Modification of a Robocup Simulation System to Increase Realism

Jeremiah Hemphill

The University of Rochester
Computer Science Department
Rochester, NY 14627

May 2009

Abstract

The Robocup competition consists of designing and programming a team of robots to play soccer. A simulation system is useful in testing out pathfinding and teamwork algorithms without having to use robots and deal with their problems. However, the chosen simulation system is limited in the amount of simulation is possible. Additional work has been done to make the simulation more realistic and test alternative algorithms.

Introduction

In the Robocup small-sized league, teams of five robots each compete in a soccer style match. An overhead camera is mounted 4 meters above the playing field. Vision and robot planning is centralized with only simple instructions being sent to each robot. A simulation system is being used to solve the problem of testing pathfinding and teamwork algorithms without the vision system and robots being finished. With this system, robots are assumed to be working perfectly so only the artificial intelligence needs to be tested.

The simulation system is based on the source code published in the book Programming Game AI by Example (Buckland). The simulation creates a basic implementation of the Robocup small-sized league with simplified rules. There are five players on each team. The bounds of the field are treated as walls so the ball never goes out of bounds. A simple team intelligence system is included. This system was chosen because it gives a usable visual output, included basic physics calculations, and the team intelligence provides a benchmark for new implementations.

The simulation falls short in several areas. The team intelligence is limited and even basic systems can outmaneuver it. The player simulation is based on a two wheeled robot that can only move in the direction it is facing. The University of Rochester Robocup team is building robots with an omniwheel design that allows full freedom of motion. The robots can move in any direction while facing or turning to any orientation. This change requires a new pathfinding system that is able to take advantage over the additional freedom of motion. Finally, the simulation assumes perfect information and execution. This is not possible in the real world. Robot commands have been clamped to simulate motor limitations and noise has been added to the position of the robots and ball to simulate vision

limitations.

Basic Simulation System

The provided simulation code was used as the basis for the Robocup simulation. The source is divided into three parts, the soccer pitch, soccer team, and the objects that play the game such as the ball and players. This system has been supplemented by classes for new versions of the players, team, and player states. The new classes generally have a 2 on the end of the class name. For example, FieldPlayer2 and SoccerTeam2. The original classes generally do not have a number in their title, such as SoccerPitch and PlayerBase. Both versions of the code use several classes from the Common package. The most important class is the Vector2D class which handles many of the position and orientation calculations.

The SoccerPitch class

The SoccerPitch class initializes all the objects including the teams, ball, goal, and bounds and connects them to the GUI. This class is the top level object that calls updates and render on all the other objects in the simulation. This class determines the game state which controls the team level state machines. Teams are either attacking or defending while the game is going. The special state prepare for kickoff is used at the beginning of the game and whenever a goal is scored. This state system will have to be expanded to work with Robocup's throw-ins, corner kicks, and penalty shots.

One part of the code that was not included in the updated version is a region system. This system divides the playing field into eighteen regions. Players were assigned home regions which they

went to when the team did not have the ball. The problem with this implementation was that the players sitting in home regions did not track the ball until another teammate passed it to them. When designing the new player states, this method was removed. However, it could be useful in the future especially in the state that sets up passing and more complex defense (Nardi, 614).

The SoccerTeam class

The soccer team initializes the players onto the blue or red team. During each update, in addition to calling updates for the individual players, several methods are called for the team based intelligence. Pointers to four key players are stored at the team level in addition to the list of players. The players are the player closest to the ball, the supporting player, player about to receive a pass, and player in control of the ball. On each update, distances to the ball are calculated first and the closest player pointer is updated if necessary. The supporting player is the player remaining when one has the ball, one is a goal keeper, and two are acting as defenders. This player seeks out a position likely to score a goal after a pass using the SupportSpotCalculator.

The SupportSpotCalculator determines the best location for the supporting player to move to using three criteria. The opponents side of the field is separated into many small regions and each is tested separately for it's usefulness in support of the controlling player. First, the support spot calculator checks if a safe pass is possible. This is done by looking at each opponent's distance to the ball. Locations are rejected if an opponent is currently blocking the path between the two players or if an opponent could potentially block the path. Next, the support spot calculator determines if a goal can be scored from the current location. This uses the same pass checking code except that instead of passing between players, it is a pass from the current supporting player to a random spot on the goal.

This check is run several times on different locations in the goal. Finally, locations farther away from the controlling player are given more weight than close locations. This method favors long passes but sets a threshold on the maximum path length.

The isPassSafeFromOpponent method is used multiple times for determining both good passing and goal scoring opportunities. Given a start and end point to the pass, a passing force/speed, and the opponent, it calculates whether the opponent can disrupt the pass. It first converts the opponents location to local space, with the origin equal to the pass start point. Next it checks if the opponent is behind the start point in relation to the end point. If this is the case, the opponent cannot affect the pass. If the opponent is closer to the target than the receiver the pass attempt is rejected. Last, it checks whether the opponent can reach the position orthogonal to the balls path. If each of these checks fails, the pass is clear for the current opponent.

The SoccerBall class

The SoccerBall class controls the movement for the simulated soccer ball. The Kick method accelerates the ball in the given direction. Using the methods TimeToCoverDistance and FuturePosition, the ball's location can be determined in future time steps. This allows for basic prediction of ball movement. TestCollisionWithWalls is used to make the ball bounce off of the edges of the soccer pitch. To work with the Robocup system, this method will need to be altered or removed.

The FieldPlayer and GoalKeeper classes

The FieldPlayer and GoalKeeper classes represent the simulated players on the soccer pitch.

Both use a state machine to determine how they move and the state machine is controlled by the SoccerTeam object.

The GoalKeeper has a state machine consisting of TendGoal, ReturnHome and InterceptBall. TendGoal moves the player between the ball and the center of the goal. When the goal keeper is close to the ball, it moves to the intercept state. After it has passed the ball to a teammate, it returns to the TendGoal state through ReturnHome. This behavior is similar to what is wanted for Robocup defenders.

The FieldPlayer has a more complex state machine consisting of states to control the player with and without the ball. Dribble, and KickBall are used to control the player with the ball. Dribble uses many small kicks to direct the ball. When an opponent comes near or a teammate becomes open, a player will attempt a pass. ChaseBall, RecieveBall, and SupportAttacker are used when the player does not have the ball. ChaseBall is assigned to the closest player to the ball. SupportAttacker attempts to move the player to a location with a clear pass from the controlling player. Once the pass is initiated, the state is switched to ReceiveBall.

Pathfinding is handled by the SteeringBehaviors class. The players are given a target location and attempt to follow a straight line course to the location. Obstacle avoidance is accomplished by adding a repelling force from each other object on the field. This a similar solution but different implementation than the potential field search used in the update version.

Improved Simulation System

In order to make the system simulate the robots more closely, several classes were added or changed. The FieldPlayer class has been replaced by FieldPlayer2 and SoccerTeam by SoccerTeam2. The state system and pathfinding system have been completely replaced.

Pathfinding System

The main difference between the movement of the original FieldPlayer class and new FieldPlayer2 class is how turning works. In the FieldPlayer class, players are restricted to only moving in the direction they are facing. With the ominwheels of the Robocup robot, the robots orientation is independent of its velocity. The independence of the two attributes means they must be calculated separately. The angular velocity calculation remains the same but the velocity calculation must be replaced.

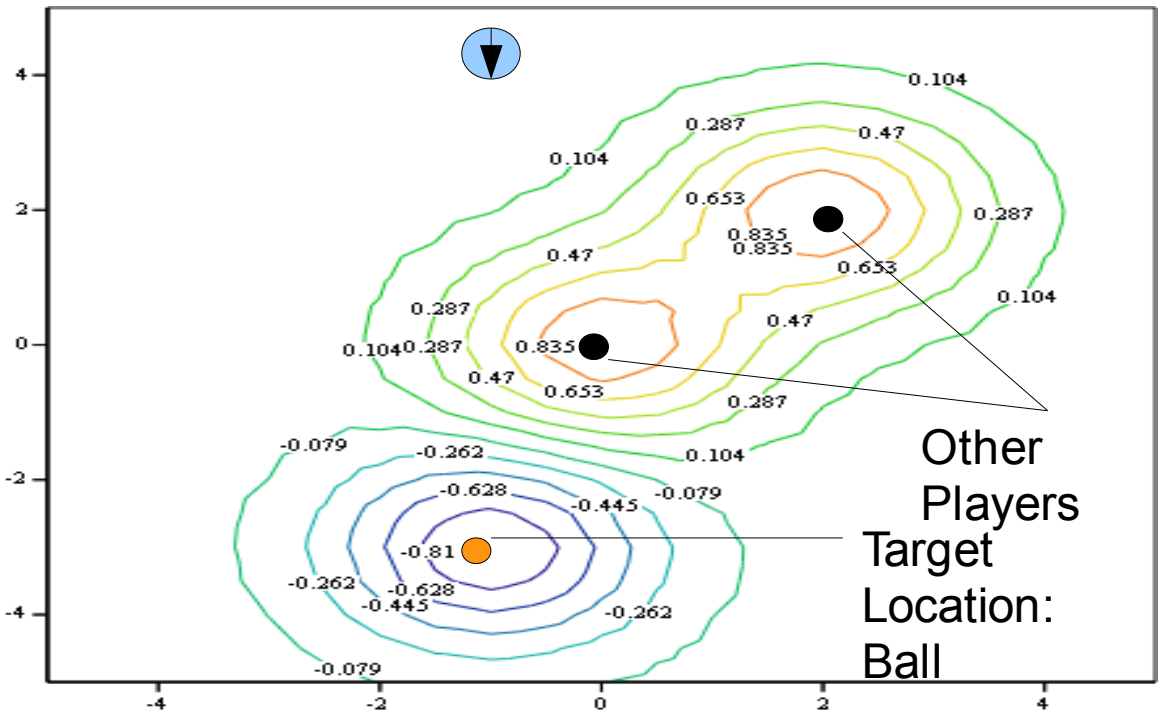
The robots current pathfinding system integrates the attraction force of the target location and repulsion force of the other players by using a potential field system. This algorithm assigns a positive normal distribution to repulsors and a negative normal distribution to attractors. This distribution is defined by

$$E(x) = -\sum_{a \in A} \alpha_a e^{\frac{-\gamma a}{2} * \|x-a\|^2} + \sum_{r \in R} \alpha_r e^{\frac{-\gamma r}{2} * \|x-r\|^2}$$

(Beard, 3).

The set A is the set of attractors and R the set of repulsors. The α_a and α_a are the means and γ_a and γ_a

are the variance of the attractors and repulsors. By summing the negative fields of the attractors plus the fields of the repulsors, a map can be created that determines how the player should move.



k

The problem with this system is that it must sample multiple points around the current position to determine which direction to move in. By taking the partial derivative with respect to x and y, the component direction ad magnitude of movement can be determined by looking at the gradient.

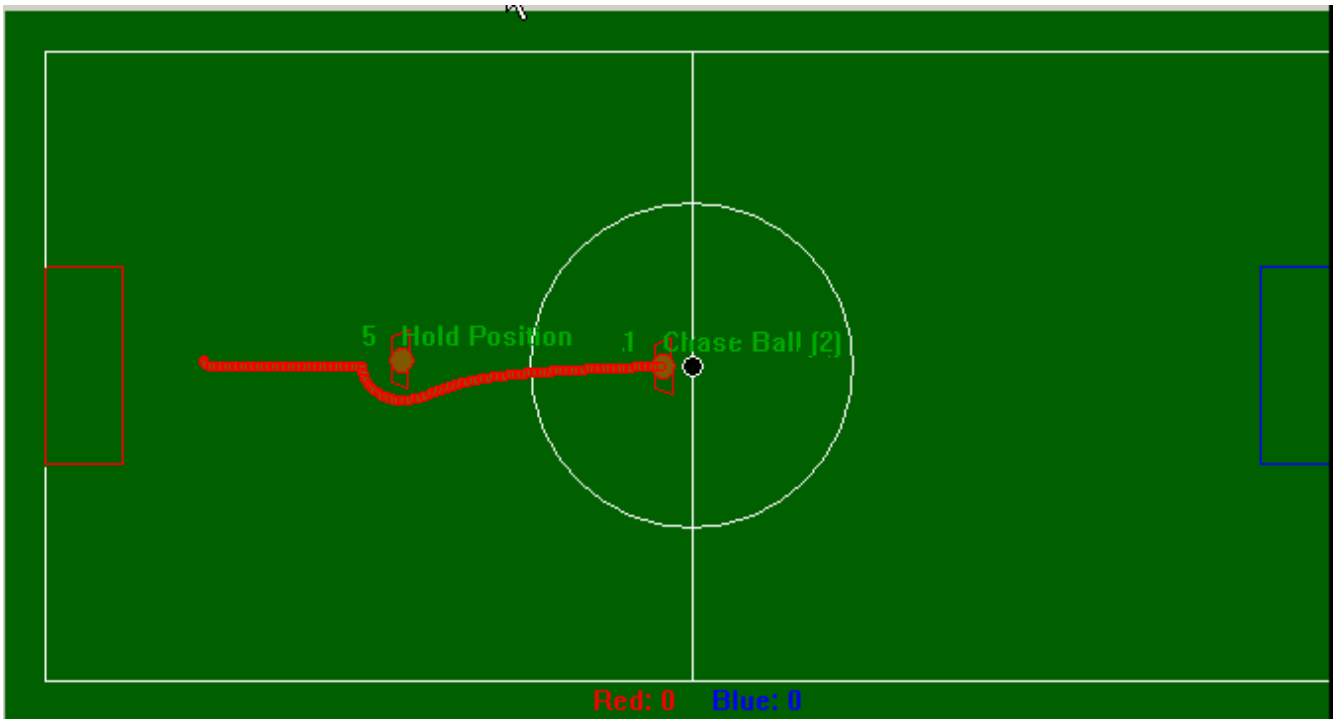
$$\frac{dE}{dx}(x) = -\sum_{a \in A} \alpha_a \gamma_a (x-a) e^{\frac{-\gamma a}{2} * \|x-a\|^2} + \sum_{r \in R} \alpha_r \gamma_a (x-r) e^{\frac{-\gamma r}{2} * \|x-r\|^2}$$

(Beard, 4).

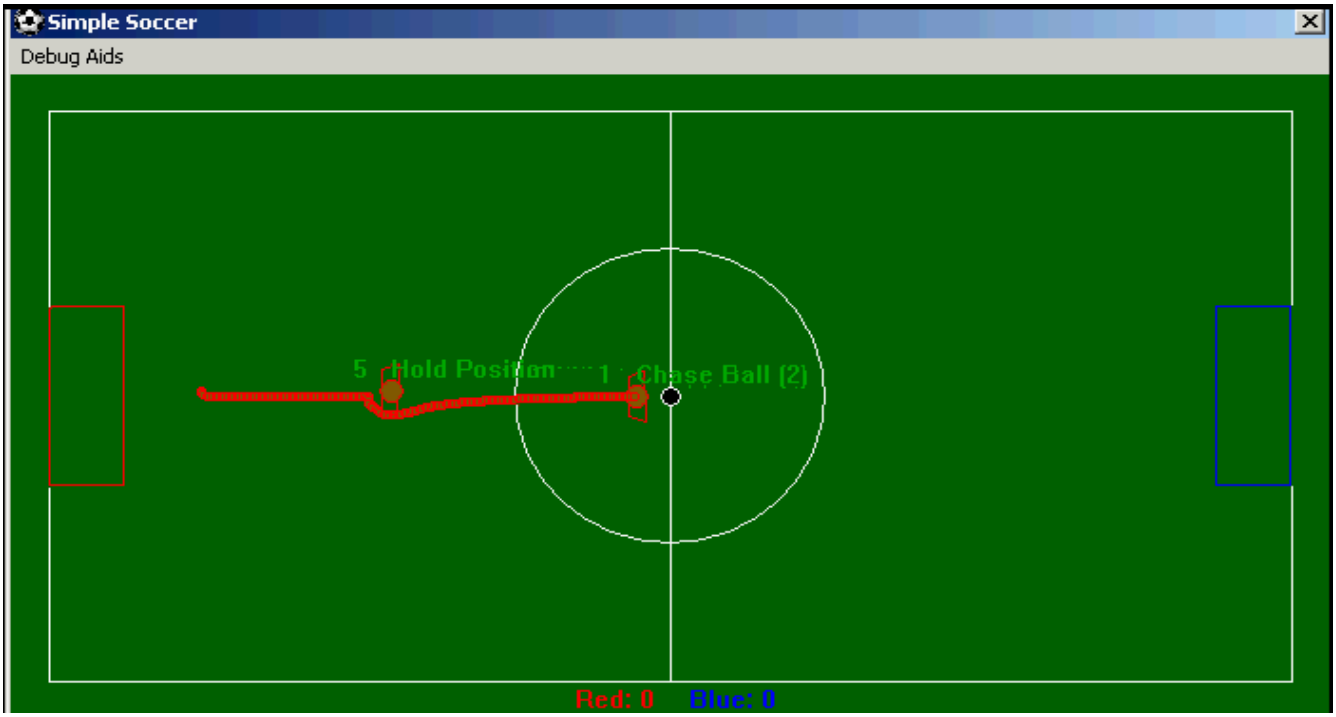
To fine tune spacing the α and γ parameters can be set. α is used to determine the maximum value of the normal distribution. The target should have the highest value and the obstacles should have the

same or lower values. Setting the obstacles to lower values ensures that the player always moves towards the target but may create problems with collisions close to the target. Setting the two αs to the same value solves the obstacle problem but can lead to cases where the players get stuck in a local minimum in the potential field.

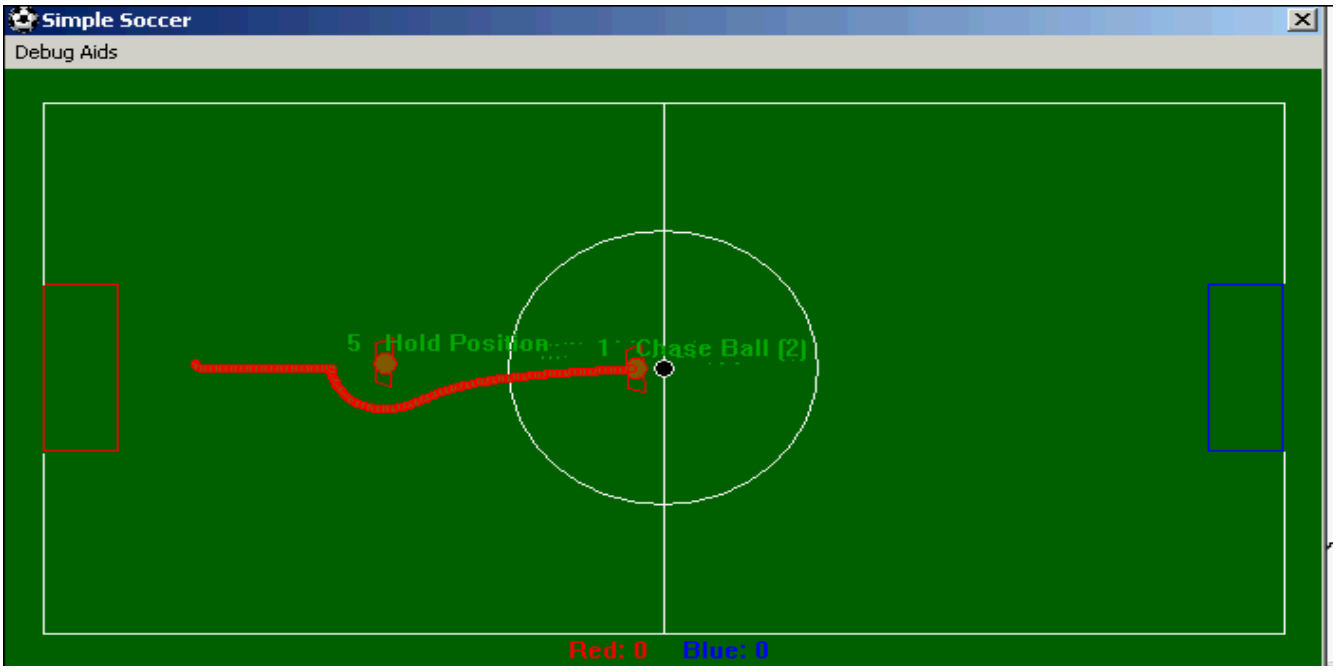
The γs represent the variance and determine the width of the potential field. Setting a lower γ value for the target is necessary so that it has some attractive force across the entire field. Setting a higher value for the other players lowers the area where the repulsing field dominates the attracting field. Setting the value too high can lead to collision problems.



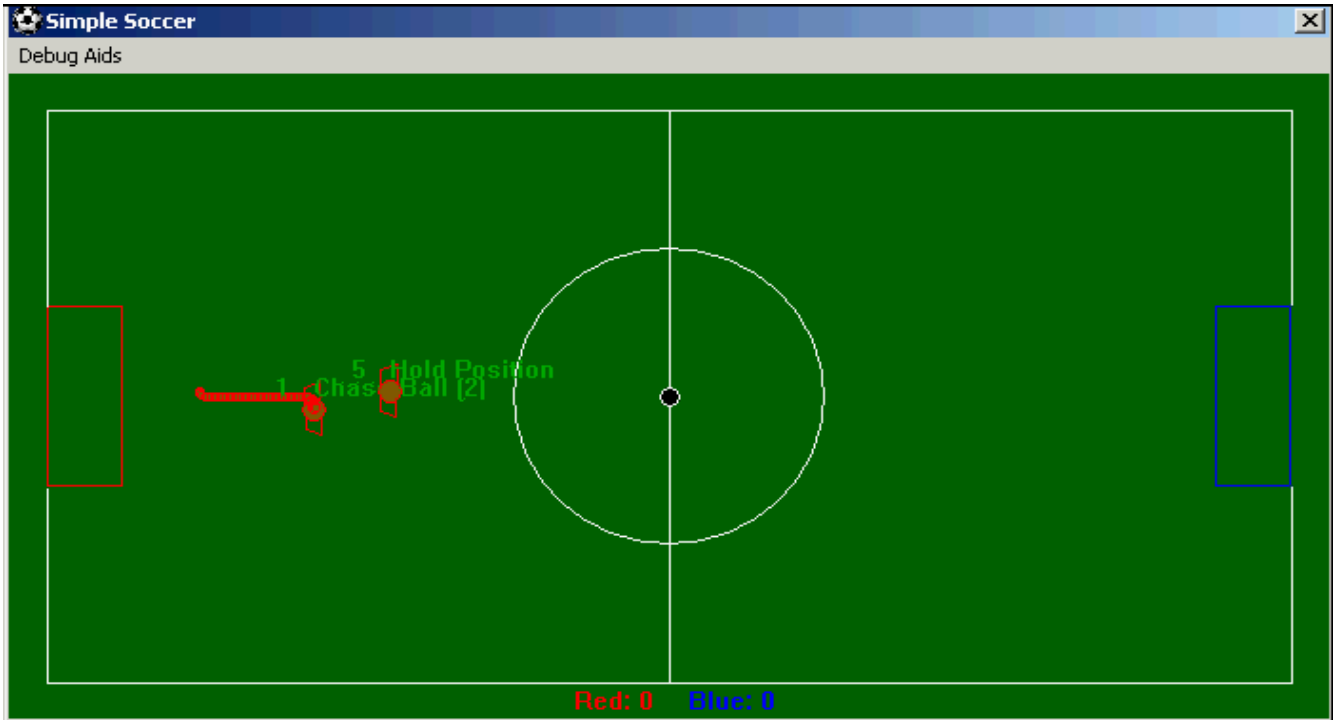
With a medium variance of $\gamma_a = 0.0002$, the player avoids obstacles by about one player width of space.



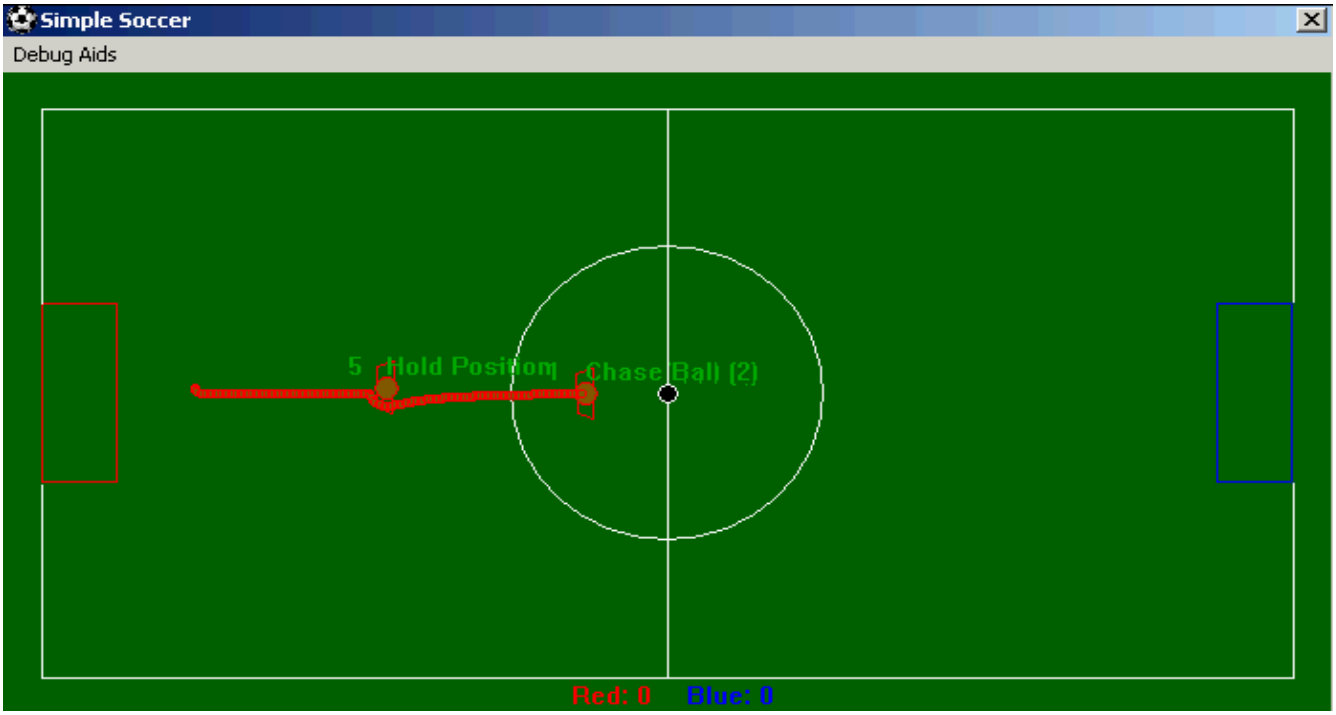
With a high variance of $\gamma_a = 0.0005$, the player comes much closer to the obstacle, with almost no margin for error.



With a low variance of $\gamma_a = 0.0001$, the player gives a wide berth to the obstacle, taking a longer path to the target location.

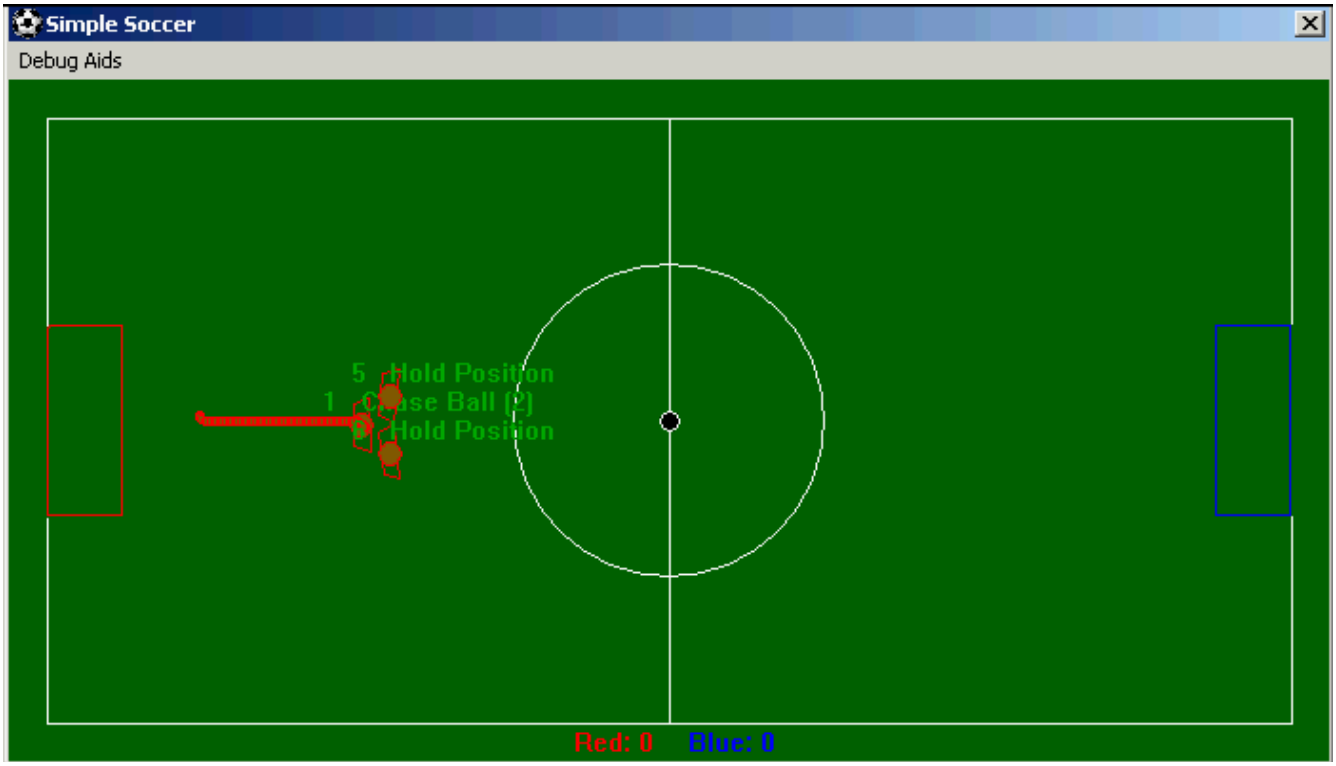


If the variance is too high, the attractive force of the ball dominates the repulsive force of the obstacle.

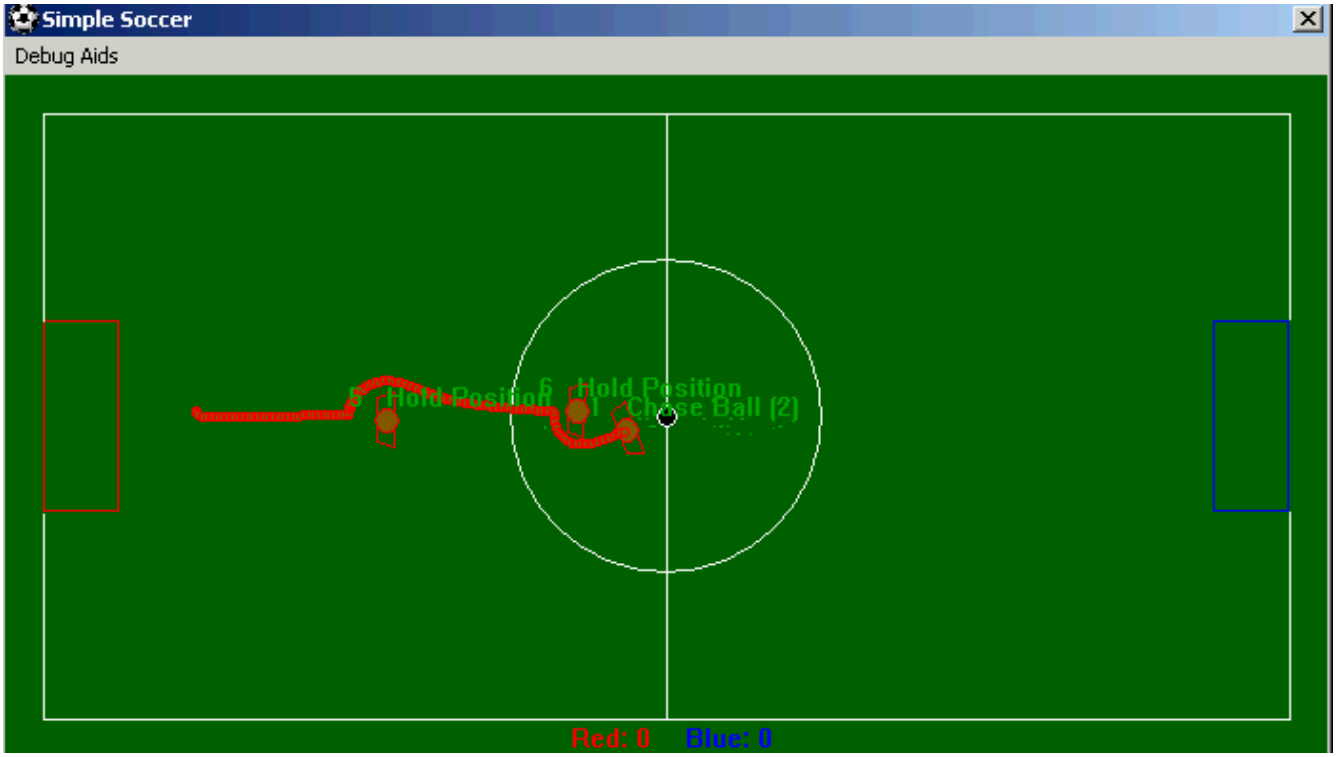


If the variance is too low, the repulsive force of the obstacle dominates the attractive force of the ball.

The player becomes stuck behind the obstacle.



The main problem with the potential field algorithm is the local minimum problem. This occurs when two repulsors are between the player and it's target. In this case, it is possible that the potential field algorithm will make the robot stuck in a single location. This has proved to be a problem with the included team intelligence because in their base defending state, the field players do not move.



Another problem with the potential field algorithm is the slow speed in a situation with multiple obstacles. It is most noticeable when the attractive force of the target overcomes the repulsive force of a nearby player by a small amount. In this case, the player will move slowly until it leaves the influence of the opponents repulsing field. A clear path to the ball may not be taken at a reasonable speed due to this problem. Due to the speed of a Robocup match, a supplemental pathfinding algorithm may be needed for this situation.

State System

The updated state system works in a similar way to the old system. Each state has a enter, execute, and exit method. The execute method for the player's current state gets called on every update. Whenever a state is changed, the exit method of the current state and the enter method of the

new state is called. Each player is assigned a start state. This is the initial state for the player. When the player or team loses the ball, the player generally reverts back to the start state.

The most important state is the ChaseBall state. In this state, the player tracks the ball and attempts to intersect it. It is used as a starting state for the two attacking players. Players stay in this state until one of several situations happen. If the player reaches, it moves to the HasBall state which has additional controls for passing, dribbling and shooting. If the player's start state is not ChaseBall, and it is no longer the closest player to the ball, it reverts back to its start state.

This state could be improved by predicting where the ball will be in the future. Currently, the player moves towards the current location of the ball. This is a problem because the ball can move faster than the players. When tracking a fast moving ball, the player will often move in circles without getting much closer to the ball. An alternate solution would be to track the location of the ball in the future based on the distance between the player to the ball. The amount of time in the future to track would be proportional to the distance between the player and ball and maximum player speed.

The DefendLeft, DefendMiddle, and DefendRight states create a defensive wall of robots. These robots try to position themselves between the ball and the goal. They stay at the halfway point between the ball and the goal with the left and right defenders offset with respect to the sides of the goal. When a defender is the closest player on the team to the ball, it moves out of position and tries to intercept the ball by moving to the ChaseBall state. These states could use the similar improvement of tracking future ball positions rather than the current position.

The HasBall state currently controls the player when it is in possession of the ball. It is

automatically left when the ball is kicked or stolen by another player. Although this state currently stores all the player plans when it has the ball, it may be split into several states in the future. This change may need to be motivated due to the limitations on the kicker of the physical robots. Due to the design, the capacitor powering the kicker cannot be discharged without powering the kicker. This means that once the decision is made to shoot the ball, a soft pass cannot be attempted until the kicker is discharged.

Simulating visual noise

One problem with the simulation system is that it does not accurately represent noise in the visual system. Noise in the vision recognition system comes from two areas. First, the camera itself can actually move or sway. It is on a four meter mount that may not be completely stationary. Second, the camera is not perfectly accurate when determining positions of objects on the field. There may be some noise based on changing lighting conditions and shadows. To simulate these effects, noise was added to the position of the ball and players. A Gaussian distribution with mean of 0 and variance of 1 was used to move the objects randomly while favoring small adjustments. Demonstrations of the noise can be found in attached videos *ballnoise-demo.avi*, *highballnoise.avi*, *ballnoise-outofboundsproblem.avi*, and *ballandplayernoise.avi.*

This added noise uncovers several possible problems when converting the simulation to real robots. The most important problem is tracking the ball when the players have the state ChaseBall. Based on the Robocup rules, the maximum speed of the ball is significantly larger than the speed of the players. This was simulated by adding a larger amount of noise to the ball's movement in *highballnoise.avi*. In this case, planning ahead to track the ball is much less effective than using perfect

motion. Due to the small size and high speed of the ball, moving to a predicted intersecting location does not always result in gaining control of the ball. One possible solution to this problem is to average the last few positions of the ball with the current position, determine an approximate velocity for the ball, and use a smoother predicted location rather than actual location of the ball.

In the current simulation, the ball bounces off of the bounds of the playing field. This is a simplification to avoid extra code to handle throw-ins and corner kicks. However, due to the noise, it is possible for the ball to jump over the boundary of the field and keep going as shown by *ballnoise-outofboundsproblem.avi*. This is only a problem with the simulation and needs to be fixed by adding extra team level states.

With the current simulation system, a player has control of the ball when it is within a small radius of the center of the player. This will not work for the Robocup system because the robots only have control of the ball when they are oriented correctly and have their dribbler on. In this case, the ball's predicted location should be at the front of the controlling robot. However, due to noise in the vision system, the ball may be incorrectly placed away from the robot suggesting it has lost control of the ball. One method for solving this problem would be to put a sensor on the front of the robot that can tell when a ball is in the dribbler, ready to be kicked. With this system, noise in the ball's location can be discarded until the sensor returns a negative confirmation.

The most important problem with noise is colliding with other robots. In Robocup, collisions are considered penalties and a free shot is given to the other team. If the amount of noise is high, the players must give other players a wider path to avoid collisions. This can be handled in the code by the moveToTarget method of the FieldPlayer2 class. The parameters of the pathfinding algorithm can be

tweaked so that the players leave a larger or smaller amount of space between each other when determining paths. The current setting leaves a space of roughly one robot's width when planning paths. Lowering this value will increase the amount of open space required between robots. However, a low value increases the likelihood of the players getting stuck in local minimums.

Visual noise on robots pose additional challenges compared to the ball for two reasons. First, the robots on either team can change direction quickly making velocity based movement prediction significantly less useful. Second, we do not have a completely correct model for translating robot motion to wheel commands. This model is further complicated by differences in motor strength, field friction, and wheel slip so there will always be some imprecision in these predictions. This is the current model for determining wheel motion:

$$(\frac{1}{r})*(-\sin(\theta))*a_i*\dot{x}*\cos(\theta+a_i)*\dot{y}+R*\dot{\theta}\,X$$

θ = *direction of movement*
$\dot{\theta}$ = *angular velocity*
$\dot{x}$ = *velocity * sin(direction), velocity on x axis*
$\dot{y}$ = *velocity * cos(direction), velocity on y axis*
r = *0.0246 m, the radius of the wheel*
R = *0.0767, the distance from the center of the robot to the center of the wheel*
a_i = *the angle of the wheel*
(Baede, 11).

This model simply adds the velocity and angular velocity of the robot to convert to motor commands. This model is not perfect when the angular velocity is non-zero. Higher angular velocities will lead to larger errors. This system is dependent on a fast update of motor commands to quickly correct errors in movement.

Conclusion

Path planning using potential fields is suitable for the Robocup problem with a few exceptions. It is especially suited to this problem because it does not look beyond the present when planning paths. In the average case, it is able to efficiently move the player towards its goal. The updated state system with three defenders and two attacker works well as a defensive system. It is able to handle noise on both the players and the ball. Determining the maximum amount of noise can be used to set parameters that define the spacing of the players.

Future Work

Testing the robot motor control code needs to be done to ensure that the player intelligence works. It is dependent on fairly accurate robot movement to avoid collisions and interact with the ball. Additional team states are needed to handle special situations dealing with the ball going out of bounds. These states are not necessary as long as the actual robots are not being used.

Although the potential field pathfinding system works, there have been multiple cases of problems of robots getting stuck due to local a minimum in the field. More testing needs to be done to see if these situations are common or rare. If the situations causing these problems are common and not solvable by adjusting parameters, a secondary pathfinding algorithm may be necessary.

The teamwork and passing systems could be improved by modifying the included support spot calculator class. It needs a simple modification to use BaseGameEntities or FieldPlayer2s rather than PlayerBase objects in it's calculations.

One potential problem that has not been considered with the current situation is the interaction between the ball and the players. The ball currently does not bounce off of players, only walls. This simplifies the pathfinding and kicking by allowing the robots to kick whenever they are close to the ball. The players can approach the ball from any direction and are artificially limited to kick it only in the the direction the robot is facing. With the physical robots, positioning the robots before they reach the ball is important. First the robots, must move to a location where they can engage the ball without touching it. After that position is reached, the robot can engage the ball. This can be accomplished by treating the ball as an obstacle most of the time. A positioning state can replace the ChaseBall state which determines a position on a circle a fixed distance away from the ball as it's target location.

Bibliography

Baede, T. A. “Motion control of an omnidirectional robot.” Department of Mechanical Engineering, National University of Singapore.

Beard, Randal W. (2003) “Motion Planning using Potential Fields.” Electrical and Computer Engineering, Brigham Young University, Provo, Utah, 2003; <http://www.ee.byu.edu/ugrad/srprojects/robotsoccer/labs/skills/potential.pdf>

Buckland, Mat. *Programming Game AI by Example*. Plano, Texas. Wordware Publishing, Inc. 2005.

“Laws of the F180 League, 2009.” 4 May 2009 <http://small-size.informatik.uni-bremen.de/_media/rules:ssl-rules-2009.pdf>

Maeno, Junya et al.”RoboDragons 2008 Team Description” Aichi Prefectural University, Nagakute-cho, Aichi. 480-1198.

Nardi, D. et al. (Eds.) “Towards a League-Independent Qualitative Soccer Theory for Robocup.” *Robocup 2004,* LNA13276, pp. 611-618, 2005. Springer-Verlag Berlin Heidelberg 2005.