To appear in *Proceedings of the RoboCup International Symposium 2012* (RoboCup 2012), Mexico City, Mexico, June 2012.

Positioning to Win: A Dynamic Role Assignment and Formation Positioning System

Patrick MacAlpine, Francisco Barrera, and Peter Stone

Department of Computer Science, The University of Texas at Austin
{patmac,tank225,pstone}@cs.utexas.edu

Abstract. This paper presents a dynamic role assignment and formation positioning system used by the 2011 RoboCup 3D simulation league champion UT Austin Villa. This positioning system was a key component in allowing the team to win all 24 games it played at the competition during which the team scored 136 goals and conceded none. The positioning system was designed to allow for decentralized coordination among physically realistic simulated humanoid soccer playing robots in the partially observable, non-deterministic, noisy, dynamic, and limited communication setting of the RoboCup 3D simulation league simulator. Although the positioning system is discussed in the context of the RoboCup 3D simulation environment, it is not domain specific and can readily be employed in other RoboCup leagues as it generalizes well to many realistic and real-world multiagent systems.

1 Introduction

Coordinated movement among autonomous mobile robots is an important research area with many applications such as search and rescue [1] and warehouse operations [2]. The RoboCup 3D simulation competition provides an excellent testbed for this line of research as it requires coordination among autonomous agents in a physically realistic environment that is partially observable, non-deterministic, noisy, and dynamic. While low level skills such as walking and kicking are vitally important for having a successful soccer playing agent, the agents must work together as a team in order to maximize their game performance.

One often thinks of the soccer teamwork challenge as being about where the player with the ball should pass or dribble, but at least as important is where the agents position themselves when they *do not* have the ball [3]. Positioning the players in a formation requires the agents to coordinate with each other and determine where each agent should position itself on the field. While there has been considerable research done in the 2D soccer simulation domain (for example by Stone et al. [4] and Reis et al. [5]), relatively little outside of [6] has been published on this topic in the more physically realistic 3D soccer simulation environment. [6], as well as related work in the RoboCup middle size league (MSL) [7], rank positions on the field in order of importance and then iteratively assign the closest available agent to the most important currently unassigned position until every agent is mapped to a target location. The work presented in this paper differs from the mentioned previous work in the 2D and 3D simulation and MSL RoboCup domains as it takes into account real-world concerns and movement dynamics such as the need for avoiding collisions of robots.

In UT Austin Villa's positioning system players' positions are determined in three steps. First, a full team formation is computed (Section 3); second, each player computes the best assignment of players to role positions in this formation according to its own view of the world (Section 4); and third, a coordination mechanism is used to choose among all players' suggestions (Section 4.4). In this paper, we use the terms (player) position and (player) role interchangeably.

The remainder of the paper is organized as follows. Section 2 provides a description of the RoboCup 3D simulation domain. The formation used by UT Austin Villa is given in Section 3. Section 4 explains how role positions are dynamically assigned to players. Collision avoidance is discussed in Section 5. An evaluation of the different parts of the positioning system is given in Section 6, and Section 7 summarizes.

2 Domain Description

The RoboCup 3D simulation environment is based on SimSpark,[1] a generic physical multiagent system simulator. SimSpark uses the Open Dynamics Engine[2] (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. ODE also provides support for the modeling of advanced motorized hinge joints used in the humanoid agents.

The robot agents in the simulation are homogeneous and are modeled after the Aldebaran Nao robot,[3] which has a height of about 57 cm, and a mass of 4.5 kg. The agents interact with the simulator by sending torque commands and receiving perceptual information. Each robot has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck. In order to monitor and control its hinge joints, an agent is equipped with joint perceptors and effectors. Joint perceptors provide the agent with noise-free angular measurements every simulation cycle (20 ms), while joint effectors allow the agent to specify the torque and direction in which to move a joint. Although there is no intentional noise in actuation, there is slight actuation noise that results from approximations in the physics engine and the need to constrain computations to be performed in real-time. Visual information about the environment is given to an agent every third simulation cycle (60 ms) through noisy measurements of the distance and angle to objects within a restricted vision cone (120°). Agents are also outfitted with noisy accelerometer and gyroscope perceptors, as well as force resistance perceptors on the sole of each foot. Additionally, agents can communicate with each other every other simulation cycle (40 ms) by sending messages limited to 20 bytes.

1 `http://simspark.sourceforge.net/`
2 `http://www.ode.org/`
3 `http://www.aldebaran-robotics.com/eng/`

3 Formation

This section presents the formation used by UT Austin Villa during the 2011 RoboCup competition. The formation itself is not a main contribution of this paper, but serves to set up the role assignment function discussed in Section 4 for which a precomputed formation is required.

In general, the team formation is determined by the ball position on the field. As an example, Figure 1 depicts the different role positions of the formation and their relative offsets when the ball is at the center of the field. The formation can be broken up into two separate groups, an offensive and a defensive group. Within the offensive group, the role positions on the field are determined by adding a specific offset to the ball's coordinates. The *onBall* role, assigned to the player closest to the ball, is always based on where the ball is and is therefore never given an offset. On either side of the ball are two forward roles, *forwardRight* and *forwardLeft*. Directly behind the ball is a *stopper* role as well as two additional roles, *wingLeft* and *wingRight*, located behind and to either side of the ball. When the ball is near the edge of the field some of the roles' offsets from the ball are adjusted so as to prevent them from moving outside the field of play.

Within the defensive group there are two roles, *backLeft* and *backRight*. To determine their positions on the field a line is calculated between the center of the team's own goal and the ball. Both backs are placed along this line at specific offsets from the end line. The goalie positions itself independently of its teammates in order to always be in the best position to dive and stop a shot on goal. If the goalie assumes the *onBall* role, however, a third role is included within the defensive group, the *goalieReplacement* role. A field player assigned to the *goalieReplacement* role is told to stand in front of the center of the goal.

During the course of a game there are occasional stoppages in play for events such as kickoffs, goal kicks, corner kicks, and kick-ins. When one of these events occur UT Austin Villa adjusts its team formation and behavior to assume situational set plays which are detailed in a technical report [8].

Kicking and passing have yet to be incorporated into the team's formation. Instead the *onBall* role always dribbles the ball toward the opponent's goal.

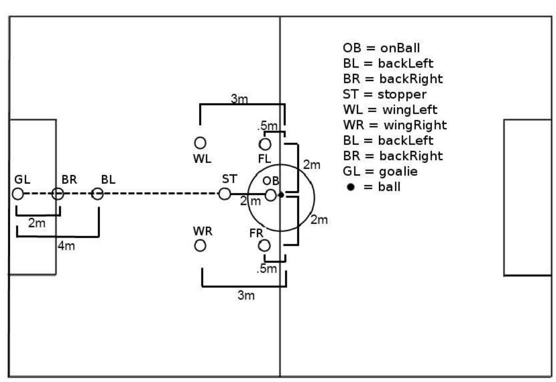

Fig. 1. Formation role positions.

4 Assignment of Agents to Role Positions

Given a desired team formation, we need to map players to roles (target positions on the field). A naïve mapping having each player permanently mapped to one of the roles performs poorly due to the dynamic nature of the game. With such static roles an agent assigned to a defensive role may end up out of position and, without being able to switch roles with a teammate in a better position to defend, allow for the opponent to have a clear path to the goal. In this section, we present a dynamic role assignment algorithm. A role assignment algorithm can be thought of as implementing a role assignment *function*, which takes as input the state of the world, and outputs a one-to-one mapping of players to roles. We start by defining three properties that a role assignment function must satisfy (Section 4.1). We then construct a role assignment function that satisfies these properties (Section 4.2). Finally, we present a dynamic programming algorithm implementing this function (Section 4.3).

4.1 Desired Properties of a Valid Role Assignment Function

Before listing desired properties of a role assignment function we make a couple of assumptions. The first of these is that no two agents and no two role positions occupy the same position on the field. Secondly we assume that all agents move toward fixed role positions along a straight line at the same constant speed. While this assumption is not always completely accurate, the omnidirectional walk used by the agent, and described in [9], gives a fair approximation of constant speed movement along a straight line.

We call a role assignment function *valid* if it satisfies three properties:

1. *Minimizing longest distance* - it minimizes the maximum distance from a player to target, with respect to all possible mappings.
2. *Avoiding collisions* - agents do not collide with each other as they move to their assigned positions.
3. *Dynamically consistent* - a role assignment function f is dynamically consistent if, given a *fixed* set of target positions, if f outputs a mapping m of players to targets at time T, and the players are moving toward these targets, f would output m for every time $t > T$.

The first two properties are related to the output of the role assignment function, namely the mapping between players and positions. We would like such a mapping to minimize the time until all players have reached their target positions because quickly doing so is important for strategy execution. As we assume all players move at the same speed, we start by requiring a mapping to minimize the maximum distance any player needs to travel. However, paths to positions might cross each other, therefore we additionally require a mapping to guarantee that when following it, there are no collisions. The third property guarantees that once a role assignment function f outputs a mapping, f is committed to it as long as there is no change in the target positions. This guarantee is necessary as otherwise agents might unduly thrash between roles thus impeding progress. In the following section we construct a valid role assignment function.

4.2 Constructing a Valid Role Assignment Function

Let M be the set of all one-to-one mappings between players and roles. If the number of players is n, then there are $n!$ possible such mappings. Given a state of the world, specifically n player positions and n target positions, let the *cost* of a mapping m be the n-tuple of distances from each player to its target, sorted in decreasing order. We can then sort all the $n!$ possible mappings based on their costs, where comparing two costs is done lexicographically. Sorted costs of mappings from agents to role positions for a small example are shown in Figure 2.

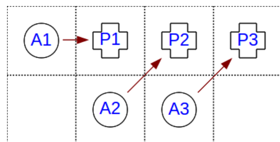

Fig. 2. Lowest lexicographical cost (shown with arrows) to highest cost ordering of mappings from agents (A1,A2,A3) to role positions (P1,P2,P3). Each row represents the cost of a single mapping.

1: $\sqrt{2}$ (A2→P2), $\sqrt{2}$ (A3→P3), 1 (A1→P1)
2: 2 (A1→P2), $\sqrt{2}$ (A3→P3), 1 (A2→P1)
3: $\sqrt{5}$ (A2→P3), 1 (A1→P1), 1 (A3→P2)
4: $\sqrt{5}$ (A2→P3), 2 (A1→P2), $\sqrt{2}$ (A3→P1)
5: 3 (A1→P3), 1 (A2→P1), 1 (A3→P2)
6: 3 (A1→P3), $\sqrt{2}$ (A2→P2), $\sqrt{2}$ (A3→P1)

Denote the role assignment function that always outputs the mapping with the lexicographically smallest cost as f_v. Here we provide an informal proof sketch that f_v is a valid role assignment; we provide a longer, more thorough derivation in a technical report [8].

Theorem 1 *f_v is a valid role assignment function.*

It is trivial to see that f_v minimizes the longest distance traveled by any agent (Property 1) as the lexicographical ordering of distance tuples sorted in descending order ensures this. If two agents in a mapping are to collide (Property 2) it can be shown, through the triangle inequality, that f_v will find a lower cost mapping as switching the two agents' targets reduces the maximum distance either must travel. Finally, as we assume all agents move toward their targets at the same constant rate, the distance between any agent and target will not decrease any faster than the distance between an agent and the target it is assigned to. This observation serves to preserve the lowest cost lexicographical ordering of the chosen mapping by f_v across all timesteps thereby providing dynamic consistency (Property 3). Section 4.3 presents an algorithm that implements f_v.

4.3 Dynamic Programming Algorithm for Role Assignment

In UT Austin Villa's basic formation, presented in Section 3, there are nine different roles for each of the nine agents on the field. The goalie always fills the *goalie* role and the *onBall* role is assigned to the player closest to the ball. The other seven roles must be mapped to the agents by f_v. Additionally, when the goalie is closest to the ball, the goalie takes on both the *goalie* and *onBall* roles causing us to create an extra *goalieReplacement* role positioned right in front of the team's goal. When this occurs the size of the mapping increases to eight agents mapped to eight roles. As the total number of mapping permutations is $n!$, this creates the possibility of needing to evaluate 8! different mappings.

Clearly f_v could be implemented using a brute force method to compare all possible mappings. This implementation would require creating up to $8! = 40,320$ mappings, then computing the cost of each of the mappings, and finally sorting them lexicographically to choose the smallest one. However, as our agent acts in real time, and f_v needs to be computed during a decision cycle (20 ms), a brute force method is too computationally expensive. Therefore, we present a dynamic programming implementation shown in Algorithm 1 that is able to compute f_v within the time constraints imposed by the decision cycle's length.

Algorithm 1 Dynamic programming implementation

1: HashMap $bestRoleMap = \varnothing$
2: $Agents = \{a_1, ..., a_n\}$
3: $Positions = \{p_1, ..., p_n\}$
4: **for** $k = 1$ to n **do**
5:     **for each** a in $Agents$ **do**
6:         $S = \binom{n-1}{k-1}$ sets of $k-1$ agents from $Agents - \{a\}$
7:         **for each** s in S **do**
8:             Mapping $m_0 = bestRoleMap[s]$
9:             Mapping $m = (a \to p_k) \cup m_o$
10:             $bestRoleMap[\{a\} \cup s] = mincost(m, bestRoleMap[\{a\} \cup s])$
11: **return** $bestRoleMap[Agents]$

Theorem 2 *Let A and P be sets of n agents and positions respectively. Denote the mapping $m := f_v(A, P)$. Let m_0 be a subset of m that maps a subset of agents $A_0 \subset A$ to a subset of positions $P_0 \subset P$. Then m_0 is also the mapping returned by $f_v(A_0, P_0)$.*

A key recursive property of f_v that allows us to exploit dynamic programming is expressed in Theorem 2. This property stems from the fact that if within any subset of a mapping a lower cost mapping is found, then the cost of the complete mapping can be reduced by augmenting the complete mapping with that of the subset's lower cost mapping. The savings from using dynamic programming comes from only evaluating mappings whose subset mappings are returned by f_v. This is accomplished in Algorithm 1 by iteratively building up optimal mappings for position sets from $\{p_1\}$ to $\{p_1, ..., p_n\}$, and using optimal mappings of $k-1$ agents to positions $\{p_1, ..., p_{k-1}\}$ (line 8) as a base when constructing each new mapping of k agents to positions $\{p_1, ..., p_k\}$ (line 9), before saving the lowest cost mapping for the current set of k agents to positions $\{p_1, ..., p_k\}$ (line 10).

An example of the mapping combinations evaluated in finding the optimal mapping for three agents through the dynamic programming approach of Algorithm 1 can be seen in Table 1. In this example we begin by computing the distance of each agent to our first role position. Next we compute the cost of all possible mappings of agents to both the first and second role positions and save off the lowest cost mapping of every pair of agents to the the first two positions. We then proceed by sequentially assigning every agent to the third position and compute the lowest cost mapping of all agents mapped to all three positions. As all subsets of an optimal (lowest cost) mapping will themselves be optimal, we need only evaluate mappings to all three positions which include the previously calculated optimal mapping agent combinations for the first two positions.

{P1}	{P2,P1}	{P3,P2,P1}
A1→P1	A1→P2, f_v(A2→P1)	A1→P3, f_v({A2,A3}→{P1,P2})
A2→P1	A1→P2, f_v(A3→P1)	A2→P3, f_v({A1,A3}→{P1,P2})
A3→P1	A2→P2, f_v(A1→P1)	A3→P3, f_v({A1,A2}→{P1,P2})
	A2→P2, f_v(A3→P1)	
	A3→P2, f_v(A1→P1)	
	A3→P2, f_v(A2→P1)	

Table 1. All mappings evaluated during dynamic programming using Algorithm 1 when computing an optimal mapping of agents A1, A2, and A3 to positions P1, P2, and P3. Each column contains the mappings evaluated for the set of positions listed at the top of the column.

Recall that during the kth iteration of the dynamic programming process to find a mapping for n agents, where k is the current number of positions that agents are being mapped to, each agent is sequentially assigned to the kth position and then all possible subsets of the other $n-1$ agents are assigned to positions 1 to $k-1$ based on computed optimal mappings to the first $k-1$ positions from the previous iteration of the algorithm. These assignments result in a total of $\binom{n-1}{k-1}$ agent subset mapping combinations to be evaluated for mappings of each agent assigned to the kth position. The total number of mappings computed for each of the n agents across all n iterations of dynamic programming is thus equivalent to the sum of the $n-1$ binomial coefficients. That is,

$$\sum_{k=1}^{n} \binom{n-1}{k-1} = \sum_{k=0}^{n-1} \binom{n-1}{k} = 2^{n-1}$$

Therefore the total number of mappings that must be evaluated using our dynamic programming approach is $n2^{n-1}$. For $n = 8$ we thus only have to evaluate 1024 mappings which takes about 3.3 ms for each agent to compute compared to upwards of 50 ms using a brute force approach to evaluate all possible mappings.[4]

4.4 Voting Coordination System

In order for agents on a team to assume correct positions on the field they all must coordinate and agree on which mapping of agents to roles to use. If every agent had perfect information of the locations of the ball and its teammates this would not be a problem as each could independently calculate the optimal mapping to use. Agents do not have perfect information, however, and are limited to

[4] As measured on an Intel Core 2 Duo CPU E8500 @3.16GHz.

noisy measurements of the distance and angle to objects within a restricted vision cone (120°). Fortunately agents can share information with each other every other simulation cycle (40 ms). The bandwidth of this communication channel is very limited, however, as only one agent may send a message at a time and messages are limited to 20 bytes.

We utilize the agents' limited communication bandwidth in order to coordinate role mappings as follows. Each agent is given a rotating time slice to communicate information, as in [4], which is based on the uniform number of an agent. When it is an agent's turn to send a message it broadcasts to its teammates its current position, the position of the ball, and also what it believes the optimal mapping should be. By sending its own position and the position of the ball, the agent provides necessary information for computing the optimal mapping to those of its teammates for which these objects are outside of their view cones. Sharing the optimal mapping of agents to role positions enables synchronization between the agents, as follows.

First note that just using the last mapping received is dangerous, as it is possible for an agent to report inconsistent mappings due to its noisy view of the world. This can easily occur when an agent falls over and accumulates error in its own localization. Additionally, messages from the server are occasionally dropped or received at different times by the agents preventing accurate synchronization. To help account for inconsistent information, a sliding window of received mappings from the last n time-slots is kept by each agent where n is the total number of agents on a team. Each of these kept messages represents a single vote by each of the agents as to which mapping to use. The mapping chosen is the one with the most votes or, in the case of a tie, the mapping tied for the most votes with the most recent vote cast for it. By using a voting system, the agents on a team are able to synchronize the mapping of agents to role positions in the presence of occasional dropped messages or an agent reporting erroneous data. As a test of the voting system the number of cycles all nine agents shared a synchronized mapping of agents to roles was measured during 5 minutes of gameplay (15,000 cycles). The agents were synchronized 100% of the time when using the voting system compared to only 36% of the time when not using it.

5 Collision Avoidance

Although the positioning system discussed in Section 4 is designed to avoid assigning agents to positions that might cause them to collide, external factors outside of the system's control, such as falls and the movement of the opposing team's agents, still result in occasional collisions. To minimize the potential for these collisions the agents employ an active collision avoidance system. When an obstacle, such as a teammate, is detected in an agent's path the agent will attempt to adjust its path to its target in order to maneuver around the obstacle. This adjustment is accomplished by defining two thresholds around obstacles: a *proximity* threshold at 1.25 meters and a *collision* threshold at .5 meters from an obstacle. If an agent enters the *proximity* threshold of an obstacle it will adjust its course to be tangent to the obstacle thereby choosing to circle around to the right or left of said obstacle depending on which direction will move the agent closer to its desired target. Should the agent get so close as to enter the *collision* proximity of an obstacle it must take decisive action to prevent an otherwise imminent collision from occurring. In this case the agent combines the corrective movement brought about by being in the *proximity* threshold with an additional movement vector directly away from the obstacle. Figure 3 illustrates the adjusted movement of an agent when attempting to avoid a collision.

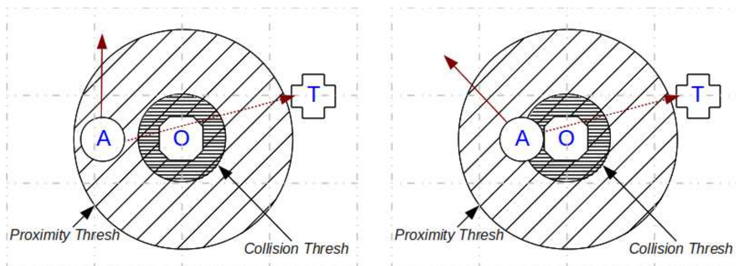

Fig. 3. Collision avoidance examples where agent A is traveling to target T but wants to avoid colliding with obstacle O. The left diagram shows how the agent's path is adjusted if it enters the *proximity* threshold of the obstacle while the right diagram depicts the agent's movement when entering the *collision* threshold. The dotted arrow is the agent's desired path while the solid arrow is the corrected path to avoid a collision.

6 Formation Evaluation

To test how our formation and role positioning system[5] affects the team's performance we created a number of teams to play against by modifying the base positioning system and formation of UT Austin Villa.

UT Austin Villa Base agent using the dynamic role positioning system described in Section 4 and formation in Section 3.
NoCollAvoid No collision avoidance.
AllBall No formations and every agent except for the goalie goes to the ball.
NoTeamwork Similar to AllBall except that collision avoidance is also turned off.
NoCommunication Agents do not communicate with each other.
Static Each role is statically assigned to an agent based on its uniform number.
Defensive Defensive formation in which only two agents are in the offensive group.
Offensive Offensive formation in which all agents except for the goalie are positioned in a close symmetric formation behind the ball.
Boxes Field is divided into fixed boxes and each agent is dynamically assigned to a home position in one of the boxes. Similar to system used in [4].

[5] Video demonstrating our positioning system can be found online at `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2011/html/positioning.html`

NearestStopper The *stopper* role position is mapped to nearest agent.

PathCost Agents add in the cost of needing to walk around known obstacles (using collision avoidance from Section 5), such as the ball and agent assuming the *onBall* role, when computing distances of agents to role positions.

PositiveCombo Combination of *Offensive*, *PathCost*, and *NearestStopper* attributes.

Table 2. Full game results, averaged over 100 games. Each row corresponds to an agent with varying formation and positioning systems as described in Section 6. Entries show the goal difference (row − column) from 10 minute games versus our base agent, using the dynamic role positioning system described in Section 4 and formation in Section 3, as well as the Apollo3D and CIT3D agents from the 2011 RoboCup China Open. Values in parentheses are the standard error.

	UTAustinVilla	Apollo3D	CIT3D
PositiveCombo	0.33 (.07)	2.16 (.11)	4.09 (.12)
Offensive	0.21 (.09)	1.80 (.12)	3.89 (.12)
AllBall	0.09 (.08)	1.69 (.13)	3.56 (.13)
PathCost	0.07 (.07)	1.27 (.11)	3.25 (.11)
NearestStopper	0.01 (.07)	1.26 (.11)	3.21 (.11)
UTAustinVilla	—	1.05 (.12)	3.10 (.12)
Defensive	-0.05 (.05)	0.42 (.10)	1.71 (.11)
Static	-0.19 (.07)	0.81 (.13)	2.87 (.11)
NoCollAvoid	-0.21 (.08)	0.82 (.12)	2.84 (.12)
NoCommunication	-0.30 (.06)	0.41 (.11)	1.94 (.10)
NoTeamwork	-1.10 (.11)	0.33 (.15)	2.43 (.12)
Boxes	-1.38 (.11)	-0.82 (.13)	1.52 (.11)

Results of UT Austin Villa playing against these modified versions of itself are shown in Table 2. The UT Austin Villa agent is the same agent used in the 2011 competition, except for a bug fix,[6] and so the data shown does not directly match with earlier released data in [9]. Also shown in Table 2 are results of the modified agents playing against the champion (Apollo3D) and runner-up (CIT3D) of the 2011 RoboCup China Open. These agents were chosen as reference points as they are two of the best teams available with CIT3D and Apollo3D taking second and third place respectively at the main RoboCup 2011 competition. The China Open occurred after the main RoboCup event during which time both teams improved (Apollo3D went from losing by an average of 1.83 to 1.05 goals and CIT3D went from losing by 3.75 to 3.1 goals on average when playing 100 games against our base agent).

Several conclusions can be made from the game data in Table 2. The first of these is that it is really important to be aggressive and always have agents near the ball. This finding is shown in the strong performance of the *Offensive* agent. In contrast to an offensive formation, we see that a very defensive formation used by the *Defensive* agent hurts performance likely because, as the saying goes, the best defense is a good offense. The poor performance of the *Boxes* agent, in which the positions on the field are somewhat static and not calculated as relative offsets to the ball, underscores the importance of being around the ball and adjusting positions on the field based on the current state of the game.

[6] A bug in collision avoidance present in the 2011 competition agent where it always moved in the direction away from the ball to avoid collisions was fixed.

The likely reason for the success of offensive and aggressive formations grouped close to the ball is because few teams in the league have managed to successfully implement advanced passing strategies, and thus most teams primarily rely on dribbling the ball. Should a team develop good passing skills then a spread out formation might become useful.

The *NearestStopper* agent was created after noticing that the *stopper* role is a very important position on the field so as to always have an agent right behind the ball to prevent breakaways and block kicks toward the goal. Ensuring that the *stopper* role is filled as quickly as possible improved performance slightly. This result is another example of added aggression improving game performance.

Another factor in team performance that shows up in the data from Table 2 is the importance of collision avoidance. Interestingly the *AllBall* agent did almost as well as the *Offensive* agent even though it does not have a set formation. While this result might come as a bit of surprise, collision avoidance causes the *AllBall* agent to form a clumped up mass around the ball which is somewhat similar to that of the *Offensive* agent's formation. For the strategy of all the agents running to the ball to work well it is imperative to have good collision avoidance. This conclusion is evident from the poor performance of the *NoTeamwork* agent where collision avoidance is turned off with everyone running to the ball, as well as from a result in [9] where the *AllBall* agent lost to the base agent by an average of .43 goals when both agents had a bug in their collision avoidance systems. Turning off collision avoidance, but still using formations, hurts performance as seen in the results of the *NoCollAvoid* agent. Additionally the *PathCost* agent showed an improvement in gameplay by factoring in known obstacles that need to be avoided when computing the distance required to walk to each target.

Another noteworthy observation from the data in Table 2 is that dynamically assigning roles is better than statically fixing them. This finding is clear in the degradation in performance of the *Static* agent. It is important that the agents are synchronized in their decision as to which mapping of agents to roles to use, however, as is noticeable by the dip in performance of the *NoCommunication* agent which does not use the voting system presented in Section 4.4 to synchronize mappings. The best performing agent, that being the *PositiveCombo* agent, demonstrates that the most successful agent is one which employs an aggressive formation coupled with synchronized dynamic role switching, path planning, and good collision avoidance. While not shown in Table 2, the *PositiveCombo* agent beat the *AllBall* agent (which only employs collision avoidance and does not use formations or positioning) by an average of .31 goals across 100 games with a standard error of .09. This resulted in a record of 43 wins, 20 losses, and 37 ties for the *PositiveCombo* agent against the *AllBall* agent.

7 Summary and Discussion

We have presented a dynamic role assignment and formation positioning system for use with autonomous mobile robots in the RoboCup 3D simulation domain — a physically realistic environment that is partially observable, non-deterministic, noisy, and dynamic. This positioning system was a key component in UT Austin Villa[7] winning the 2011 RoboCup 3D simulation league competition.

For future work we hope to add passing to our strategy and then develop formations for passing, possibly through the use of machine learning. Additionally we intend to look into ways to compute f_v more efficiently as well as explore other potential functions for mapping agents to role positions.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. Thanks especially to UT Austin Villa 2011 team members Daniel Urieli, Samuel Barrett, Shivaram Kalyanakrishnan, Michael Quinlan, Nick Collins, Adrian Lopez-Mobilia, Art Richards, Nicolae Ştiurcă, and Victor Vu. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030). Patrick MacAlpine is supported by a NDSEG fellowship.

References

1. Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., Shimada, S.: Robocup rescue: search and rescue in large-scale disasters as a domain for autonomous agents research. In: Proc. of 1999 IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC). Volume 6. (1999) 739 –743 vol.6
2. Wurman, P.R., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI Magazine **29** (2008) 9–20
3. Kalyanakrishnan, S., Stone, P.: Learning complementary multiagent behaviors: A case study. In: RoboCup 2009: Robot Soccer World Cup XIII, Springer (2010) 153–165
4. Stone, P., Veloso, M.: Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial Intelligence **110** (1999) 241–273
5. Reis, L., Lau, N., Oliveira, E.: Situation based strategic positioning for coordinating a team of homogeneous agents. In Hannebauer, M., Wendler, J., Pagello, E., eds.: Balancing Reactivity and Social Deliberation in Multi-Agent Systems. Volume 2103 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2001) 175–197
6. Chen, W., Chen, T.: Multi-robot dynamic role assignment based on path cost. In: 2011 Chinese Control and Decision Conference (CCDC). (2011) 3721 –3724
7. Lau, N., Lopes, L., Corrente, G., Filipe, N.: Multi-robot team coordination through roles, positionings and coordinated procedures. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009). (2009) 5841 –5848
8. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011 3D Simulation Team report. Technical Report AI11-10, The Univ. of Texas at Austin, Dept. of Computer Science, AI Laboratory (2011)
9. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012). (2012)

[7] More information about the UT Austin Villa team, as well as video highlights from the 2011 competition, can be found at the team's website: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/`