CommLang: Communication for Coachable Agents

John Davin, Patrick Riley, and Manuela Veloso

Carnegie Mellon University,
Computer Science Department, Pittsburgh, PA 15232
jdavin@cmu.edu, {pfr, mmv}@cs.cmu.edu

Abstract. RoboCup has hosted a coach competition for several years creating a challenging testbed for research in advice-giving agents. A coach agent is expected to advise an unknown coachable team. In RoboCup 2003, the coachable agents could process the coach's advice but did not include a protocol for communication among them. In this paper we present CommLang, a standard for agent communication which will be used by the coachable agents in the simulation league at RoboCup 2004. The communication standard supports representation of multiple message types which can be flexibly combined in a single utterance. We then describe the application of CommLang in our coachable agents and present empirical results showing the communication's effect on world model completeness and accuracy. Communication in our agents improved the fraction of time which our agents are confident of player and ball locations and simultaneously improved the overall accuracy of that information.

1 Introduction

In a multi-agent domain in which agents cooperate with each other to achieve a goal, communication between those agents can greatly aid them. Without communication, agents must rely on observation-based judgments of their collaborator's intentions [1, 2]. Communication enables agents to more explicitly coordinate planning decisions and execute more intelligent group behaviors.

In domains where agents have only partial knowledge of their environment, communication is equally important for sharing state information with other agents. In most robotic applications, agents have only a limited view of the world. However, by communicating with a team, they can achieve a much broader and more accurate view of the world.

Communication is an important aspect of many multi-agent systems, especially those involving teamwork. Recently, several formal models have been proposed to capture the decisions which agents face in communicating [3, 4].

In RoboCup simulation soccer, teams of eleven agents compete against an opposing team. A coach agent with global world information may advise the team members. The soccer players are permitted to communicate through auditory messages, but are limited to ten character messages. In order to maximize the utility of communication, particularly with messages limited to a short length, we needed an architecture that could flexibly represent a number of different types of information, and at the same time be easy to implement so that it could be adopted as a standard.

Soccer simulation already has a language for communicating between the coach and the players, named CLang, but no standardized language for inter-agent communication has previously existed. While past work has developed general purpose agent communication languages like KQML [5] and FIPA-ACL [6, 7], these languages are not directly applicable here because of the extremely limited bandwidth between the players.

Other soccer simulation teams have used fixed communication schemes in which they always send the same information — such as ball position and player position. While this is beneficial, we believe that communication can be better utilized.

In this paper we will present CommLang, a communication language that we developed that has been adopted as the standard for coachable agents in the 2004 RoboCup competition. We will also describe the algorithms we used to implement CommLang in our coachable team, and show empirical evidence that communication improved the accuracy of the world model in our agents.

2 The CommLang Communication Standard

We developed the CommLang communication standard to provide a means of communicating between coachable agents. However, the protocol is also useful for other simulation agents. It defines representations for different types of information to be transmitted over a limited bandwidth communication channel. The standard specifies how to encode information into character strings for use in the soccer server's character-based communication messages.

CommLang addresses only the composition and encoding of communication messages. We do not specify which specific types of information should be sent, nor how often messages should be sent. We also do not specify how the received information should be used, except that there should be some reasonable utilization of the information. The messages are useful because they contain meaning about what the sending agent's beliefs are.

The soccer simulation server is configured to allow messages of ten characters (out of a set of 73) to be sent each game cycle. Players receive one audio message per cycle as long as a teammate within hearing range (50 meters) uttered a message. We encode communication data into a character representation in order to achieve full use of the 73 character range. After messages are received by a player, they must be decoded to extract the numeric data from the character string.

2.1 Message Types

CommLang defines a set of message types which each represent a specific type of information. A single communication message can be composed of multiple

Table 1. Message types used to encode information

Message Type	Syntax	Cost (characters)
Our Position	[0, X, Y]	3
Ball Position	[1, X, Y, #cycles]	4
Ball Velocity	[2, dX, dY, #cycles]	4
We have ball	[3, player#]	2
Opponent has ball	[4, player#]	2
Passing to player#	[5, player#]	2
Passing to coordinate	[6, X, Y]	3
Want pass	[7]	1
Opponent(player#) Position	[8+player#-1, X, Y, #cycles]	4
Teammate(player#) Position	[19+player#-1, X, Y, #cycles]	4

Table 2. Data types used in composing messages. Each data type uses one character of a message

Data Type	Description	Range	Precision
X	x coordinate of a position	[-53, 53)	1.45
Y	y coordinate of a position	[-34, 34)	0.93
dX	x component of a velocity	[-2.7, 2.7)	0.074
dY	y component of a velocity	[-2.7, 2.7)	0.074
#cycles	number of cycles since data last observed	[0, 72)	1
player#	a player number	[0, 11]	1
msg type ID	the message ID	[0, 29]	1

message types. The available message types and their syntax are listed in Table 1. All message types begin with a message type ID which allows us to identify the type of message and the number of arguments that will follow.

Each message type is made up of discrete units of data referred to as data types. In the interest of simplicity and ease of implementation, each unit of information (data type) uses one character of the message string. The data types are shown in Table 2. Each data type has a precision that is determined by the data range that is represented by the 73 characters. For the data types with floating point ranges, the precision is equal to the maximum loss of accuracy that can occur from encoding the data.

This design is flexible in that auditory messages sent to the server may be variable lengths — they can be composed of any number of message types, as long as they fit within the ten character limit.

Note that the teammate and opponent position message types do not have a single message type ID. Rather, the player number of the player whose position we are communicating is encoded in the message type ID. This allows us to reduce the length of the teammate and opponent message types by including the player number in the message ID number rather than as a separate argument.

Since the message type IDs do not yet use the full 73 character range, it would be possible to modify some of the other message types to encode the

```
char validchars[] = $''0123456789abcdefghijklmnopqrstuvwxyz$
$ABCDEFGHIJKLMNOPQRSUVWXYZ().+-*/?<>\_''$;
NUMCHARS = 73

char getChar( int $i$ ):
    Returns validchars[NUMCHARS-1] if $i \geq NUMCHARS$
    Returns validchars[0] if $i < 0$
    Returns validchars[i] otherwise
int getIndex( char $c$ ):
    Returns the index of character $c$ in the validchars array.
```

Fig. 1. Character conversion functions. The validchars array contains the characters that are permitted in soccer server say messages

player number in the ID. However, we chose not to do this because it would quickly use up the ID range, which may be needed in the future if new message types are added to the standard.

The player number data type ranges from 1 to 11 to indicate a player number. However, in the "We have ball" and "Opponent has ball" message types, a zero may be sent as the player number to indicate that we know which team has the ball, but do not know the player number.

Player and ball positions that are sent in messages should be the position that the player predicts the object is at in the current cycle. The #cycles data type should be equal to the number of cycles ago that the player received data about the object.

2.2 Character Conversions

The task of converting numeric information such as a field position to a character that can be accepted by the soccer server is handled by a character array and lookup functions. A specification for these functions is shown in Fig. 1. These functions are used to assist with encoding and decoding, as described in the next two sections.

2.3 Encoding Messages

We encode communication messages by converting numeric information into character encodings according to the equations in Table 3. The encoded strings of each message type being used are then concatenated into one string and sent to the soccer server as a say message.

The arithmetic in the encoding functions is done using standard floating point operations, and the final integer result is the floor of the float value.

Table 3. Data Type encoding functions

Data type value	Encoded character value
X	getChar((X+53)/106 * NUMCHARS)
Y	getChar((Y+34)/68 * NUMCHARS)
dX	getChar((dX+2.7)/5.4 * NUMCHARS)
dY	getChar((dY+2.7)/5.4 * NUMCHARS)
#cycles	getChar(#cycles)
player#	getChar(player#)
msg type ID	getChar(msg type ID)

Table 4. Data Type decoding functions

Data type of character	Decoded numeric data
X	(getIndex(X)/NUMCHARS * 106) - 53
Y	(getIndex(Y)/NUMCHARS * 68) - 34
dX	(getIndex(dX)/NUMCHARS * 5.4) - 2.7
dY	(getIndex(dY)/NUMCHARS * 5.4) - 2.7
#cycles	getIndex(#cycles)
player#	getIndex(player#)
msg type ID	getIndex(msg type ID)

2.4 Decoding Messages

When a coachable agent receives a message from a teammate, it decodes the message by using the message type IDs to identify which message types are included, and then decodes the data types according to the definitions in Table 4.

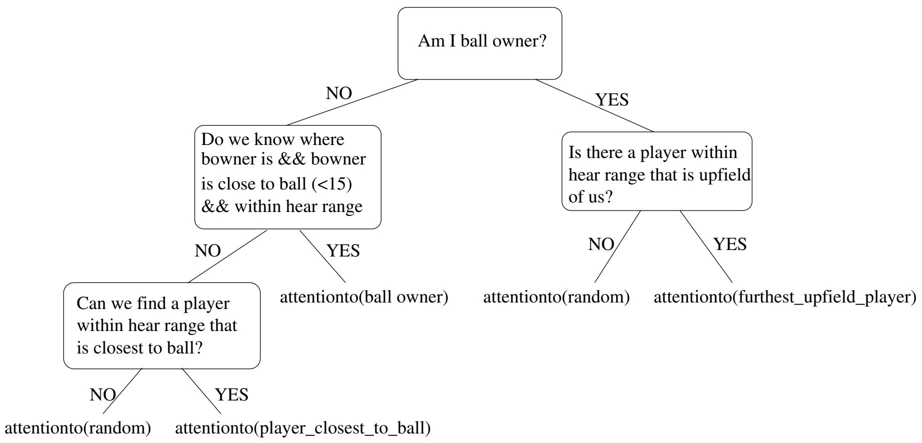
Am I ball owner?
NO
YES
Do we know where bowner is && bowner is close to ball (<15) && within hear range
Is there a player within hear range that is upfield of us?
NO
YES
NO
YES
attentionto(ball owner)
attentionto(random)
attentionto(furthest_upfield_player)
Can we find a player within hear range that is closest to ball?
NO
YES
attentionto(random)
attentionto(player_closest_to_ball)

Fig. 2. The decision process used to decide which teammate to listen to. "Attentionto(random)" indicates that we allow the server to randomly choose a message for us

The message must be sequentially processed to extract each message type based on its message type ID.

Since our numeric data was encoded in a character representation, the decoded data may be slightly different from the original data due to the limited precision of each data type (see Fig. 2).

2.5 CommLang in RoboCup 2004

To assist other teams in using CommLang, we have released our encoding and decoding library at: *http://www-2.cs.cmu.edu/~robosoccer/simulator/*. The library is easily extensible — if new types of messages are added to the standard, a new version of the library can be released that is backwards compatible with previous versions so that teams' existing code will still operate.

The RoboCup 2004 simulation league rules require that coachable agents conform to the communication standard as described in this paper. Since simulation games will take place with teams composed of a mix of coachable agents from different research teams, it is necessary for all agents to use the same standard in order to understand each other. It is our belief that use of CommLang will improve cooperation between coachable agents and lead to better performance in the competition.

3 Implementation of Agent Communication

We have implemented coachable agent communication in the CMU Wyverns coachable team which was used in RoboCup 2003. The version of our agents used with our communication implementation also includes a number of improvements that were made after the competition. We had two primary goals in mind while adding communication to our agents:

1. To maximize the utility of agent communication by supporting all of the message types and by increasing the likelihood that communicated information would be useful to teammates.
2. To create a flexible architecture for experimenting with different strategies of communication.

3.1 Sending Message Frequency

In our implementation, every player broadcasts a say message each cycle. We chose this approach because it insures that every player receives a message every cycle. Simulation soccer teams have sometimes used coordinated strategies of having designated players speak each cycle. Our approach has the advantage of allowing us to listen to any player each cycle and attempt to select the most useful player to listen to. The trade-off is that we may not listen to a player for an extended period of time even though it might have something important to say.

Another reason for using this message sending strategy is that the simulation soccer coachable agents are mixed in with agents from other developers to make up a coachable soccer team. Since the CommLang standard does not make any stipulations about message frequency, our agents can not make any assumptions about the sending strategy of teammates.

3.2 Receiving Messages

We use the soccer server "attentionto" command to focus our attention on a teammate of our choosing. This is the player that we will receive say messages from if it is within range. This control gives us the ability to establish preferences towards listening to a certain player. For example, it is often preferable to listen to the ball owner.

The determination of which player to listen to is handled by a decision tree, which is shown in Fig. 2.

3.3 Selecting Message Composition

The most complex part of our implementation is the process for deciding what message types to include in the messages that the player sends. We typically have room to fit three or four message types within the ten character message size limit. The strategy we use to choose those three or four message types is an important factor in determining how useful the message is. For example, if a teammate needs to know where the ball is, but we generally send our player position instead, our messages will have little value.

Our implementation uses a randomized scheme to select the message composition. Each message type is assigned a weight which influences the likelihood of using it in the final message. These weights are intended to reflect the overall utility of the message type. Message types that we think are highly useful in most situations are assigned higher values while less important message types

Table 5. The predicate functions and weights for each message type, as currently configured in our agents

	Predicates	Weight
OurPos	none	0.5
BallPos	Confident of ball position	0.5
BallVel	Confident of ball velocity	0.3
WeHaveBall	Confident of ball info AND We own ball currently	0.2
OppHasBall	Confident of ball info AND Opponent team owns ball	0.2
PassToPlayer	I executed a pass action within the last 5 cycles	1.1
PassToCoord	I executed a pass action within the last 5 cycles	0.8
WantPass	I'm not ball owner AND I'm close to opponent goal AND I have a good goal shot	1.0
OppPos(pnum)	Confident of position of opponent player #pnum	0.4
TeammatePos(pnum)	Confident of position of teammate player #pnum	0.4

```
$M$ := Set of all message types
$C$ := empty message
$\forall\ m \in M,\ m.p = getWeight(m)$
remove $m_i \in M$ if $predicate(m_i)$ = false Or $m_i.p = 0.0$
foreach $m \in M$, from largest $m.p$ to smallest
    If( $m.p \geq 1.0$ )
        insert $m$ in $C$
        remove $m$ from $M$
Normalize such that $\{m.p \mid m \in M\}$ is a probability distribution
while ( $M \neq \emptyset$ )
    Choose $m_i$ from distribution defined by $m_i.p$
    If( $m_i.size + C.size \leq$ MAX_MESSAGE_SIZE )
        insert $m_i$ in $C$
    remove $m_i$ from $M$
    Renormalize probability distribution in $\{m.p \mid m \in M\}$
return $C$
```

Fig. 3. Algorithm for choosing message composition

receive lower values. We also define predicates to filter out any message types that are not applicable at the current time. For example, PassToPlayer is only applicable when we have the ball and are passing to a teammate. Table 5 lists the predicates and weights for each message type.

The weights are generally within the range 0.0 to 1.0, but may also go over 1.0. Values over 1.0 serve to guarantee that we use the message type, as long as there is sufficient space. Weights of 0.0 indicate that the message type will never be used.

The main component of the selection algorithm is shown in Fig. 3. We first automatically select any message types that have a weight of 1.0 or greater. Then, the main loop chooses the next message type to include based on the probability distribution over the weights of the remaining message types.

This probabilistic selection method allows us to define preferences towards using particular message types while at the same time insuring that we do not use the same message composition every time.

In addition to the weighting strategy that is in use now, for testing we have also implemented a uniform probability distribution and a distribution for selecting messages from a fixed subset of message types (specifically, it can be used to send only "Our Pos" and "Ball Pos" types).

3.4 Processing Communicated Information

When we receive messages, we process the information using our decoding library routines and then integrate the information into the world model. Data in the world model is replaced with communicated information only if the information is more recent than the knowledge we already have. Therefore, we implicitly trust our teammates' communication information to be accurate and we store it in the same location as our own sensor information. We use the world model from the 2002 UvA Trilearn team [8], which is the team that was used as the original base for our coachable agents.

Three of the message types — PassToPlayer, PassToCoord, and WantPass — are messages that primarily communicate information about a teammate's intentions rather than perceptions of the world's state. Therefore, we respond to these message types by executing new actions as appropriate. If a PassToPlayer message indicates a teammate is passing to us, we immediately attempt to intercept the ball. Similarly, if a PassToCoord message indicates a teammate is passing to a position close to us, we assume the pass is intended for us and attempt to intercept. If a teammate indicates WantPass, we attempt to pass to that player if we possess the ball or obtain possession within the next 5 cycles. These actions can take precedence over coach advice.

4 Empirical Evaluation

After implementing communication in the Wyverns coachable agents, we ran tests of the agents to assess the impact of communication on their world model. Two sets of ten games were run, with each game lasting for 3000 cycles. In the experimental set, communication was used, and in the other set communication was not used.

In both sets, the opponent team consisted of the Wyverns coachable agents as publicly released after RoboCup 2003. The other team, which contained the communication support, was an improved version of the Wyverns players with a number of bug fixes and skill improvements.

Both teams were advised by the CMU Owl coach [9]. Note that the improved Wyverns won most of the games, with 26 total goals, versus 4 goals for the original Wyverns. Therefore the experimental players were often in offensive positions, but they also were sometimes forced into defensive formations.

Since the team's formation can affect the frequency that we see other players, we will note that our offensive players were numbers 2, 7, and 8, our midfielders were 3, 5, and 6, the defensive players were 4, 9, 10, and 11, and player 1 is always the goalie.

4.1 Improvements in Player Confidence

The players' confidence in the ball and player locations was recorded during the games. As currently configured, players are confident in a teammate or opponent location when they have seen the player (or updated the player's position based on communicated data) within the last 12 cycles. Players are confident of the ball's location when they have seen or updated it within the last 6 cycles.

Figure 4 shows the mean percentage of time that the players were confident of the location of the ball and teammates. The communication group had higher confidence rates for all the objects, with most of the increases statisti-

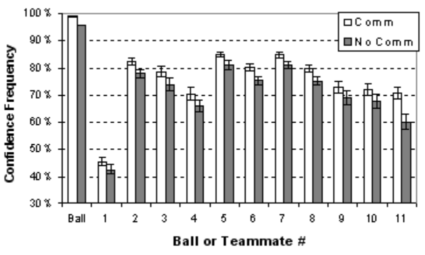

Fig. 4. Confidence frequencies for the ball and teammates. Error bars indicate 95% confidence intervals

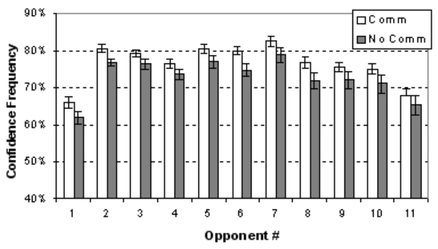

Fig. 5. Confidence frequencies for the opponent players

cally significant. Similar results were seen for our perceptions of the opponent players (Fig. 5).

Although this improvement was expected, it is still a useful confirmation of one of communication's benefits. Confidence frequencies play a role in many of an agent's behaviors - for example, our players can not pass to a teammate unless they are confident in its location. Therefore, increases in confidence frequencies can directly influence agent behavior.

4.2 Improvements in Positional Error

The other statistic compiled from the games was the positional error of players in our world model. The positional error values are the difference between where a player thinks a teammate or opponent is, and where that player actually is. The actual positions were obtained using the soccer server's full state mode. Errors were only counted during cycles when the player was confident in the location of the relevant teammate or opponent.

Figure 6 shows the players' mean world model errors for each of their teammates, as well as the ball. The 95% confidence intervals could not be displayed on the figure, but were small enough for the results to be statistically significant.

For most player positions, communicating players had lower mean error than non-communicating players. This was especially noticeable with the estimates of ball position, and the position of player 1 (the goalie).

Since position errors were only collected when the player was confident in the location of the teammate, more error data was sampled in the communication set than the non-communication set because communication increases the frequency at which players are confident of teammate locations. As seen in Fig. 6, the error for communicating players actually increased on a small number of the player-player comparisons. Since communicating players are confident of teammate locations more often, this may cause more error to accumulate in some cases. For example, if an offensive player is out of visible range of the goalie, but receives the goalie's position via communication, the player will still be confident of the goalie's location ten cycles later, but the estimated position could be significantly different from the actual position.

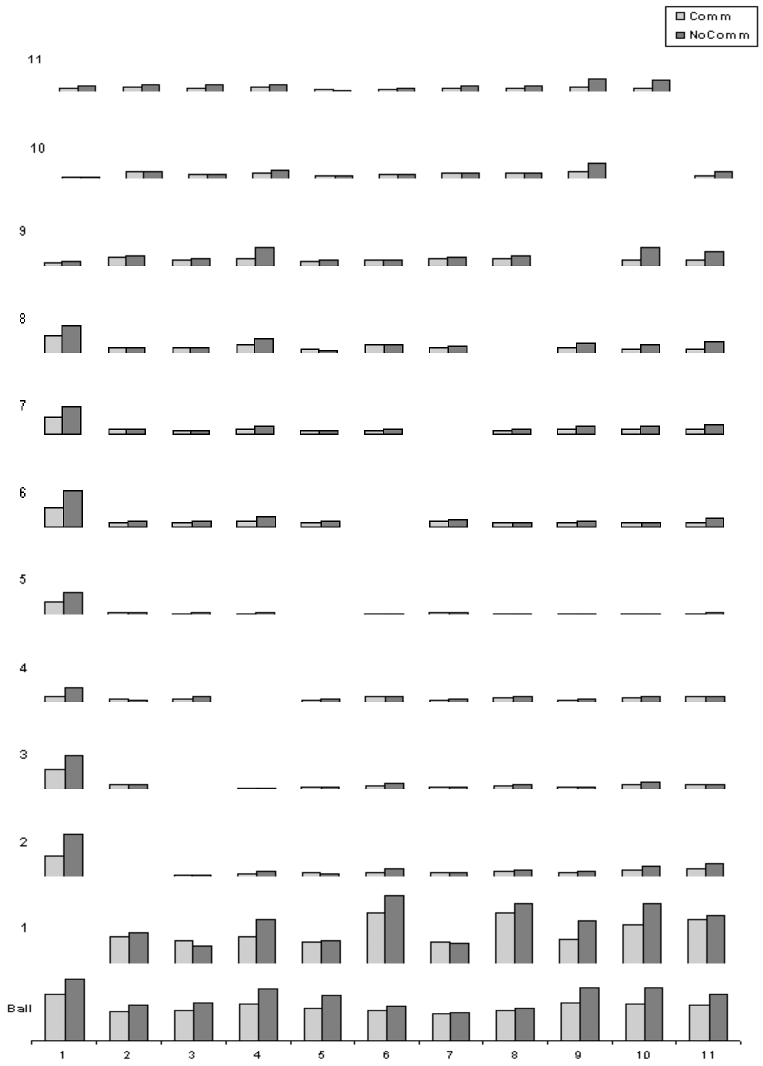

Fig. 6. Mean world model error (in meters) of ball and teammate positions. The numbers along the x axis indicate our players. Each column of bars above represents the positional errors in that player's world model. The players' errors for themselves (the diagonal on the graph) are displayed as zero because self position errors were not recorded (communication does not affect those values). The charts for our perceptions of the teammates listed on the y axis are scaled to a max error value of 20 (meters), and the row for the ball is scaled to a max of 3

In addition, communication can also change the behavior of the agents in subtle ways. For example, if players know the locations of their teammates more often (due to communication), they may decide to scan the field less often.

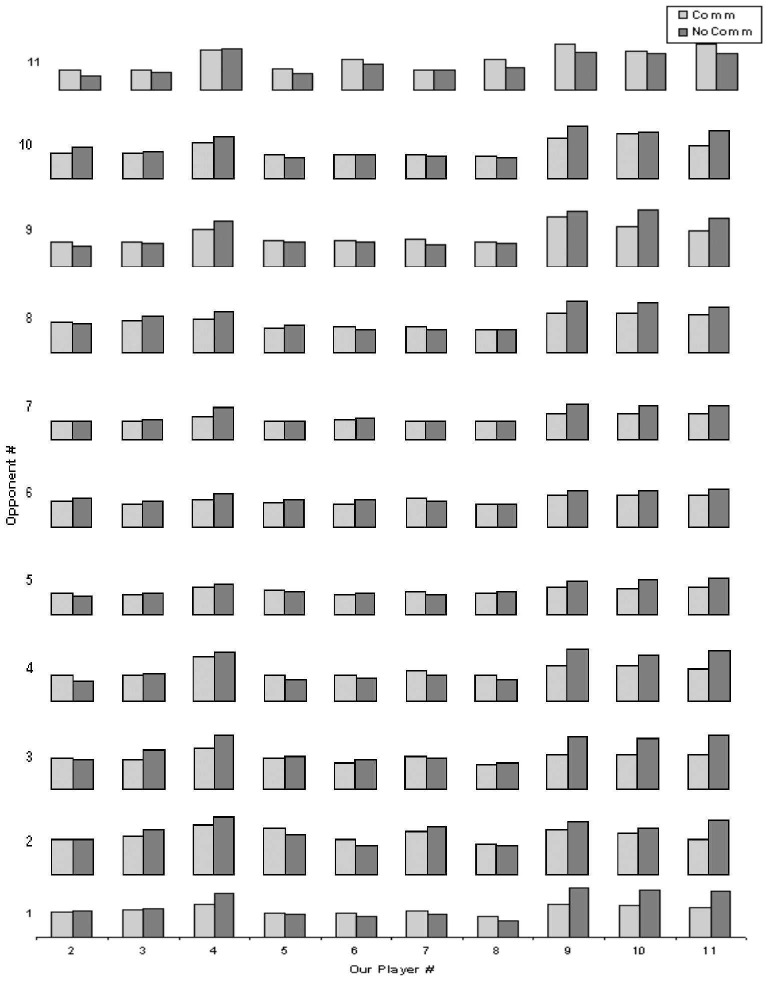

Fig. 7. Mean world model error of opponent positions. The column for our goalie's world model (player 1 on x axis) was omitted because the errors were beyond the scale of the graph — our goalie has high error for the opponents since it usually can not see them. The rows for opponents 2-11 are scaled to a maximum of 4, and the row for opponent 1 is scaled to 18

Therefore, changes in the accuracy of player world models are not necessarily only a direct result of communication — additional factors may include changes to scanning frequency, and other differences.

Communicating players had lower error values for their estimated position of the ball. Their mean ball error was 22.6% lower than non-communicating players. As seen previously in Fig. 4, communicating players were also confident of the ball location 3.3% more often than non-communicating players.

For the agents' estimation of opponent positions (Fig. 7), most of the changes were less significant. This could indicate that our agents were able to get better information about the opponents simply through visual sensors.

However, our players 4, 9, 10, and 11, which are defensive players, all improved their estimation of most of the opponent positions (see columns 4, 9, 10, 11 in the figure). Since our defensive players rarely get good visual information about the opponents since they are far away most of the time, communication helps them a great deal in this regard.

The empirical results with player confidences and positional errors are important because they show that communication improves not only the frequency with which a player is confident of ball and player locations, but also the accuracy of the player's position estimate. This improved accuracy and confidence has implications in many areas. For instance, ball positions are used in calculating interception trajectories, ball handling strategies, and in determining which advice from the coach is applicable.

In summary, coachable agent communication as implemented in the Wyverns agents increases the frequency with which players are confident of the ball and other players. It also decreases the error in the players' estimation of where the ball, teammates, and opponents are.

5 Conclusion and Future Work

The CommLang standard for RoboCup agent communication described in this paper is a flexible and easily extendable language for communicating between agents. It allows a variety of information to be exchanged, expanding the potential for inter-agent cooperation. All coachable agents will need to use this standard and we encourage other soccer simulation teams to do so also. Use of this standard by simulation teams would improve agent interoperability, facilitating interesting mixed-team pickup games.

Our implementation of coachable agent communication in the Wyverns agents is a versatile architecture that provides control over the composition of messages, the source of received messages, and the incorporation of received information into our world model. It is designed to construct communication messages that maximize the usefulness of a limited bandwidth channel of communication.

In past studies of previous versions of the simulated soccer environment [10], communication had an overwhelmingly positive effect. However, at that time, the communication bandwidth was over 50 times the bandwidth allowed currently (512 characters compared to 10 characters). The players in general did not need to reason about what information to include in each communication.

We have presented an algorithm for determining message composition and shown that our initial parameters lead to improvements in world model completeness and accuracy. We have not yet explored what settings (e.g. weights of the message types) yield the largest improvements and this is an interesting avenue for future work. Further, reasoning about the tradeoffs involved in limited bandwidth communication could lead to improvements in some performance measures.

In the future we may wish to consider adding new message types to the communication standard. One general type of message which is not currently in the standard is the information request form (such as exists in KQML [5]). These messages would allow agents to request from their teammates state information such as the ball location.

Another area meriting consideration is the option to maintain information about communication from earlier in the game. We could consider using earlier communication to learn who to listen to — if one of our teammates often sends more useful information than other agents, we could focus on that agent more frequently.

Through our empirical evaluation, we found that communication improved the world model knowledge of our coachable agents. We believe there is significant research potential in this area to determine how communication between agents can be used to the greatest advantage.

References

1. Doyle, R., Atkinson, D., Doshi, R.: Generating perception requests and expectations to verify the executions of plans. In: AAAI-86. (1986)
2. Kaminka, G., Tambe, M.: I'm OK, you're OK, we're OK: Experiments in distributed and centralized socially attentive monitoring. In: Agents-99. (1999)
3. Xuan, P., Lesser, V., Zilberstein, S.: Communication decisions in multi-agent markov decision processes: Model and experiments. In: Agents-2001. (2001) 616–623
4. Pynadath, D., Tambe, M.: The communicative multiagent team decision problem: Analyzing teamwork theories and models. Journal of Artificial Intelligence Research **16** (2002) 389–423
5. Labrou, Y., Finin, T.: Semantics and conversations for an agent communication language. In: IJCAI-97, Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1997) 584–591
6. FIPA: FIPA ACL message structure specification. `http://www.fipa.org/specs/fipa00061/XC00061E.html` (2001)
7. FIPA: FIPA communicative act library specification. `http://www.fipa.org/specs/fipa00037/XC00037H.html` (2001)
8. Kok, J.R., Spaan, M.T.J., Vlassis, N.: Multi-robot decision making using coordination graphs. In: Proceedings of the 11th International Conference on Advanced Robotics. (2003)
9. Riley, P., Veloso, M.: Advice generation from observed execution: Abstract Markov decision process learning. In: AAAI-2004. (2004) (to appear)
10. Riley, P.: Classifying adversarial behaviors in a dynamic, inaccessible, multi-agent environment. Technical Report CMU-CS-99-175, Carnegie Mellon University (1999)