Resolving Conflicts in Highly Reactive Teams

Hendrik Skubch, Daniel Saur, and Kurt Geihs

Distributed Systems
Universität Kassel
Wilhelmshöher Allee 73, Kassel, Germany
{skubch, saur, geihs}@vs.uni-kassel.de

Abstract

In distributed cooperation frameworks for completely autonomous agents, conflicts between the involved agents can occur due to inconsistent data available to the agents. In highly dynamic domains, such as RoboCup, it is often neccessary to accept a certain level of conflicts in order to decrease reaction time of single agents, instead of relying on error-free, but extensive communication. However, this may lead to situations in which unresolved conflicts linger and cause cooperation to break down completely. In our cooperation framework, ALICA, we designed and implemented a simple but effective approach to detect these cases and resolve them quickly through a bully algorithm.

Keywords and phrases Agents, Multi-agent, cooperation, time critical, conflict resolution

Digital Object Identifier 10.4230/OASIcs.KiVS.2011.170

1 Introduction

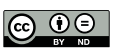
Cooperation between autonomous agents acting in highly dynamic domains under time constraints is an important and difficult challenge. As it becomes feasible to tackle increasingly challenging scenarios using autonomous robots, such as search and rescue or exploration, completely distributed approaches to teamwork become more important. In this light, central points of failure need to be eliminated, and adaptability and robustness increased.

One prominent scenario in which such conditions are examined is the Middle Size League of RoboCup[1], where teams of five autonomous robots compete against each other. The RoboCup domain has several key features common to many real world problems: *Noisy data:* Data obtained from the sensors is very noisy due to the high speeds at which objects move and unforseen confusing objects in the background, e.g., in the audience. *Incomplete information:* The sensor range of a single robot covers only a fraction of the field's size, hence important objects such as the ball are sometimes not observed by any robot. *Wireless Communication:* 802.11 wireless is used for communication, which is an unreliable, potentially crowded medium, so packet loss and latency are relevant. *Dynamic Environment:* The game is very quick, as robots move with up to $6m/s$, i.e., $20cm$ per decision cycle (30 Hz). *Full Autonomy:* During a game, there is absolutely no human interaction with the playing robots, with the exception of the referee starting and stopping the game.

Hence, both highly reactive acting and coherent teamwork are needed. However, these two goals interfere with each other, as reactivity forbids communication before acting, and coherent teamwork requires communication, given noisy and incomplete sensor information.

Using RoboCup as a test scenario, we developed a modelling framework for cooperative agents, called ALICA [8], which uses a completely distributed and autonomous way of decision making. Although each robot informs its team about its actions and integrates received

[1] www.robocup.org



17th GI/ITG Conference on Communication in Distributed Systems (KiVS'11).
Editors: Norbert Luttenberger, Hagen Peters; pp. 170–175

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

information into its decision making process, due to timing constraints a robot cannot afford to wait for messages from its team members before it decides and acts. While this approach guarantees swift reactions, it can cause inconsistent decisions. This is most pronounced if the sensory data are very noisy, or even diverge completely. In most cases, these inconsistencies are momentarily and are solved within a few decision cycles [8], however in some cases, the problem can persist and cause the teamwork to break down for a significant period of time. Requiring sensory data to be of sufficient quality to avoid this problem is unrealistic. Instead, we deem a more explicit handling of the conflicts in question necessary. In this paper, we present a simple method extending the teamwork approach of ALICA, which solves the problem of persisting conflicts.

In the next Section, we briefly discuss related work, focussing on approaches in similar domains. In Section 3, we explain the fundamental notions of teamwork within ALICA. Section 4 explains how persisting conflicts can be detected and handled in detail. Finally, Section 5 shows evaluation results on real robots and concludes our discussion.

2 Related Work

Plenty of teamwork approaches use a central processing unit to either guarantee consistent data, or to make the team-relevant decisions. Among them, several are employed in RoboCup [6, 1, 2]. In these approaches, perceived data is sent from the robots to a central processing unit which either returns commands or fused sensory data, thus guaranteeing consistency. These approaches are robust and easy to implement, however they lack the flexibility to scale up to larger teams or to teams operating in unreliable communication environments with real-time constraints. Furthermore, a central processing unit always presents a single point of failure.

Other teamwork approaches, such as STEAM [9], rely on explicit communication before acting. In particular an extension to STEAM, CONSA (COllaborative Negotiation System based on Argumentation) [5, 4], allows the involved agents to present arguments to each other when confronted with a conflict.

Although the requirement of explicit communication is partially alleviated by a decision theoretic estimation of the risk of not communicating, we deem this approach not suitable for very dynamic domains. In particular, in domains such as robotic soccer, the situation changes too fast for argumentation to take place. Moreover, it is unclear how local data can be used in an argument, when sensor fusion has already failed to avoid the conflict.

Some approaches, such as [3], follow a related line of thought, by basing argumentation on belief revision operators, focussing on how arguments are understood and what kind of agreements can be achieved. These approaches are not suitable for fast paced domains, where it is more important to act immediately than it is to make the ideal choice.

3 Teamwork in ALICA

The central notion within ALICA are *plans*. Intuitively, a plan is a recipe, describing how a certain goal can be achieved by a set of robots. More specifically, a plan contains a non-empty set of finite automata, each labelled with a *task*. If a team of robots is to execute a plan, the robots have to allocate themselves to these tasks and hence each executes one finite automaton. A task allocation is defined by a set of believes of the form $\mathrm{In}(a, p, \tau, z)$, denoting that agent a takes on task τ within plan p, and currently inhabits state z.

Figure 1 shows a simple plan, with the two tasks *Attack* and $Defend$. *Attack* contains

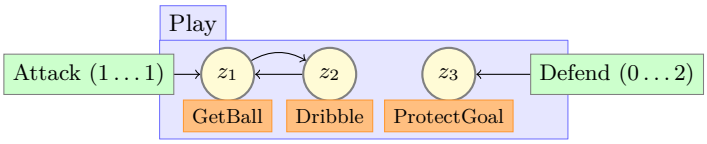


Figure 1 A simple ALICA plan

two states, one for obtaining the ball, one for dribbling towards the opponent's goal. $Defend$ contains only a single state, in which agents should protect the goal. Each task can be executed by multiple agents, restricted by a cardinality, denoted $\xi(p, \tau)$. Here, exactly one robot must execute the task *Attack*, while up to two can execute the task $Defend$.

The task allocation problem is subject to several restrictions, namely said cardinalities, potential preconditions the plan might have, formulated in first order logic, and role restrictions, which can forbid robots with certain roles to take on specific tasks. For instance, a robot designated as goalie should not take on the task *Attack*.

Although these restrictions potentially rule out plenty of allocations, there are usually a vast number of valid allocations for a given situation and plan. For each plan, a utility function maps the currently believed situation, including a potential allocation, onto the real numbers. Each robot chooses the task allocation which maximises this function. The utility of the plan can depend on arbitrary conditions reflected in a robot's belief base, ranging from the roles a robot is assigned to highly dynamic properties such as distances to other robots, the ball, or the goal. Hence, each participating robot calculates the allocation individually.

In ALICA, plans occur grouped in *plan types*, such that each plan within such a plan type represents an alternative solution to the given problem. In order to execute a plan type, a single plan together with a task allocation needs to be picked by the executing robots. Thus, task allocations for different plans are evaluated and compared, adding a further dimension to the task allocation problem.

Accommodating the dynamic environment, each robot constantly recalculates the best achievable allocation and adopts it if a small hysteresis criteria is met. This is called *reallocation*. In the example above, a suitable utility function could maximise if the robot closest to ball takes on the task *Attack*. If an opponent kicks the ball, the robots can switch tasks immediately. Calculating an allocation and acting upon it is done completely autonomously by each robot. Since the involved beliefs are noisy, conflicts arise. ALICA assumes that the robots communicate periodically their chosen tasks, thus most conflicts are dealt with automatically by reallocation. If however, their beliefs regarding the decision-relevant data do not tend to converge, these conflicts can persist. For example, our current ball detection algorithms tend to underestimate small distances and overestimate large distances. Thus, if two robots are very close to the ball, each will perceive it as being closer to itself. In this situation, a persistent conflict can occur.

4 Dealing with Conflicts

Our approach to deal with persisting conflicts is divided in two parts: Detection and reaction. In the following, we will present an indicator of such conflicts and afterwards discuss a simple solution based on bullying to resolve these conflicts. Firstly, we discuss how conflicts can be detected. The decisions which are subject to potential conflicts are called *task allocations*.

▶ **Definition 1** (Task Allocation). (Slightly simplified from [7]) A task allocation for a plan type pt is a conjunction of the form $\mathrm{In}(a_1, p, \tau_1, z_1) \wedge \mathrm{In}(a_2, p, \tau_2, z_2) \wedge \ldots \wedge \mathrm{In}(a_n, p, \tau_n, z_n)$,

such that plan p belongs to pt. A task allocation C for a plan p is valid wrt. belief base $\mathcal{F}$ iff

$$\mathcal{F} \cup C \models (\forall \tau \in \text{Tasks}(p))(\exists A)(\forall a)\,(a \in A \leftrightarrow (\exists z) In(a, p, \tau, z)) \wedge \tag{1}$$

$$(\exists n_1, n_2)\,\xi(p, \tau) = (n_1, n_2) \wedge n_1 \leq |A| \leq n_2 \tag{2}$$

$$(\forall \tau \in \text{Tasks}(p))(\exists n_1, n_2)\,\xi(p, \tau) = (n_1, n_2) \wedge n_1 \leq |A| \leq n_2 \tag{3}$$

$$\mathcal{F} \cup C \models \text{In}(a, p, \tau_1, z_1) \wedge \text{In}(a, p, \tau_2, z_2) \rightarrow \tau_1 = \tau_2 \wedge z_1 = z_2 \tag{4}$$

$$\mathcal{F} \cup C \models \text{Pre}(p) \tag{5}$$

$$\mathcal{U}_p(\mathcal{F} \cup C) \geq 0 \tag{6}$$

Where $\text{Tasks}(p)$ denotes the set of tasks of plan p, $\xi(p, \tau)$ the cardinality of a plan-task pair and $\text{Pre}(p)$ indicates the precondition of p.

Since each agent maintains its own task allocation, it is possible for the team to disagree on the allocation. In this case, the team is in *conflict*. Each agent periodically broadcasts all plan-task-state triples it executes, enabling the team to detect inconsistent allocations. We call these messages *plan messages*. When an agent receives a plan message, it updates its allocation accordingly, trusting the sender completely.

While this treatment of conflicts enables the team to react swiftly [8], it can cause reoccurring conflicts. If the underlying data is inconsistent within the team, it is possible for two or more agents to consider conflicting allocations as optimal. Reallocation can then cause them to continuously inform each other about the outcome of their decisions, while reallocating the respective other agents. This can go on until an external event stops the loop or the data becomes consistent. In the remainder, the specific state an agent inhabits given a plan and task is irrelevant, we will thus write $\text{In}(a, p, \tau)$ instead of $\text{In}(a, p, \tau, z)$.

▶ **Definition 2** (Conflict). Two allocations C_1 and C_2 are in conflict, if and only if they are for the same plan type pt, and, for some agent a, $\text{In}(a, p, \tau) \in C_1$, but $\text{In}(a, p, \tau) \notin C_2$. Two agents a_1 and a_2 are in conflict wrt. plan type pt iff they believe in conflicting allocations. We say $\text{In}(a, p, \tau)$ is a cause of the conflict. Note, there can be multiple causes for a conflict.

▶ **Proposition 1.** *If two agents a_1 and a_2 are in conflict wrt. plan type pt, caused by $\text{In}(b, p, \tau)$, then there is a conflict between one of them and agent b.*

This property guarantees that the robot, which needs to change its actions, is in conflict with at least one robot and thus has a chance to detect it. In order to devise a detection mechanism, we examine how allocations are modified over time. An agent's allocation for a plan type pt can change due to: *Reallocation*: the agent adopts a new allocation to improve the utility. *Messages*: the agent receives a message informing it about the state of another wrt. pt. *Leaving*: the agent leaves the plan and thus no longer tracks its allocation. *Deletion*: the agent has not received a message from another agent for a predetermined time. In this case the agent is removed from the team and thus from all task allocations.

Deletion is done to account for agents breaking down. We will not consider the deletion event further, as it can be seen as an empty plan message. We will also not consider agents leaving a plan, this event terminates the local agent's tracking of the plan and is visible to other agents by message events. These are able to detect a conflict arising by an agent entering and leaving a plan repeatedly. Conflicts which endure for a longer period of time cause a temporal pattern to appear in allocations believed by the involved agents. We therefore define formally how allocations change over time.

▶ **Definition 3** (Allocation Event). An allocation event e is a tuple $(\vartheta^+, \vartheta^-)$, consisting of allocation additions ϑ^+ and allocation subtractions ϑ^-, both sets of atoms of the form $\text{In}(a, p, \tau)$, such that $\vartheta^+ \cap \vartheta^- = \emptyset$.

▶ **Definition 4** (Allocation Event Composition). Let $e_1 = (\vartheta_1^+, \vartheta_1^-)$ and $e_2 = (\vartheta_2^+, \vartheta_2^-)$ be two task allocation events. Then their composition is defined as:

$$e_1 \circ e_2 \stackrel{def}{=} \left((\vartheta_1^+ \setminus \vartheta_2^-) \cup (\vartheta_2^+ \setminus \vartheta_1^-), (\vartheta_1^- \setminus \vartheta_2^+) \cup (\vartheta_2^- \setminus \vartheta_1^+)\right)$$

This composition allows to reason about results of multiple events independently of the allocations they apply to. Note, allocation events form an abelian group under event composition. If an agent a receives a plan message from another agent b, this causes an allocation event for every plan type agent a is executing. There are four cases to distinguish:

- a agrees with b on its allocation, hence the allocation event is empty and can be ignored.
- a does not believe that b participates in the plan type, but b does, this yields the allocation event $(\{\mathrm{In}(b, p, \tau)\}, \emptyset)$ for some p and τ.
- a believes that b executes a different plan or task than b actually does, yielding the event $(\{\mathrm{In}(b, p, \tau)\}, \{\mathrm{In}(b, p'\tau')\})$ such that $p \neq p' \vee \tau \neq \tau'$.
- a wrongly believes that b participates in the plan type, causing $(\emptyset, \{\mathrm{In}(b, p', \tau')\})$.

By Proposition 1, these are the only cases we need to consider. If a now reallocates and the composition of both events contains no statement about b, a cycle occurred.

▶ **Definition 5** (Allocation Cycle). Let e_m be a non-empty allocation event due to a plan message sent by agent b to agent a, and let e_r be the next allocation event after a receives the message. If $e_m \circ e_r = (A, B)$ such that $(\forall p, \tau)\, \mathrm{In}(b, p, \tau) \notin A \cup B)$, a cycle occurred.

Hence, a cycle occurs, if an agent reverts its allocation with respect to a message by reallocation. An agent can easily monitor its allocation events and thus detect such cycles. Of course, a cycle does not necessarily entail a conflict, it might well be that the cycle just reflects a very quickly changing situation. However, if multiple cycles occur subsequently, this becomes more and more unlikely to be a proper reaction to the changing conditions.

Given this detection scheme, we can devise a reaction to arising conflicts. The cycle length depends on the frequency with which the robots communicate and deliberate. In our case, messages are sent every $100ms$ and the deliberation loop takes $30ms$. Thus, the duration of two subsequent cycles is in average $115ms$. It is improbable that during this time frame the situation requires the team to change its allocation back and forth twice. Thus, we set a limit $n \geq 2$ on the number of subsequent cycles which may occur before an agent acts upon them.

In order to resolve the detected conflicts, we use a bully algorithm, where one agent forces the others into the allocation it deems best wrt. to the relevant utilities. The bullying agent assumes authority on the corresponding task allocation for a fixed time period. This time interval should be large enough to overcome the cause of the conflict, yet as small as possible so the team can return to its more dynamic and adaptive state as quickly as possible. Once the interval passed, all agents resume normal operation. An agent detecting n subsequent cycles will trigger the bully algorithm by broadcasting its allocation, triggering all involved agents to either respond or assume the proposed allocation. The leader is determined using unique ids for each robot. Only those agents which execute the corresponding plan type participate, not the whole team. Thus, the conflict is dealt with locally.

5 Evaluation & Conclusion

We evaluated our solution on a team of real robots, with both lab experiments and observations of the behaviour during real tournament situations. In all our experiments, we set the number of subsequent cycles to four and the bullying duration to two seconds, as mentioned above.

Fighting for the ball: In cases, where two robots moved very close to the ball, persistent conflicts become most apparent. Previously we observed cases, in which a conflict persisted for several minutes. The presented approach reliably detected and solved this conflict.

Phantom ball: Another typical scenario where conflicts can occur, is if at least one robot of the team detects a phantom ball far from the actual ball position. This can be due to an object looking similar to the ball appearing in the audience (e.g., an orange t-shirt), or an actual second ball on the field. In this case, depending on where the phantom ball is seen, the conflict will cause either two robots to pursue the ball, or none at all. Again, the conflict could be reliably detected and solved, however the solution to the conflict is arbitrary, as the team will react to the hypothesis of the robot with the highest id.

Tournament situations: We employed our approach during real tournament situations at RoboCup German Open 2010. Although we could not gather enough data for a rigorous statistical analysis, it can be said, that during one hour of playing, bullying occurred on average twelve times. In all cases, there was an actual conflict, which needed to be solved. In one third of the cases, the solution to the conflict was incorrect, due to the bullying robot not being correctly localised. In the remaining cases, the solution was acceptable and lead to teamwork being reinstated quickly.

In conclusion, our simple, yet efficient conflict resolution method deals well within teams of agents, extending the ALICA framework for cooperation. During real matches, the robotic team was able to detect and repair conflicts quite reliably.

The drawback of our approach is that the solution to a conflict is only as good as the data, the bullying robot bases its decision on. Hence, the robot with the best data should take precedence, which could be indicated by confidence values of the sensory information. However, since these confidence values vary over time and are possibly similar among the participating agents, this bears the danger of being another source for disagreement. Argumentation based methods, as in [5], might help to identify which team member should take lead, using confidences, or geometric arguments such as line-of-sight.

References

1 J. J. T. H. de Best, D. J. H. Bruijnen, R. Hoogendijk, R. J. M. Janssen, K. J. Meessen, R. J. E. Merry, M. J. G. van de Molengraft, G. J. L. Naus, and M. J. C. Ronde. Tech United Eindhoven Team Description 2010. Technical report, University of Eindhoven, 2010.

2 Nau Lau, Luís Seabra Lopes, Gustavo A. Corrente, and Nelson Filipe. Multi-robot team coordination through roles, positionings and coordinated procedures. In *IROS*, 2009.

3 Benedita Malheiro and Eugenio Oliveira. Argumentation as distributed belief revision: Conflict resolution in decentralised co-operative multi-agent systems. In *EPIA '01*, 2001.

4 Zhun Qiu and Milind Tambe. Flexible Negotiation in Teamwork (Extended Abstract). In *AAAI FALL Symposium on Distributed Continual Planning*, 1998.

5 Zhun Qiu and Milind Tambe. Towards Argumentation-based Collaborative Negotiation: A Preliminary Report. In *Int. Workshop on MAS*. Mass. Institute of Technology, 1998.

6 H. Rajaie, U.-P. Käppeler, O. Zweigle, K. Häussermann, A. Tamke, A. Koch, B. Eckstein, F. Aichele, D. DiMarco, A. Berthelot, T. Walter, and P. Levi. 1. rfc stuttgart,overview of hardware and software. Technical report, University of Stuttgart, 2010.

7 Hendrik Skubch, Michael Wagner, Roland Reichle, and Kurt Geihs. A modelling language for cooperative plans in highly dynamic domains. *Mechatronics*, 2010.

8 Hendrik Skubch, Michael Wagner, Roland Reichle, Stefan Triller, and Kurt Geihs. Towards a comprehensive teamwork model for highly dynamic domains. In *ICAART*, 2010.

9 M. Tambe. Towards flexible teamwork. *Journal of AI Research*, 7:83–124, 1997.