Distributed, Play-Based Coordination for Robot Teams in Dynamic Environments

Colin McMillen and Manuela Veloso

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, U.S.A.
{mcmillen,veloso}@cs.cmu.edu

Abstract. We present refinements to our previous work on team communication and multi-robot world modeling in the RoboCup legged league. These refinements put high priority on the communication of task-relevant data. We also build upon past results within the simulation and the small-size leagues and contribute a distributed, play-based role assignment algorithm. This algorithm allows the robots to autonomously adapt their strategy based on the current state of the environment, the game, and the behavior of opponents. The improvements discussed in this paper were used by CMDash in the RoboCup 2005 international competition.

1 Introduction

A common goal of distributed autonomous robotic systems is the development of teamwork and coordination strategies. The benefits of adding multiple robots to a system, such as increased performance and reliability, have been demonstrated in many different situations. However, depending on the domain and the task, different sorts of approaches might be needed. We are interested in multi-robot coordination in domains with high uncertainty and dynamic environments.

In this paper, we present two main contributions: refinements to our previous work in multi-robot world modeling and a novel approach to role assignment. Both contributions are discussed in the context of the RoboCup four-legged league [1], in which two teams of four Sony AIBO robots compete in a robot soccer game. This domain presents many challenges, including: full robot autonomy, distributed robot team control, limited individual robot perception, the presence of robot adversaries, task-dependent temporal constraints, and high communication latency. In this paper, we contribute a new distributed *play-based* system that equips the robots with *plays* – alternative teamwork strategies. This method was developed to overcome limitations of previous approaches. In particular, it assigns roles to robots in a fault-tolerant manner that minimizes role switching and synchronization problems. This paper is explicitly targeted at the very challenging issues posed by the RoboCup four-legged league; however, our approach is designed to be relevant to general multi-robot domains that share some of the challenging features of robot soccer.

In section 2, we discuss our enhancements to communication and multi-robot world modeling. Section 3 introduces our play-based teamwork strategy, and

refers to some positive experimental results. Section 4 presents our conclusions. Related work is discussed throughout the paper as needed.

2 Communication Strategies

Many multi-robot teams make use of communication for world state sharing. Due to the AIBOs' limited perception range and the extensive object occlusion in the RoboCup environment, teams can benefit greatly by building a shared world model. A common approach, used by our team in the past [2], is to have each robot periodically broadcast a packet containing all the shared information, such as the robot's current position, its best estimate of the ball position, and the positions of detected opponent robots. However, some domain information (such as the position of the ball) is inherently more important to the success of the team than other types of information. We have therefore developed a factored communication strategy. In this strategy, there are several different types of message, containing different pieces of information. We can then independently choose the transmission rate for each type of message. This communication strategy allows the robots to respond more quickly to important events (such as a change in the ball's position) without the need to transmit a large message over the communication network. In this section, we present a brief overview of the world-modeling information our robots shared in the RoboCup 2005 competition.

2.1 Ball Messages

These messages are sent by all robots to indicate important information about the status of the ball. Each message contains the following information:

- Ball state. This is a new feature that was added to our world model for RoboCup 2005. This can take on one of the following values:
  - LOST: No reliable estimate of the ball's location is available.
  - VISIBLE: The ball is currently seen.
  - POSSESSION: The ball is believed to be in the possession of the robot (i.e., the robot has grabbed the ball and is lining up for a kick.)
  - NOTINFOV: The ball is not currently seen, but is not expected to be seen because it is outside the robot's field of view. This happens (e.g.) when a robot takes its view off the ball to look at a localization marker.
  - INFOVBUTMISSING: The ball is not currently seen, even though the robot believes it is looking at the ball's location.
  - INFOVBUTOCCLUDED: The ball is not currently seen, but the robot believes that an object (such as another robot) is occluding the ball.

  We now send these symbolic ball states instead of numerical confidence values. These symbolic values allow the team to more accurately characterize the true state of the ball.
- Whether the robot transmitting the message believes that it is lost. This is determined by thresholding the robot's localization uncertainty.

- The global position of the ball. Global ball position estimates are not used from any robot that claims to be lost, since a lost robot is very likely to project its local ball estimate to an incorrect global position.
- The position of the ball relative to the robot. If the ball is very close to the reach of a robot, and that robot intends to kick the ball, the robot's teammates should avoid interfering with the kick, even if the robot believes that it is lost. The transmission of relative ball locations allows robots to back off in this situation, without the need to rely on visually seeing the teammate near the ball.

Since the location of the ball is of utmost importance to the proper functioning of the team, the ball messages are sent frequently. A robot will send a new ball message every 1/8 second if it has a good ball hypothesis and is not lost. If the robot becomes lost or does not have a valid ball hypothesis, it waits a while longer to see if the situation improves. This is done because a valid global ball location provides more valuable information to teammates. After 1/4 second has passed, however, the robot sends a ball message regardless of the circumstances.

2.2 Status Messages & Intentions

Another type of message is the status message. Status messages are sent by each robot at periodic intervals (typically 4 Hz). They include the robot's current position and angle (as reported by localization) as well as the current "intention" of the robot. Intention is a very important concept that we have added to our teamwork this year. When a robot is very close to the ball, its teammates should stay out of the way, to ensure that they do not interfere with the attacker's actions. However, there are specific times when nearby robots might not be intending to go for the ball. In these cases, the teammates should not back away just because another robot is near. The intention of the robot is determined by the robot's top-level behavior, and can take on any of the following values:

- ATTACK: the robot intends to approach the ball and manipulate it.
- WAIT: the robot does not intend to approach the ball. This happens when a robot is returning to position or is searching for the ball.
- YIELD: the robot would intend to approach the ball, except that it is yielding to a teammate instead.

2.3 Periodic Messages

Periodic messages are provided as a form of robustness to failure. The information contained in periodic messages allow the robots to determine when network failures have occurred, when a teammate has crashed, or other anomalous events have occurred. The team can then take appropriate actions to ensure that team play degrades gracefully in the presence of failure. The periodic message is typically sent at a rate of 1 Hz.

3 Distributed Play-Based Role Assignment

It is our experience that it is rather challenging to generate or learn a team control policy in complex, highly dynamic (in particular adversarial), multi-robot domains. Therefore, instead of approaching teamwork in terms of a mapping between state and joint actions [3], we follow a *play-based* approach, as introduced by Bowling *et al.* [4]. A play-based approach allows us to handle the domain challenges introduced in section 2. A play specifies a *plan* for the team; i.e., under some applicability conditions, a play provides a sequence of steps for the team to execute. Multiple plays can capture different teamwork strategies, as explicit responses to different types of opponents. Bowling showed that play selection weights could be adapted to match an opponent. Plays also allow the team to reason about the zero-sum, finite-horizon aspects of a game-playing domain: the team can change plays as a function of the score and time left in the game. Our play-based teamwork approach ensures that robots do not suffer from hesitation nor oscillation, and that team performance is not significantly degraded by possible periods of high network latency. We believe that ours is the first distributed play-based teamwork approach within the context of the RoboCup four-legged league.

3.1 Plays

A *play* is a team plan that provides a set of roles, which are assigned to the robots upon initiation of the play. Bowling [4] introduced a play-based method for team coordination in the RoboCup small-size league. However, the small-size league has centralized control of the robots. One of the significant contributions of our work is the development of a play system that works in a distributed team. The play language described by Bowling assumes that the number of robots is fixed, and therefore always provides exactly four different roles for the robots. In another extension to Bowling's work, our plays also specify which roles are to be used if the team loses some number of robots due to penalties or crashes. This extension to the role-assignment aspects of Bowling's play language allows the team to robustly adapt to the loss or penalization of team members without the need for additional communication.

Our play language itself is also strongly inspired by the work of Bowling. Our language allows us to define *applicability conditions*, which denote when a play is suitable for execution; what *roles* should be assigned when we have a specific number of active robots on the team; and a *weight*, which is used to decide which play to run when multiple plays are applicable.

Applicability. An applicability condition denotes when a play is suitable for execution. Each applicability condition is a conjunction of binary predicates. A play may specify multiple applicability conditions; in this case, the play is executable if any of the separate applicability conditions are satisfied.

Roles. Each play specifies which roles should be assigned to a team with a variable number of robots by defining different `ROLES` directives. A directive

applies when a team has k active robots, and specifies the corresponding k roles to be assigned. If a robot team has n members, each play has a maximum of n `ROLES` directives. Since our AIBO teams are composed of four robots, our plays have four `ROLES` directives.

Weight. Weight is used to decide which play to run when multiple plays are applicable. In our current algorithm, the play selector always chooses the applicable play with greatest weight. Future work could include choosing plays probabilistically based on the weight values or updating the weights at execution time to automatically improve team performance. *Playbook adaptation* of this sort was introduced by Bowling for the small-size league [4].

Unlike the work of Bowling, we do not have `DONE` or `TIMEOUT` keywords that specify when a play is complete. Rather, the play selector runs continuously, and each play is considered to be complete as soon as a different play is chosen. This may happen because the current play is no longer applicable or because another play with greater weight has recently become applicable. Each predicate used in an applicability condition is designed with some hysteresis, such that it is not possible for the predicate to rapidly oscillate between true and false. The predicates used in our approach depend on features of the environment – such as the time left in game, the number of goals scored by each team, and the number of robots available to each team – that by their nature cannot rapidly oscillate. This ensures that the play choice also cannot rapidly oscillate.

Figure 1 shows an example of a defensive play. Its applicability conditions specify that this play is applicable 1) when our team is winning and has fewer active players than the opponents or 2) when the game is in the second half and our team is winning by at least two points. If we have only one active robot on our team, we will assign it the Goalkeeper role; if we have two robots, one is assigned the Goalkeeper role and the other is assigned the Defender role; and so on. We have developed a total of sixteen plays, but not all were used in the RoboCup 2005 competition. Figure 2 shows a summary of the seven plays that were used in the competition. (Only the roles used for a 4-robot team are shown.)

```
PLAY Guard
APPLICABLE winning fewerPlayers
APPLICABLE secondHalf winningBy2OrMoreGoals
ROLES 1 Goalkeeper
ROLES 2 Goalkeeper Defender
ROLES 3 Goalkeeper Defender Independent
ROLES 4 Goalkeeper Defender Midfielder Independent
WEIGHT 3
```

Fig. 1. An example play with multiple applicability conditions.

Default: Goalkeeper Defender Striker Independent
Defensive: Goalkeeper Defender Midfielder Independent
Guard: Goalkeeper Defender MidfieldDefender Independent
Flankers: Goalkeeper Defender LeftFlanker RightFlanker
Aggressive: Goalkeeper LeftFlanker RightFlanker Independent
PullGoalie: Midfielder LeftFlanker RightFlanker Independent
Kickoff: Goalkeeper Defender Charger KickoffDodger

Fig. 2. Summary of the seven plays used by our team in RoboCup 2005.

3.2 Play Selector

The *play selector* runs on one robot that is arbitrarily chosen to be the leader. The play selector chooses which play the team should be running. The leader periodically broadcasts the current play (and role assignments) to its teammates. Distributed play-based coordination is achieved through a predefined agreement among the team members to resort to a *default play* if a robot doesn't hear a play broadcast within a *communication time limit*. A failure of the leader or a network problem may trigger this default coordination plan. A more sophisticated approach could incorporate an algorithm for *leader selection* in the event of failure. However, we did not pursue such an approach for the work presented in this paper. The algorithm used by the play selector is presented in Figure 3.

```
SELECT_PLAY(S: world state, P: playbook, D: default play):
  BEST_PLAY <- D
  BEST_WEIGHT <- WEIGHT(D)
  for each PLAY in P:
    if WEIGHT(PLAY) > BEST_WEIGHT:
      for each CONDITIONS in APPLICABLE(PLAY):
        if all CONDITIONS are satisfied in STATE:
          BEST_PLAY <- PLAY
          BEST_WEIGHT <- WEIGHT(PLAY)
  return BEST_PLAY
```

Fig. 3. Algorithm used by the play selector.

3.3 Role Allocator

The selection of a play determines which roles need to be allocated to the robots. However, it does not specify which robots should be assigned to each role. Therefore, a role allocation algorithm is still needed to assign the roles. This algorithm also runs on the leader robot, which broadcasts the assignment along with the selected play. Our role allocator has two features that differentiate it from those used by many other RoboCup teams [5]. First, it only runs when a play is

initially selected, as opposed to continuously. Second, it allocates roles in a *role-preserving* manner – minimizing role switching. Formally, if a new play P_t is selected at time t, and P_t specifies n roles $\{R_{1..n}\}$ for the n robots $r_{1..n}$, and r_i was already assigned to R_j in P_{t-1}, r_i is guaranteed to still be assigned to R_j in P_t. (Any remaining roles can be allocated in a greedy fashion.) If two plays share some roles, this strategy guarantees that some of the robots can assume their new roles without any transitional cost. These features provide additional resistance to oscillation in cases in which two plays share common roles.

3.4 Roles

The *role* assigned to each robot determines what behaviors the robot actually runs. Our approach, used in RoboCup 2005, is unique in that it is *region-based*: each robot is assigned to a region of the field. A robot is primarily responsible for going after the ball whenever the ball is in that robot's region. Roles are designed simply by configuring a generic "Player" behavior with appropriate settings for that role. The configurable items include:

- Region: an area of the field that the robot is responsible for covering.
- Ball in Region Policy: the behavior the robot should adopt when it knows that the ball is in its region. Typically, this will involve approaching the ball and trying to clear it downfield or take a shot on goal.
- Ball out of Region Policy: the behavior the robot should adopt when it knows that the ball is not in its region. Some roles specify that a robot is simply to return to a home position, while other roles may have the robot move to block the path between the ball and the goal, or to position for a pass.
- Ball Lost Policy: the behavior the robot should adopt when it believes the ball is lost. This is typically some sort of searching behavior.

Unlike our previous approaches, robots no longer need to negotiate with one another in order to gain the *attacker* role that allows them to approach the ball. In this way, the performance of the team does not degrade significantly under high network latency. We have developed algorithms that prevent the robots from interfering with one another even when they are playing in overlapping regions. To provide robustness against communication failure, these algorithms are designed to operate without the need for communication, using local information such as a robot's vision of its own teammates. If communication is available, our robots use additional features (such as reported teammate positions) that provide added confidence that our robots will not interfere with one another.

3.5 Experimental Results / Discussion

Due to lack of space, we are unable to present detailed experimental results here. Instead, we refer the reader to previous work in which we have presented related results. In [6], we show that using high-level features, such as the presence of opponents, to select a team strategy can improve the goal-scoring performance

of a team of two robots. In [7], we explore the problem of *ball advancement* in a team of three robots. These results show that the role-preserving behavior of our role-assignment algorithm allows our team to maintain a consistent level of performance even when the active play is switched at a rapid rate.

The presented role-assignment algorithm and plays have been tested in the RoboCup 2005 competition. Our team came in fourth place in a challenging competition of twenty-four teams. Our team typically rotated through three well-balanced plays in the first minutes of each game, which allowed us to see the performance of each play against the specific opponent. We could manually change the team's strategy at halftime or during a timeout.

Our role assignment system is unique in that it allows role assignments to happen to all robots, including the goalkeeper. If there is not much time left in the game and our team is losing, we have plays that will "pull" the goalkeeper out of the goal box, which provides us with another field player that could score a goal. In fact, in the 3rd-4th place game of the RoboCup 2005 competition, our goalkeeper robot nearly scored a goal in the final seconds of the game.

4 Conclusion

In this paper, we have presented improvements to our team's communication and world modeling strategies. These improvements place high priority on the communication of task-relevant data and ensure that robots communicate some useful information even when lost. We have also presented a distributed, play-based role-assignment algorithm, which aims to solve several important challenges, including the presence of adversaries, task-based temporal constraints, and robustness to network failure. Our future work includes automatic play adaptation within the underlying challenges of a distributed team, and principled reasoning about adversarial temporal constraints, such as changing strategies based on the time left in the game and the current score.

References

1. Committee, R.T.: Sony four legged robot football league rule book (2006)
2. Roth, M., Vail, D., Veloso, M.: A real-time world model for multi-robot teams with high-latency communication. In: Proc. IROS. Volume 3. (2003) 2494–2499
3. Pynadath, D., Tambe, M.: The communicative Multiagent Team Decision Problem. Journal of Artificial Intelligence Research **16** (2002) 389–423
4. Bowling, M., Browning, B., Veloso, M.: Plays as team plans for coordination and adaptation. In: Proceedings of ICAPS. (2004)
5. Gerkey, B.P., Mataric, M.J.: On role allocation in RoboCup. In: RoboCup 2003: Robot Soccer World Cup VII. (2004) 43–53
6. McMillen, C., Rybski, P., Veloso, M.: Levels of multi-robot coordination for dynamic environments. In: Multi-Robot Systems: From Swarms to Intelligent Automata, Volume III. Kluwer Academic Publishers (2005) 53–64
7. McMillen, C., Veloso, M.: Distributed, play-based role assignment for robot teams in dynamic environments. In: Proc. Distributed Autonomous Robotic Systems. (2006)