Unpublished

Generating Motion Patterns Using Evolutionary Computation in Digital Soccer

Masood Amoozgar, Daniel Khashabi, Milad Heydarian, Mohammad Nokhbeh

***Abstract*— Dribbling an opponent player in digital soccer environment is an important practical problem in motion planning. It has special complexities which can be generalized to most important problems in other similar Multi Agent Systems. In this paper, we propose a hybrid computational geometry and evolutionary computation approach for generating motion trajectories to avoid a mobile obstacle. In this case an opponent agent is not only an obstacle but also one who tries to harden dribbling procedure. One characteristic of this approach is reducing process cost of online stage by transferring it to offline stage which causes increment in agents' performance. This approach breaks the problem into two offline and online stages. During offline stage the goal is to find desired trajectory using evolutionary computation and saving it as a trajectory plan. A trajectory plan consists of nodes which approximate information of each trajectory plan. In online stage, a linear interpolation along with Delaunay triangulation in xy-plan is applied to trajectory plan to retrieve desired action.**

I. Introduction

THE research in distributed Multi Agent Systems and Artificial Intelligence and their applications is one of common fields of study in recent years. A major focus of machine learning research is to learn and imitate complex patterns and make intelligent decisions based on sampled data. Some machine learning systems attempt to eliminate the need for human-intuition in data analysis, while others adopt a collaborative approach between human-intuition and machine. Human-intuition cannot, however, be entirely eliminated, since the system's designer must specify how the data is going to be represented and what mechanisms will be used to search for a characterization of the data. One other reason for collaborative human-machine approach is due to complexity and difficulty of a pure machine analysis. In this study we are going to use a collaborative human-machine approach for designing a motion trajectory. Regarding to the fact that the way a mobile agent treating is somehow unpredictable before online stage, human's intuition can be so effective for generating an initial dribbling behavior.

In this paper as a complex problem, analyzing the avoidance of a mobile obstacle is focused on. Obviously it is comparable with traditional obstacle avoidance problem. The manner of obstacle in which it behaves has enormous effect on how the algorithm must be designed. The most important concern in this paper is the case in which obstacle agent tries to harden motion task. The most glorious example of this problem is encountered in soccer environment, where a ball-holder agent wants to dribble (avoid) a (several) defender agent(s), simultaneously reach a specific position or dribble toward specific direction.

In this paper, we propose a hybrid computational geometry and evolutionary computation approach for generating motion trajectories to avoid a mobile obstacle. Offline stage is when the agent is out of competition, hence there is no strong limitation on competition time during this step. So it tries to explore for better behaviors. In online stage, an agent must exploit what it has learnt before. In fact it is the time when it tries to act in desired manner, so online stage is limited to whole game time that there is a limitation of process in this stage. Transferring the process of calculating the desired motion to offline-time results in reducing computational cost of online which causes increment in agents' performance. In offline stage the goal is to find desired trajectory plan using evolutionary computation. A trajectory plan consists of nodes which approximate information of each trajectory plan. In online stage a linear interpolation along with Delaunay triangulation in xy-plan is applied to trajectory plan to get desired action.

In the rest of the paper, we first present some issues on dribbling a mobile agent. Then offline stage and online stage parts of method are discussed. At the end of the paper, several results of proposed method are shown. Software is used for designing motion trajectories which is called Dribble Editor (DEdit in brief). By using this software it is possible to design motion patterns and simulate their results. As it will be mentioned in detail in the rest of paper, one can apply GA to any dribble trajectory using equipments deserved in DEdit. More pictures are depicted in (Fig. 5). The latest version of software which was designed to implement the method (DEdit) and its source codes (in Java) and all information about it, are available on this address:

`http://cs.sharif.edu/~amoozgar/focus/dedit`

II. Dribbling a Mobile Agent

Dribbling is a sequence of several decisions which they overall determine an agent's position on a plan. So every decision by changing agent's relative position to other agents would make a significant change in agent's next performance. Therefore it is a complicated task.

As mentioned, the center of attention is on special case of dribbling a mobile agent/obstacle. In traditional obstacle avoidance, the main focus is allocated to minimize time

Masood Amoozgar, and Milad Heydarian are with Computer Science Department, Sharif University of Technology. Email: {masood.moozgar,heydariaan}@cs.sharif.ir. Daniel Khashabi and Mohammad Nokhbeh are with Electrical Engineering Department, Amirkabir University of Technology. Email: {d.khashabi, nokhbeh100}@gmail.com.

needed to achieve one specific destination meanwhile considering algorithm's complexity and time cost. In some cases environmental position of an obstacle changes dynamically and algorithms must be as adaptive as possible with real-time environment. But the way that obstacles behave is so important. In fact if rate of intelligence for an obstacle agent is low, the obstacle agent won't do its job to grab the ball, hence it's a conventional obstacle avoidance problem and with less time and energy cost, target of dribbling agent will be achieved. On the other hand if it is high, it will try to grab dribbler agent's ball. So the ball-holder agent must avoid defender at least with a distance a little more than kickable distance of obstacle agent, called safety distance. Obstacle agent could be more intelligent and learn the way dribbler is acting. So the actions of dribbler agent must be adjusted greedier.

III. OFFLINE STAGE

A. Trajectory plan Generation

For generating a dribble trajectory we need to make an structure for defining a dribble pattern. In fact, this structure is a function which gives information about dribble in every position of the field. The optimization process which will be discussed in further parts will be applied to this structure. In defining such a structure we have used Delaunay triangulation and linear interpolation which is going to be described in next parts.

1) Delaunay triangulation: Delaunay triangulation is one of the most promising methods to triangulate the plate region, based on a given point-set. Delaunay triangulation for a point-set $\boldsymbol{P}$ is *DT* ($\boldsymbol{P}$) such that there is no point member of $\boldsymbol{P}$, inside circum circle of any triangle member of *DT*($\boldsymbol{P}$). For every set of points more than 3, there is a unique Delaunay triangulation. In our data extraction method, vertices of triangles are agent's relative position to obstacle agent. For every vertex (position in xy-plan), information of dribbler agent is generated, e.g. direction of motion, direction of ball and acceleration of motion.

2) Linear Interpolation: For finding information of specific point P_s inside of triangle T_n (Fig. 1) we use simple linear interpolation. If P_a , P_b and P_c are assumed as vertices of triangle, each containing information of I_a , I_b and I_c , we can simply use a weighted average for getting information of point I_s :

$$I_s = \frac{\frac{I_a}{|P_s, P_a|} + \frac{I_b}{|P_s, P_b|} + \frac{I_C}{|P_s, P_c|}}{\frac{1}{|P_s, P_a|} + \frac{1}{|P_s, P_b|} + \frac{1}{|P_s, P_c|}} \quad (1)$$

The formula (1) can be considered as a conventional weighted mean of three values where $|P_i , P_j|$ is distance between two points P_i and P_j. This kind of function approximation using Delaunay triangulation has following advantages [1]:

--*High approximation accuracy:* It is claimed that this method of approximation is more accurate than other methods.

--*Locally adjustable:* Even if an existing data modified, the triangle region where the data is not in, is not affected.

--*Simple and fast*: Regarding to time complexity of triangulation (O*(nlogn)* see [2]) and linear interpolation, it is possible to use it in a real-time environment.

There are some other works that use combination of Delaunay triangulation and linear interpolation as a function approximation. In [1] and [3] a method is proposed to approximate positioning of a multi-agent system based on combination of Delaunay triangulation and linear interpolation. It's claimed that this method is fast, accurate and simple.

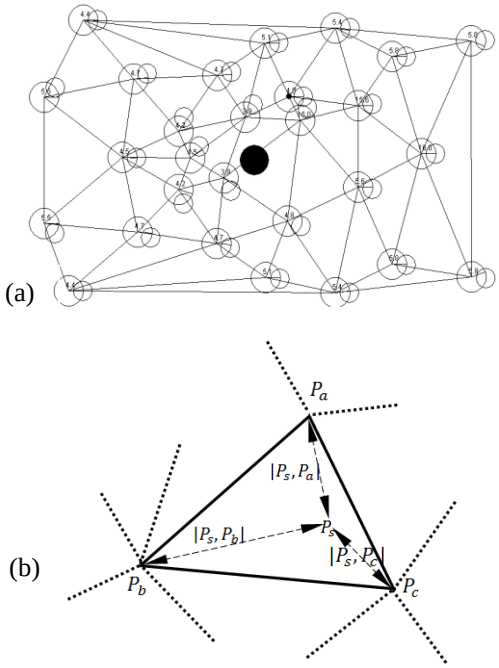

Fig. 1. (a) A Delaunay Triangulation for 27 point-set. (b) Linear interpolation in one triangle for retrieving information of a specific inner point e.g. P_s . The black player is assumed to be opponent player. White Player is assumed a dribbler agent.

B. Designing trajectory plans

Human intuition used in trajectory plan design can be so effective. One can design his desired trajectory (Fig. 5). In defining every pattern, one can define these specifications:

--Position of an agent relative to opponent agent's position.

--Direction of agent's movement relative to field's global zero direction.

--Direction of ball relative to field's global zero direction.

--Acceleration of agent in specified direction.

After inserting every new node to a trajectory plan, DEdit triangulates plan. It is possible to edit specifications of all predefined nodes.

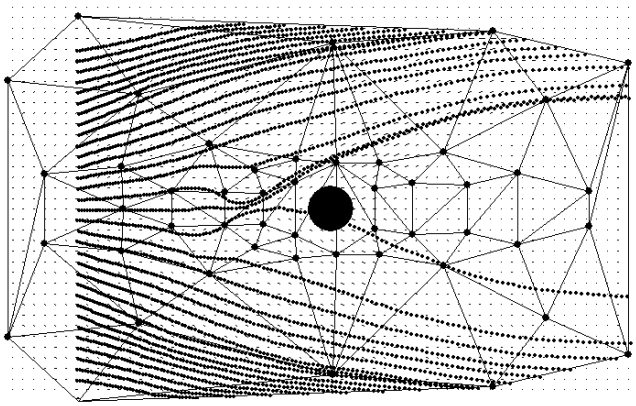

Fig. 2. Primary trajectory plan designed by human in Dribble Editor and simulated trajectories. Every simulation starts from x = -12 with initial velocity of 4 m/s.

For testing a trajectory plan, a simulation engine is devised which simulates agent's movement when starting from a specific initial position and with a specific initial velocity given by user. One example of a trajectory generated by user is shown in (Fig.2).

C. Optimizing strategy of trajectory plan

For using Genetic Algorithm, the problem requires to be defined as a generation and then it is possible to use evolution over this generation. The first stage is to define individuals. Each trajectory plan is defined as an individual. Practically, there exist complex of nodes in each individual that every node includes 5 parameters: *x*, *y*, *acceleration*, *ball-direction* and *body-direction*. Parameters *x* and *y* are just for specifying each point from others and are not used in evolution process.

There is a process of generating individuals based on human-generated trajectory plan. This is done until reaching a population in number of *population-size* which will be used as initial population. The main process of GA will be repeated in *generation-count* times. In next parts, steps of evolution process is described:

[Step 1] Selecting several individuals as parents and copying them into mating pool: Parent selection basically should better be a random process for diverging individuals in search space. Moreover, better individuals (which are ranked by their fitness value) should gain more chance to be selected. Therefore, four algorithms are assumed:

-- Roulette Wheel
-- Rank
-- Stochastic Universal Sampling
-- Tournament

A parameter called *parent-selection-probability* controls the selection process.

[Step 2] Applying crossover on individuals: The crossover process consists of selecting two random parents and making two new individuals using parents. This process is repeated over and over until reaching the desired quantity. To control crossover process, a parameter *crossover-probability* is applied.

When two random individuals $P_i(g^{t-1})$ and $P_j(g^{t-1})$ are chosen from previous generation parent group $P(g^{t-1})$, two offspring $P_i(g^{t-1})^{child}$ and $P_j(g^{t-1})^{child}$ are generated by the crossover. This is carried out using the weighted mean, so k^{th} parameter of offspring is:

$$P_i(g^{t-1})^{child}.param_k = \frac{w_i \times P_i(g^{t-1}).param_k + w_j \times P_j(g^{t-1}).param_k}{w_i + w_j} \quad (2)$$

$$P_j(g^{t-1})^{child}.param_k = \frac{w_j \times P_i(g^{t-1}).param_k + w_i \times P_j(g^{t-1}).param_k}{w_i + w_j} \quad (3)$$

The parameters w_1 and w_2 are two random numbers between *0.8* and *1.2*.

[Step 3] Mutating on population and creating new individuals and adding them into population: Mutation process is defined by adding a Gaussian Random Number (*GRN*) to all parameters of each individual. These small random numbers are limited to *1%* of maximum value of the parameters, e.g. for *body-direction* and *ball-direction*, maximum value is 180° and corresponding *GNR* is a number between *-1.8* and *1.8*. In order to mutate individuals, an *initial mutation coefficient* is applied to create first generation with an individual by mutation, so *mutation-coefficient* is multiplied by the small random number to make difference between primary mutation (which was applied on first individual to generate first population) and recent mutation. If $P_i(g^{t-1})^{mutated}$ is a mutated individual, the value of k^{th} parameter of individual under mutation will be:

$$P_i(g^{t-1})^{mutated}.param_k = P_i(g^{t-1}).param_k + \left(\frac{\max\left[P(g^{t-1}).param_k \right] \times GRN \times mutationCoefficient}{100} \right) \quad (4)$$

[Step 4] Selecting individuals from the population and transferring them to next generation: Selection of the individuals is a deterministic process i.e. transferring best *generation-number* of individuals to next generation. This is done after evaluating individuals by using a fitness function. Hence, this process results in maximizing fitness value of best individual, which leads to optimize the behavior.

In next part, structure of proposed fitness function is discussed.

D. Fitness function

Design of a fitness function, lets us find a desired behavior plan in every position and every condition of plan. It is supposed that positions of nodes in all trajectory plans are the same. So the fitness function of a trajectory plan *tp* is sum of fitness function of the nodes.

$$planFitness(tp) = \sum_{i=1}^{\#nodes} fitnessFunction(tp.node_i) \quad (5)$$

And fitness function for every node in trajectory plan is:

$$fitnessFunction(node) = \exp(desiredFunction(node)/\pi^2) \quad (6)$$

$$desiredFunction(node) = \alpha(node.bodyDir)^2 - (node.bodyDir - node.bodyRlObs)^2 - \beta(node.ballDir)^2 - (node.ballRlObs - \pi)^2 \quad (7)$$

$$\alpha = \alpha_{user}\left(0.1 + 50\exp(-2dist)\right)$$
$$\beta = \beta_{user}\left(0.1 + 50\exp(-2dist)\right) \quad (8)$$

Where in (7) *bodyDir* and *ballDir* are direction of agent's body and direction of ball relative to global zero direction, respectively. *ballRlObs* is the angle between ball and obstacle agent. *bodyRlObs* is the angle between dribbler agent and obstacle. Two factors are considered in designing this fitness function. The first is considering variables which will cause in low outcome i.e. straight motion and the other one, variables which causes in generating safe motion for dribbler agent. These factors behave against each other. This is like what happens in real football game. Direct movement is the fastest and low cost course and increases excellence of ball control whereas moving spirally increases safety of motion and of course motion's energy and time cost. Therefore, balancing effectiveness of these factors in fitness function, will lead to desired motion.

In order to adjust values of *bodyDir* and *ballDir* to desired ones, two factors are included for each of variables. First factor maximizes each variable in the function and the other one minimizes it. Rate of effectiveness for each of factors, depends on their coefficients (α and β). Meanwhile α_{user} and β_{user} are coefficients, which are applied by user. Increment of α encourages agent to move smoothly which results in wasting less stamina and increment of β encourages agent to keep ball in front of its body which results in maximum control on it, and decrement of α and β makes agent more sensitive to the opponent. For achieving a more similar behavior to the real agent, parameters α and β should be dependent on distance between agent and obstacle (parameter *dist* in (8)).

IV. Online Stage

A. Using calculated values

In online stage, dribbling action is done based on generated pattern. Depending on the position, in any cycle a trajectory plan determines agent's acceleration, body-direction and ball-direction using linear interpolation. The reasons that linear interpolation is proposed as a default algorithm for calculating values of agent's movement action is discussed e.g. local approximation and low computational cost. One can use other interpolating methods especially nonlinear approximators (e.g. Neural Networks) depending on desired complexity

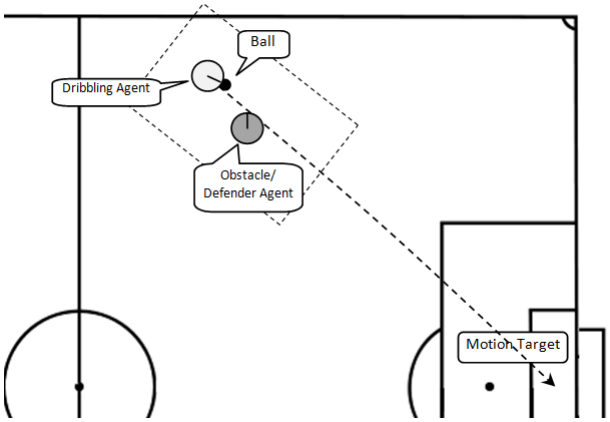

Fig. 3. Using a trajectory plan regarding to the position of obstacle .It depends on direction of obstacle relative to opponent's goal.

Trajectory plans are obstacle-centric, which means position and direction of any trajectory plan in the digital soccer field, depends on obstacle (Fig. 3). Therefore, in order to acquire an algorithm for choosing the best trajectory plan for agent's motion, Delaunay triangulation is used again. But in this layer of processing, nodes are assumed to be obstacles. In fact the trajectory plan in every position of field is calculated based on what is explained before. This is because of the fact that in different positions of field, different manners of dribbling is desirable, e.g. in there exist a region in field that more safely action is needed while in other region of field low-cost and rapid action is desirable. This will cause in dynamic manner of motion based in where the agent is acting. By encountering any obstacle, desired trajectory plan will be retrieved by applying linear interpolation on Delaunay mesh of soccer field based on saved trajectory plans which is calculated in offline stage for various obstacle positions.

V. Results & Future Works

In this study we proposed a method for designing motion trajectories where the goal is to dribble a moving agent. As mentioned a special case that is most concerned is where a ball-holder agent wants to dribble a defender agent. The proposed method is a Genetic Algorithm which is applied on solutions that are saved using Delaunay Triangulation. Even though there is no strong proof on optimality of this trajectory generation, the results depicted in previous parts show that by adjusting fitness function and the use of offline dribble data-set, it is possible to achieve trustable solutions. There are some other points in this work:

--Delaunay triangulation can be used as a function approximator where there is a need for local and smooth approximation. Here we showed its usefulness for saving motion patterns.

--Even though dribbling is a sequential action, proposed method is free from sequential analysis. It is referring to the fact that defining final results for dribbling is so hard. So it

is too difficult to apply methods like Reinforcement Learning to solve these kinds of problems. Results of generating desired patterns are shown in (Fig. **4**), (Fig. **6**) and (Fig. **7**)

TABLE I.
PARAMETERS OF GENETIC ALGORITHM

Parent selection algorithm	Crossover probability	Mutation coefficient	Parent selection coefficient	Generation count
Roulette wheel	0.8	4	0.6	1000

Max acceleration	Range of *bodyDir*	Range of *ballDir*	α_{user}	β_{user}
3 [m/s^2]	$[-\frac{\pi}{2},\frac{\pi}{2}]$	$[-\frac{\pi}{2},\frac{\pi}{2}]$	0.66	0.33

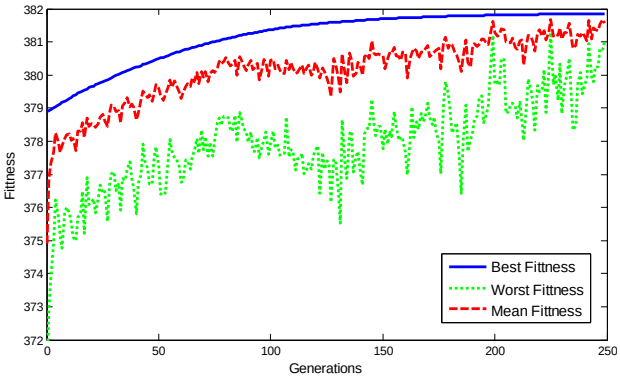

Fig. 4. Optimizing trajectory plans using EA. Best, mean and worst of fitness values in every generation during optimizing with parameters in TABLE I. Values are averages of 8 times running.

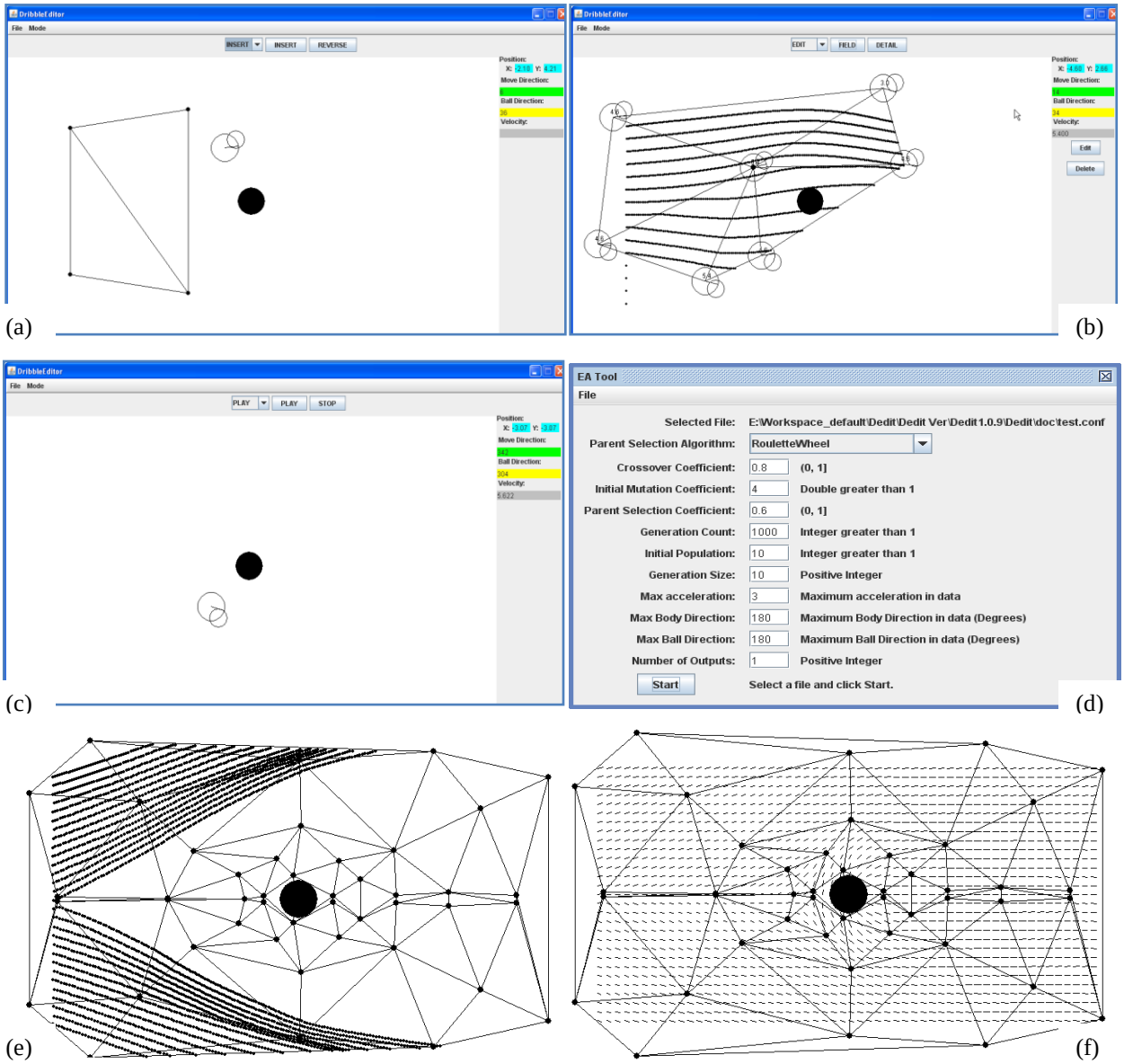

Fig. 5. (a) An screenshot software used for generating motion patterns in INSERT mode (b) EDIT mode (c) PLAY mode. (d) adjusting GA parameters. This application is designed to create, edit and test the trajectory plans. Every trajectory plan contains nodes which they consists of several independent variables: position(x , y), acceleration, body-direction and ball-direction (e) Resulted motion trajectories when agent started from different initial positions in the field (f) Direction and acceleration in every position of field. It is similar to an electric field!

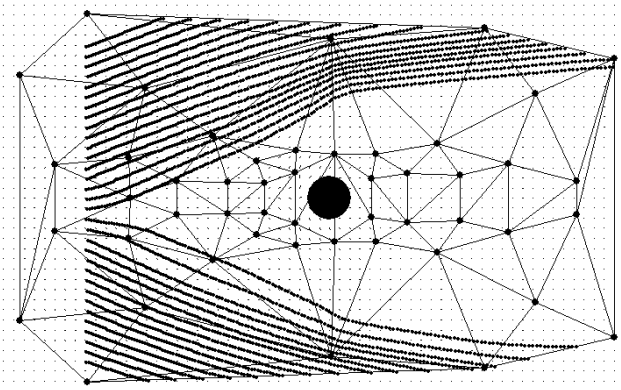

Fig. 6. Trajectory-plan and optimal trajectories after optimizing trajectory plan in Fig. 2, optimizing parameters came in TABLE I.

Getting more optimal results is what should be considered for future works. Online stage adaptive behavior is an important issue, where the dribbler agent changes its behavior based on its failures and successes. Because agent needs to learn in real-time, computational load of this updating is so important. In general, the real world problems are dynamic and basis of action selection is to deceive the opponent. The performing algorithm must be probabilistic, regarding to reactions of the opponents. In advance steps, as the defense goes along legislating attacker's behavior, the reactions must be changed. So we are encountered with a dynamic action.

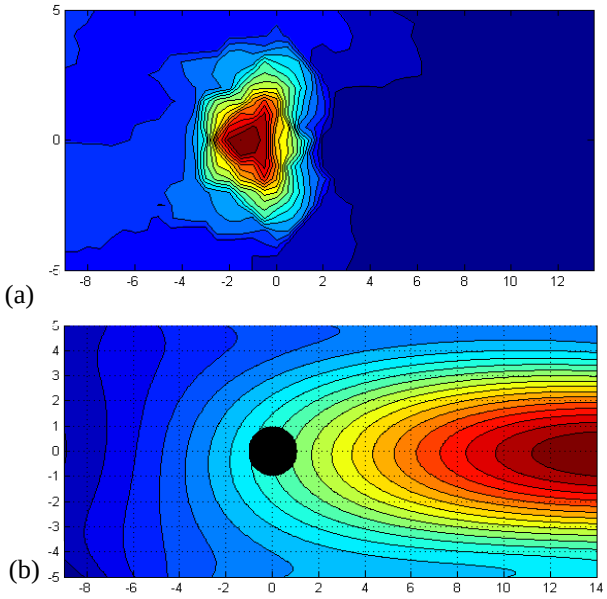

Fig. 7. (a) This figure shows distribution of acceleration in xy-plane. This picture is inferred from Fig.5 which is result of Genetic algorithm on motion field. In our implementation acceleration of each node is proportional to changes in velocity direction. As depicted acceleration near opponent has highest value. (b) Shows velocity distribution through the dribble field for a common initial speed for all initial position (x~-9). As depicted speed behind opponent player is increasing.

ACKNOWLEDGMENT

The authors would like to thank Mojtaba Khalidji for his comments and helpful discussions.

REFERENCES

[1] H. Akiyama and I. Noda, "Multi-agent Positioning Mechanism in the Dynamic Environment," *Robocup2007,LNA 5001*, vol. 5001/2008, pp. 377-384, 2008.

[2] M.V.Kreveld, M.D.Berge, O.Schwarzkopf, and M.Overmars, *Computational Geometry: Algorithms and Applications*, 2nd ed.: Springer, 2000.

[3] H.Akiyama, D.Katagami, and K.Nitta, "Training of Agent Positioning Using Human's Instruction," *Journal of Advanced Computational Intelligence and Intelligant Informatics*, vol. 11, no. 8, pp. 998-1006, 2007.

[4] L. D. Davis and Melanie Mitchell, *Handbook of Genetic Algorithms*.: Van Nostrand Reinhold Company, 1991.