

# Potential-Based Processing for Action-Game Experimentation

Russ Abbott

Department of Computer Science  
California State University, Los Angeles  
Los Angeles, California  
Email: RAbbott@CalStateLA.edu

Valentino Crespi

Department of Computer Science  
California State University, Los Angeles  
Los Angeles, California  
Email: vcrespi@CalStateLA.edu

**Abstract**— We define a framework for what we call *potential-based computing* and show how it may be applied to action-motion applications in general and dynamic games in particular. We also describe how the framework lends itself to various forms of evolutionary computing.

## I. INTRODUCTION

Agent-based systems have become today an important technology for the development of complex and scalable distributed systems [1].

In this paper we propose an agent-based billiard-ball soccer game where agents are designed to move in an artificial force field. In particular we have built a game software infrastructure where competing players can design their agent (**Player** class) based on determining convenient artificial potential functions and evolutionary/learning algorithms to adapt them over time.

In this framework the *engine* is responsible for enforcing the rules of the game. This implies that, for example, the acceleration and velocity of the players remain bounded regardless of what is computed locally by each agent’s controller.

Each player on a team is controlled by the same program, i.e., is an instance of the same **Player** class. There are no specialized classes of players for different positions on a team.

The primary constraint is that player movement is based solely on conservative forces whose potentials the players *choose* to recognize in their environment. In other words, player velocities may be modified by accelerations, but that is the only effect a player’s program can have on a player’s motion. This leads to a very natural-looking result as the players glide around the field. (Actually, it looks more like ice hockey than soccer.) Players may decide which forces to feel based on information supplied to them. For example, players can determine if they are closest on their team to the ball or if they are the closest to the ball of all players. Players also know, for example, if they are the slowest player on their team. Such slowest players respond to forces that goalies would feel.

This structure offers a simple but interesting framework for action-game experimentation. If programming the players explicitly, the issues are what are the forces one wants to define (e.g., attraction to the ball, etc.) and what information (which player is closest to the ball, etc.) should be computed and made available to the players.

If one wants to experiment with learning, these same issues can be parameterized for genetic algorithms, genetic programming, and neural net learning.

This paper describes the strategy used to define the current players, and it outlines a framework, both explicitly programmed and learning, within which competing players may be produced.

## II. POTENTIAL-BASED COMPUTING

Hill climbing, of course, has been a mainstay of all forms of intelligent computing. Potential-based computing is in some sense a form of hill climbing in which the gradients are obtained by differentiating scalar (artificial) potential functions. Artificial potential functions have themselves been a mainstay of engineering as they have been widely employed to solve problems of robot navigation [2]. The method is based on designing a control law that drives the robot along trajectories obtained by solving

$$\dot{x} = -\nabla V(x) , \quad (1)$$

for an appropriate *artificial potential function*  $V$  that is globally minimized at the intended destination point and diverges as the robot moves towards an obstacle. The expected behavior of the robot is roughly a motion from high potential states to low potential states similar to that of an electric charge under the effects of some kind of electrostatic field [3], [4]. Research in this field focused on the identification of classes of potential functions that do not possess local minima so that their minimization will not lead moving robots to stop in undesired locations. For example, Rimon and Koditschek [5] devised a method to build “good” navigation functions in the case of a single robot moving in a perfectly known static environment. Alternatively, Connolly et al. [6] and Decuyper and Keymeulen [7] and also Kim and Khosla [8] investigated the use of Harmonic potential functions that solve the Laplace equation:  $\nabla^2 V = 0$ . In their approach the potential is interpreted as the *velocity potential* of an irrotational and incompressible steady fluid. In this view, the robot moves like a unit of fluid in ideal conditions along streamlines. Decuyper and Keymeulen investigated also vector fields that are not necessarily gradients of a potential function [9].

Applications of artificial potential functions to agent-based systems have been proposed by Crespi et al. [10]–[12]. For example, in [10], [11] they investigated problems of *coordinated navigation* of many robotic autonomous agents in 2D and 3D using only local information.

Potential functions have been studied also in connection with *game theory* as shown in the work of Cesa-Bianchi and Lugosi [13].

Recently, alternative applications of artificial fields in vehicle formation and sensor coverage were investigated by Murray [14] and Poduri and Sukhatme [15].

In this paper we describe an approach to controlling agents based on potentials that they experience in their environment. However these potentials are not *velocity potentials*, as in robot path and trajectory planning but, as in classical mechanics, they determine forces that cause agents to accelerate or decelerate.

In our soccer application, the potentials are not assumed to be static. The potentials that exist and the potentials that an agent may experience may differ from moment to moment. Moreover our agents have global visibility of the system meaning that they can access global information when computing their gradients.

### III. POTENTIAL-BASED SOCCER

In our soccer game, player motion is controlled strictly by (dynamic) potentials that the players experience in their environments.

An abstract **Player** class is defined in which each soccer player is specified by three parameters: **location**:  $\langle x, y \rangle$  (a pair of doubles), **velocity**:  $\langle x, y \rangle$  (a pair of doubles), and a scalar **maxSpeed** (a double).

The soccer game is implemented as a MASON [16] simulation and is controlled by the MASON scheduler. MASON is a discrete event simulator; it schedules events on an event queue. In this case, however, we use the MASON scheduler simply to schedule events at unit intervals.

At each time step, each **Player**'s `step()`<sup>1</sup> method is executed. That method calculates an acceleration to be applied to the **Player**'s velocity, which is then applied to the **Player**'s location. Thus the essence of each **Player**'s `step()` method is the following:

```

accel = getAcceleration();
if ( accel.magnitude() > 1 )
    accel.setMagnitude( 1 );
velocity = velocity.add( accel );
if ( velocity.magnitude() > maxSpeed )
    velocity.setMagnitude( maxSpeed );
location.add( velocity );

```

Fig. 1.

Acceleration is limited to a unit acceleration rate. Velocity is limited to a **Player**'s maximum speed.

<sup>1</sup>Anything scheduled on the event queue must implement a `step()` method.

A team consists of players instantiated from a single subclass of the **Player** class. Each such subclass is required to implement the `getAcceleration()` method, which is abstract in the **Player** class. Thus, the only software control one has over a player's movement on the field is their acceleration. Although there is nothing in the code to enforce this constraint, the method `getAcceleration()` must be (is always) expressed as a *sum of forces* felt by a **Player**.

Currently we ignore mass, so forces may be expressed directly as accelerations. The **Acceleration** class, as a subclass of the **Vector** class, has a method that allows one to add accelerations to each other.

Thus `getAcceleration()` must be of the form described in Fig. 2.

```

Accel accel = new Accel();
if ( <condition1> )
    accel.add( getForce( <c1>, <loc1>, <pow1> ) );
if ( <condition2> )
    accel.add( getForce( <c2>, <loc2>, <pow2> ) );
if ( <condition3> )
    accel.add( getForce( <c3>, <loc3>, <pow3> ) );
...

```

Fig. 2.

The method `getForce()` takes a location and two doubles as parameters. In our current implementation it returns an acceleration vector that is directly proportional to the first parameter  $\langle c \rangle$  and inversely proportional to the distance between the calling object and the  $\langle loc \rangle$  parameter raised to the power specified by the second parameter  $\langle pow \rangle$ .

In other words, the current artificial field is of gravitational-like nature and derives from the following class of potential functions:

$$V_{c,p}(r) = \frac{c}{r^{p-1}}.$$

So, if  $\Delta \mathbf{x}$  is the vector from the calling object to  $loc$  and  $\|\Delta \mathbf{x}\|$  is its euclidean norm then `getForce(c, loc, p)` returns

$$-\nabla V_{c,p}(\|\Delta \mathbf{x}\|) = c \frac{\Delta \mathbf{x}}{\|\Delta \mathbf{x}\|^{p+1}}.$$

For a gravitational attraction (repulsive force), for example, the second parameter,  $p$ , should be 2 and the first parameter,  $c$ , should be negative.

Counterbalancing attractive and repulsive forces can be used effectively to position the players. Most visually striking is the combination of the following two forces, which are applied to the one player on each team that is closest to the ball. (The constraint on which players experience which forces is implemented by the guard  $\langle condition \rangle$ s in Fig. 2.)

- 1) *An attractive force* to a position on the side of the ball away from the opponent's goal that would direct the ball into the opponent's goal if kicked from that position. In the current implementation, this force is implemented with a  $\langle power \rangle$  parameter of 0 – the force does not

diminish with distance; the player is always attracted to this so-called *offensive location*.

- 2) A *repulsive force* away from the center of the ball. This force is felt only when the ball is in the path from the player's current position to the *offensive location* indicated in (1). The repulsive force is stronger than the attractive force, but it is implemented with a  $\langle power \rangle$  parameter of 2 – the force diminishes with the square of the distance.

The combination of these two forces leads the player experiencing them to move toward the offensive location but to swerve around the ball as necessary.

These definitions will be elaborated a bit as we proceed, but for now they provide the right intuition.

#### IV. FOUR PARADIGMS

Potential-based computing lends itself both to explicit programming and to a number of different learning paradigms.

##### A. Explicit Programming

It is a worthy challenge simply to program players to feel forces such as those described in the previous section. The primary issues, of course, are as follows:

- What points should identify the focus of forces? In the previous section we spoke of two points: the center of the ball and the position from which to kick a goal. These points are both dynamic; they change at each time step. Other points may be static, such as points that identify one's goal or the opponent's goal. When programming players explicitly, a primary task is to identify points with respect to which it is useful to define forces.
- What should the parameters be for forces defined with respect to those points? Should the forces be attractive or repulsive? Should they diminish with distance, and if so, how rapidly? What should the force constants be?
- Under what conditions should a player experience each of the defined forces? Defined forces need not be felt by every player at every time step. The guard  $\langle condition \rangle$ s allow one to specify when a force should be felt. What should those conditions be?

In the example described in the previous section, we indicated that the force impelling a player to the offensive position while moving it around the ball is felt only by the one player on each team that is closest to the ball. Whether or not any particular player satisfies that condition will change from time step to time step. (In addition, as indicated, the repulsive force is felt only if the ball is in the path between the player and *offensive location*.) These guard conditions allow players to “trade-off” duties in a natural way as play proceeds.

Another force defined in the current implementation is toward a position just outside the goal and between the goal and the ball. This *goalie* position is generally felt by the player closest to that position.

As indicated above, the primary challenge is the creative programming challenge of deciding what forces to define and when to apply them.

This is really similar to most other software development challenges – but with a somewhat different API. A *Player's* API consists of its ability to respond to forces. The developer's job is to decide what those forces should be and when, i.e., under what circumstances, they should be applied.

##### B. Genetic Algorithms

The previous approach may easily be expressed as a genetic algorithm problem. Once one defines a set of forces and a set of conditions, the genetic algorithm issue is: what are the parameters? The job of defining the forces and conditions is still the responsibility of the software developer. But simply defining forces and conditions leaves a lot unspecified. Forces have parameters, which may have a wide range of values, and conditions may be used as guards in the affirmative or in the negative or not at all. It would be an interesting experiment to set up an genetic algorithm system that optimized such a set of parameters.

The creative part, defining interesting forces (e.g., a force to a position from which kicking a goal is possible) and conditions (e.g., being the closest on one's team to the ball), is still left to the developer. The GA simply optimizes the use of the concepts the developer supplies.

##### C. Genetic Programming

It is easy to see a generalization from the genetic algorithm problem to a genetic programming problem. A genetic programming system would be given just the raw data points and required to come up with its own forces and conditions, i.e., to evolve its own `getAcceleration()` expression. The terminals available for use in that expression would consist of the positions and velocities of the ball and the players along with constants such as the height and width of the field, the locations of the goals, etc. The function set would include the standard mathematical functions generalized to operate on vectors along with the standard Boolean and relational operators. The output would be an acceleration vector to be computed and applied to a player at each time step.

##### D. Neural Nets

It is also easy to see how this problem could be expressed as one for neural nets. Like the genetic programming version, the inputs would be the positions and velocities of the ball and players, etc. the output would be an acceleration to be computed and applied to a player at each time step.

#### V. CHOREOGRAPHED PLAY

As currently conceived, each player is scheduled individually to run once each time step. That is, at each time step each player independently makes a determination about what acceleration should be applied to its own motion. There is no mechanism for players explicitly to coordinate with each other. An alternative approach would be to have only one object for

each team, e.g., a *coach* agent, scheduled to execute at each time step. The coach would compute accelerations for all of the players on its team.

Clearly such a coach agent could be defined to do no worse than independent players. A coach agent could simply run whatever code the independent agents would have run had they been independent. An interesting question is whether a coach agent could be defined that does significantly better – or even whether its additional complexity would make it difficult to define one that does as well.

## VI. ADDITIONAL CONTROL PARAMETERS

Two other parameters are available for controlling the players.

- 1) *How much force should be applied to the ball.* If a player is in contact with the ball, it is offered the opportunity to kick it. In particular, the ball's velocity is modified by a vector of magnitude  $m$  from the player's center to the ball's center. (We really are talking about billiard ball soccer. The ball and the players are simple circles.) The scalar value  $m$ , which must be non-negative and is limited to a fixed maximum, is determined by a call to a method in the player. The ball itself has a maximum velocity, which is then used to bound the resultant velocity.

Determining the amount (although not the direction) of the force applied to the ball can be critical in that it allows players to distinguish among dribbling, passing, or shooting the ball. Thus at each time step, if a player is in contact with the ball a second function that returns a **double** is called. That function may be programmed explicitly, or it may be evolved in any of the way described previously. The same information is available to is as is available to `getAcceleration()`.

- 2) *Determination of the players' speeds.* At a more global level, when the teams are created at the start of a game (and this happens only once), each team is allotted a total maximum speed, which it is free to divide among as many players (up to a reasonable maximum) as it wishes. Each team is allocated the same total maximum speed. In the current implementation, this usually results in teams of size two or three, but that is not a requirement. Again, the decision about how many players should exist on each time and how the teams allocates its speed allotment may be made either through explicit programming or through some evolutionary mechanism.

## VII. CONCLUSIONS AND FUTURE WORK

We have defined a framework for potential-based computing and shown how it may be applied to action-motion applications in general and dynamic games in particular. We also described how the framework lends itself to various forms of evolutionary computing.

This work will evolve into a real platform for competing agents. This will imply the formalization of a notion of scoring and victory. Competing agent designers will have

to devise implementations of a **Payer** interface that produce agents capable of cooperating (in a soccer match a player has to decide to whom to pass the ball and players follow schemas) towards a common goal (maximizing the score), and of adapting by learning the capabilities of the opponents.

The final game system will also be fully distributed. So the ability of players to learn the positions of any other component of the system will be limited and subjected to noise proportional to the euclidean distance. In other words if two players are very far from each other and one tries to cross a ball towards the other it is fair to expect the kick to be quite imprecise. But mathematically this means that the accuracy with which each player knows the positions of the other players should decrease with the distance. The rate of decrease captures exactly the notion of locality in a distributed system.

From a theoretical perspective we intend to investigate techniques to determine optimal potential functions and evolutionary strategies that characterize victory in this game.

## REFERENCES

- [1] M. Wooldridge, "Intelligent Agents," in *Gerhard Weiss, ed., Multiagent Systems*. MIT Press, 2000, pp. 27–77.
- [2] O. Khatib, "Real-time Obstacle Avoidance for Robot Manipulator and Mobile Robots," *International Journal on Robotics Research*, vol. 5, no. 1, 1986.
- [3] Y. K. Hwang and N. Ahuja, "Gross Motion Planning – A survey," *ACM Computing Surveys*, vol. 24, no. 3, pp. 219–291, Sept. 1992.
- [4] J. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
- [5] E. Rimon and D. E. Koditschek, "Exact Robot Navigation using Artificial Potential Functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, October 1992.
- [6] C. I. Connolly, J. B. Burns, and R. Weiss, "Path Planning using Laplace's Equation," in *Proceedings of the International Conference on Robotics and Automation, Cincinnati, OH.*, 1990, pp. 2102–2106.
- [7] J. Decuyper and D. Keymeulen, "A Reactive Robot Navigation System Based on a Fluid Dynamics Metaphor," in *Hans-Paul Schwefel and Reinhard Maenner, editors, Parallel Problem Solving from Nature, Berlin*. Springer-Verlag, 1990, pp. 378–386.
- [8] J. Kim and P. Khosla, "Real-time Obstacle Avoidance Using Harmonic Potential Functions," in *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA.*, 1991.
- [9] D. Keymeulen and J. Decuyper, "The Fluid Dynamics Applied to Mobile Robot Motion: The Stream Field Method," in *Proceedings of the International Conference on Robotics and Automation*, 1994, pp. 378–386.
- [10] V. Crespi, G. Cybenko, and D. Rus, "Decentralized Control and Agent-Based Systems in the Framework of the IRVS," *Darpa Task Program paper*. <http://actcomm.thayer.dartmouth.edu/task/>, Apr 2001.
- [11] V. Crespi, G. Cybenko, D. Rus, and M. Santini, "Decentralized Control for Coordinated flow of Multiagent Systems," in *Proceedings of the 2002 World Congress on Computational Intelligence, Honolulu, Hawaii*, May 2002.
- [12] V. Crespi and G. Cybenko, "Decentralized Algorithms for Sensor Registration," in *Proceedings of the 2003 International Joint Conference on Neural Networks (IJCNN2003), Portland, Oregon*, July 2003.
- [13] N. Cesa-Bianchi and G. Lugosi, "Potential-Based Algorithms in On-Line Prediction and Game Theory," in *Machine Learning*. Kluwer Academic Publishers, 2003, vol. 51, pp. 239–261.
- [14] J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," in *2002 IFAC World Congress*, 2002.
- [15] S. Poduri and G. S. Sukhatme, "Constrained coverage for mobile sensor networks," *To appear in IEEE International Conference on Robotics and Automation*, 2004.
- [16] S. Luke, et al., "Mason: A multi-agent simulator of neighborhoods," available online at <http://cs.gmu.edu/~eclab/projects/mason/>.