 0 < p < SafetyProb and
                SafetyVal(nearest) < SafetyVal(extended) then
                    AddNode (tree,extended);
    return tree;
```

Table 1: the SB- RRT algorithm

Table 1 shows the SB-RRT algorithm. If the random state has a higher or equal value of C from the nearest node of the tree (parent node), the tree extends toward the state with the probability P and the node is added to the tree. If the state had a smaller value for C, it would be ignored and another random state will be analyzed. By implementing this algorithm the tree expands to a safer node in each step with probability P.

Results show with increasing the value of P, a safer path is obtained, but under some circumstances it makes the algorithm to fail to give an output in the desired time. On the other hand, lowering the value of P increases the probability of obtaining a result, but reduces the clearance of the trajectory nodes and therefore the safety of the path. Consequently, to achieve a more generic planner, it is required to adjust the value of P based on field specifications' comment on acceptable safety levels of the path. But since in an adversary world like the SSL field the environment is constantly and rapidly changing, choosing a constant value for P seems challenging and rather impossible.

In this method, the SB-RRT is executed several times with descending values of P, and the first feasible planned trajectory is chosen as the outcome of the algorithm which could be considered as the safest path amongst computed paths. This method results the algorithm to have a high overhead, so a revised approach to overcome this load of work is presented in the following.

This algorithm makes the RRT executable on GPU which is based on a massively parallel architecture [15]. Before the randomized planner initiates execution, the values that the algorithm is about to use frequently, such as field evaluation are all calculated and stored in the GPU memory to avoid calculation redundancy overhead. And so is for the random values. As mentioned before, the SB-RRT needs lots of random values like the random state in the field or the one that determines whether to move towards a safer state or not. This random Look up Table (LUT) is randomized every time the algorithm is run, to ensure random values distributed fortuitous.

Having all required values, the algorithm is executed simultaneously on cores of the GPU. To obtain the safest trajectory, the SBRRT algorithm is executed with the P parameter set to 1. In case this value had no result, the algorithm would be executed with smaller values for P. this procedure continues until one of the threads obtains an obstacle free path to the destination.

The algorithm is also altered to optimize the execution speed. For instance, since all cores of one warp in GPU execute the same instruction and execution time prolongation of a single core in a warp stops the other cores functionality, the algorithm is rewrote in a way to possess the least number of loops to minimize the execution time. The maximum extend tries of the RRT instances are also limited to avoid pause of other cores functionality, when having an ―unlucky‖ expands in a tree and force the program to terminate. Figure 1 illustrates a potential instance of this phenomenon: two independent forward-search RRTs planning the same query (a source – the left dot and a destination – the right dot) are almost on the same region in the progression but with much more difference in their planned path. The path on the left as it is called the "lucky" one, is shorter in terms of extension; hence, done in less iterations than the "unlucky" one on the right (166 versus 1887 iterations). With continuing growth of the tree, it is possible that this ―unlucky‖ tree will require more time to complete than the entire instance of the ―lucky‖ one [16].

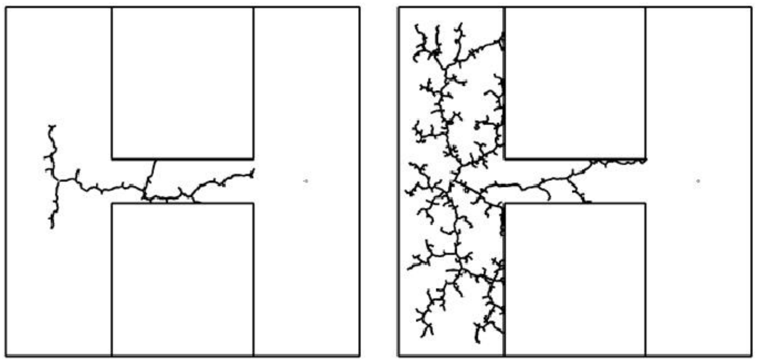

Figure 4: Lucky and Unlucky Instances of an RRT

Each execution delivers the planned path, in case of existence, to the video memory as an array of waypoints to be later transferred to the main memory. The Central Processing Unit (CPU) evaluates each planned path based on its defined evaluation methods and chooses the best as the results.

The following section includes execution times for both CPU and GPU. The CPU used for tests is an Intel™ Core2® Quad 9450 and the GPU is an NVidia™ GTX570.

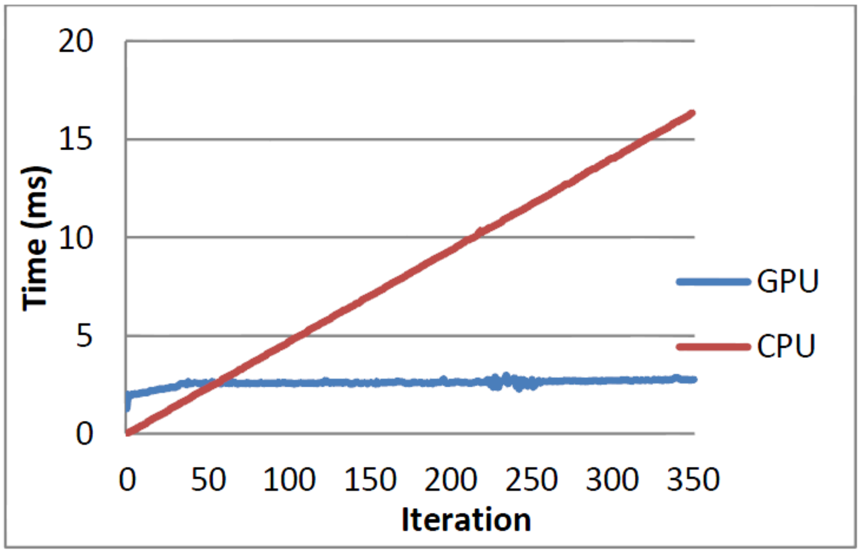

Figure 5: Execution time vs. Iteration numbers on CPU and GPU 4

Figure 5 illustrates an execution time comparison between GPU and CPU in several runs of SBRRT with different numbers of iterations. The horizontal axis shows the number of simultaneous iterations, whereas the vertical axis is the execution time of that run in milliseconds. As illustrated in figure 5, heavy loads of threads on the GPU takes less time due to its Massively-Parallel-Architecture than CPU after a certain point. On the other hand, by running the SBRRT on GPU, the CPU resources would be freed up for other simultaneous AI tasks or desired algorithms. The following section includes experimental results from a SSL robot platform. In each test, 10 obstacles are placed in the field in different patterns and the robot should cross the field from a constant initial state to a preset destination state without having any collision or glitch with obstacles.

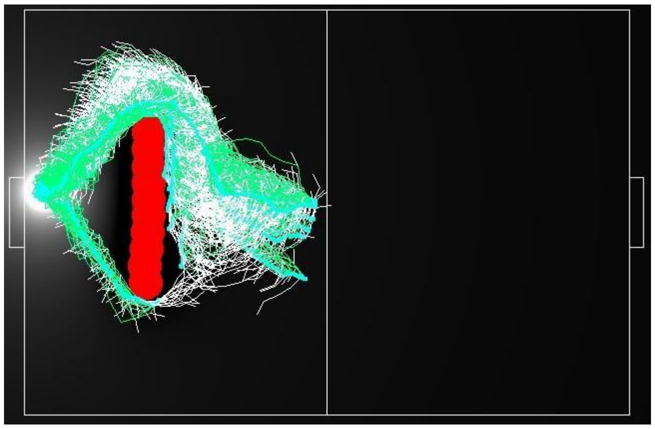

Figure 6: Planned SB-RRT with value %99.8 for P

The results of each experiment are recorded from 50 rounds of execution. Each experiment is also run three times with different values for probability P.

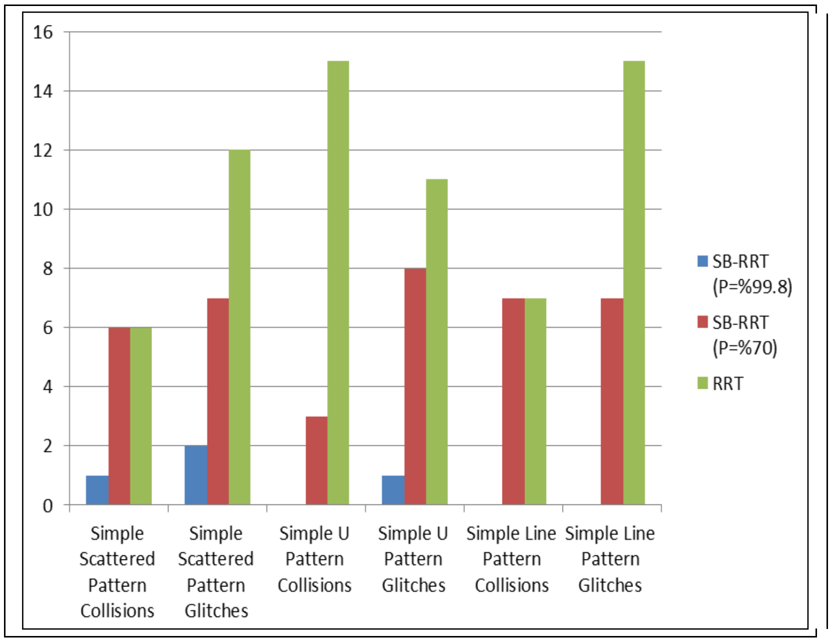

Figure 7: Number of collisions and glitches of planned SB-RRT with different values for P

As illustrated in figure 7, in all experiments, SB-RRT has less collision and glitch. Also, it is clear that increasing the value of P reduces total number of collisions and glitches, but decreases the probability of obtaining a result in desired time (control period C). Consequently, there is a tradeoff between obtaining a result in desired time (C) and probability of having collision. However, in domains that existence of a solution is an asset and having glitch is tolerable (like soccer robot), a smaller P value would result in a safer and more reliable path compared to that of pure RRT.

References

[1] Zickler, S., Veloso, M. Playing Creative Soccer: Randomized Behavioral Kinodynamic Planning of Robot Tactics, Pennsylvania: Carnegie Mellon University, 2008
[2] Bruce, J. Real-time Machine Vision Perception and Prediction, Pennsylvania: Carnegie Mellon University, May 2000
[3] Marr. D. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. W. H. Reeman and Company, San Francisco, 1982

[4] Jain R., Kasturi R., and Schunck, B.G. Machine Vision. McGraw-Hill, 1995
[5] Zickler, S. Bruce, J. Biswas, J. Licitra, M. and Veloso, M. CMDragons 2009 Extended Team Description, Pennsylvania: Carnegie Mellon University, 2009
[6] Bruce, J. and Veloso, M. Real-Time Randomized Path Planning For Robot Navigation, Pennsylvania: Carnegie Mellon University, 2008
[7] Barraquand, J. Langlois, B. and Latombe, J. Numerical Potential Field Techniques for Robot Path Planning, Pennsylvania: Carnegie Mellon University, 1992
[8] Browning, B. Bruce, J. Bowling, M. and Veloso, M. STP: Skills tactics and plans for multi-robot control in adversarial environments. In: Journal of System and Control Engineering, 2005
[9] nVidia web page: http://www.nvidia.com
[10] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. International Journal of Robotics Research, 20(5):378–400, May 2001.
[11] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J.P. How. Realtime motion planning with applications to autonomous urban driving. IEEE Transactions on Control Systems, 17(5):1105–1118, 2009.
[12] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In IEEE Conference on Roboticsand Automation, 1991.
[13] Browning, B. Bruce, J. Bowling, M. and Veloso, M. STP: Skills tactics and plans for multi-robot control in adversarial environments. In: Journal of System and Control Engineering, 2005
[14] http://small-size.informatik.uni-bremen.de, About the Small Size League
[15] NVIDIA CUDA Compute Unified Device Architecture Programming Guide, V. 1.0, 06/01/2007.
[16] Nathan A. Wedge, and Michael S. Branicky On Heavy-tailed Runtimes and Restarts in Rapidly-exploring Random Trees
[17] A. Salehi, M. R. Niknezhad, and E. Kamali ―A Novel Approach to Real-Time Processing: Implementing Complex Calculations on GPU‖ 2nd Iran Open Symposium, April, 2010.
[18] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In Proceedings of the IEEE/RSJ International Conference onRobotics and Systems (IROS), 2003.
[19] Barraquand, J. Langlois, B. and Latombe, J. Numerical Potential Field Techniques for Robot Path Planning, Pennsylvania: Carnegie Mellon University, 1992
[20] S. Karaman, E. Frazzoli, ―Incremental Sampling-based Algorithms for Optimal Motion Planning‖ May 2010
[21] Y. Liu and N.I. Badler. Real-time reach planning for animated characters using hardware acceleration. In IEEE International Conference on Computer Animation and Social Characters, pages 86–93, 2003.