Multi-Agent versus Multi-Robot and Other Byzantine Discussions

Vicente Matellán
Editor of the special issue on multi-robot systems
E-mail: vicente.matellan@unileon.es

***Abstract*—This special issue of the Journal of Physical Agents is devoted to multi-robot systems. Some might perhaps argue that the term should have been "multi-agent systems" or "physical multi-agent systems", in accordance with the title of the journal. In any language, there are clusters of several words with closely similar meanings. Expressions in science and technology are no exceptions, although they are supposed to be more specific. In particular, in the world of computer science, the term "agent" has become a buzzword and has to be combined with numerous adjectives to be correctly understood. This short introductory article will try to justify why the name was chosen for the journal, but not for this special issue, the intended scope of the journal and the contents of this issue.**

***Index Terms*—JoPhA, agent, robot, distributed.**

JOPHA SCOPE

WHEN a new journal is born, it has to define its own editorial line. When their promoters decide to establish it, they have a clear idea in their minds about its scope, and it is usual for all of them to think that this is well defined and even that their views coincide perfectly. However, it is impossible for all of them to share exactly the same idea about what it will look like. There will be shades of opinion due to their personal interests, education, environment, and so forth that will prevent any group of humans from achieving this perfect coincidence.

The evolution of a publication over time will create its own personality. In this process the founders are not the dominant factor. The authors sending in papers and, above all, the editors and reviewers will determine this by giving greater prominence to some papers than to others, in such a way defining the real scope of the journal.

The Journal of Physical Agents is no exception. In the scope description of this new journal the editors stated that it was seeking contributions in all areas that use agent-based technologies. In particular, mobile robotics, autonomous systems, or domotic systems were cited as among potential areas of interest for the journal.

This was the seminal idea of the founders of the journal, the members of the Spanish National Network on Physical Agents. In this grouping, we had been organizing national meetings for almost 10 years and realized we had not found a regular publication reflecting our common interests, at either a local or an international level.

Vicente Matellán is with the department of Mechanical and Computer Engineering, University de León. Edif. Tecnológico II (D-333). Campus de Vegazana. 24071 León (Spain).

Most of the members of this community work with robots. However, typically we are not robot builders, which makes us appear to fall in the category "soft-robotics" from the point of view of the traditional robotics community. In this sense, existing robotics journals do not fit our work well, although they are our most usual route for dissemination of research. In other words, we did not want to create yet another robotics journal.

Much of the work that we undertake does not use robots alone, although it can be considered as work in robotics. We are interested in any type of device that can be controlled by a computer, that is connected to the real world (the physical world, we mean), and that is able to establish communication with other devices. This might be our definition of a physical agent. Figure 1 outlines it as being at the crossroads of the areas of several different research communities, focusing primarily on the intersection between robots and agents.

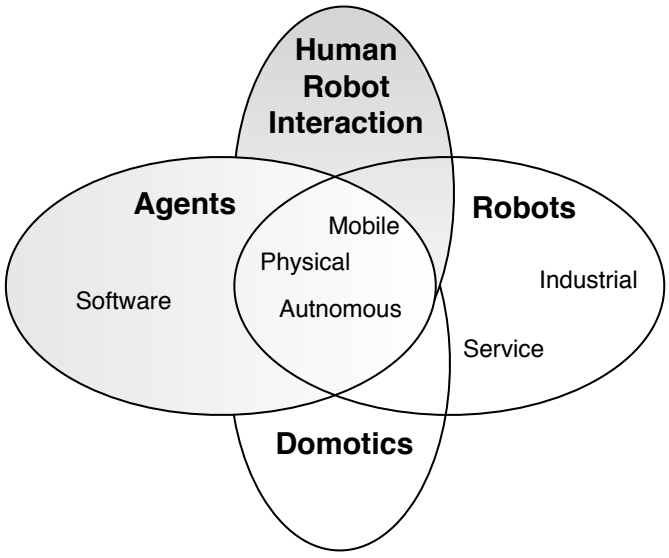


Fig. 1. JoPhA scope as seen by journal founders

As an example, in this issue, devoted to multi-robot systems, there are papers covering traditional autonomous legged robots, but also others on computerized cars. Similarly, the next issue will be given over to "Human interaction with domestic agents"[1], in other words, to an exploration of aspects of interaction with embodied devices such as Aibo, Pleo, Paro

[1]http://www.jopha.ua.es/JoPhA_CFP-.pdf

and Nabaztag, or software such as Catz & Dogz or Nintendogs, as well as the numerous non-commercial devices and systems that have been developed in many research labs. As was explained above, these numbers of the journal are defining its real scope for potential readers.

On the other hand, most people in this community are also involved in matters more closely related to "software agents", probably owing to their computer science background. They are also very active in that field, in both research and development. Such agents are not physical, but there have been many developments in this area that have influenced physical work. Moreover, physical agents can interact with software agents.

In addition it was realized that this profile is not merely a Spanish one, but can also be found in other countries. This encouraged us to establish the journal in English and give it an international scope. For instance, Dr. P. Stone, one of the authors of the articles that make up this issue, perfectly matches such a profile, having made significant contributions to the area of software agents, such as [12], but also being very active in traditional robotics, for instance [11].

Another question when a new journal is proposed is the desired level of applicability of the papers expected, in other words, whether it will be a theoretical or a practical journal. We think that in the current state of this field within computer science these two aspects can still be kept together, that there is not yet any formal corpus, so that many contributions are both theoretical and applied.

In short, JoPhA wishes to become a medium for the dissemination of theoretical and practical achievements in the field of physical agents, that is, in computer-controlled devices, embedded in the real world, capable of establishing communication with other similar devices and humans.

A. Robot versus Agent

The previous section attempted to focus on the scope of the journal. However, it should be stressed that from our point of view the expressions "robot" and "physical agent" can be used indistinguishably in some cases, but not all. In order to delimit these cases, an attempt will now be made to define the terms.

In the light of the above, "agent" and "robot" may have different meanings, depending on the author and the context. For instance, a reader coming from an industrial background is likely to see a robot as "A re-programmable multi-functional manipulator designed to move materials, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks" (Robotics Institute of America).

However, a reader from a computer science department would be more likely to define a robot as an autonomous, computer-controlled device made up of sensors and actuators. Something similar happens with the definition of an agent. The classic definition [14]describes an agent as a hardware or software-based computer system with the characteristics of autonomy, reactivity, pro-activeness and social ability.

Taking the first definition of robot, there would be few points of contact between the two areas. However, in robotics,

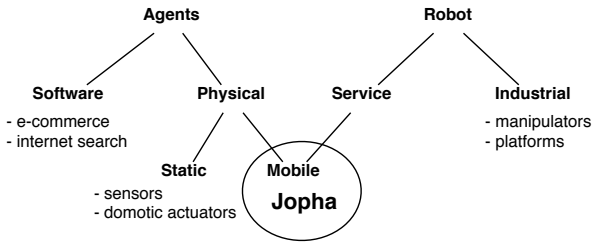


Fig. 2. Light taxonomy of robots and agents

as stated in [13], a mobile robot is often called a mobile agent, either virtual or physical. Nevertheless, sometimes a robot may be controlled by a single-agent architecture, while in other situations there may be several agents controlling the same robot.

Figure 2 tries to show the overlap between these two concepts. Robots can be divided into "industrial" robots, typically manipulators or automated platforms for in-factory delivery, and service robots, typically mobile robots. On the other hand, agents can be classified into software agents, usually Internet agents that can travel from one server to other, and physical agents, those that "live" in the real world. Hence, mobile robots and physical agents overlap.

This overlap has also appeared clearly in the literature. For instance, Dudek et al. [5] presents a detailed taxonomy of "multi-agent robotics", although this is not the only way of referring to multiple robot systems. For example, the expression "Cooperative robots" is used in [4], while in [2] "robot teams" is used. In brief, groups of mobile robots working together, or at least interacting, are among the clearest examples of physical agents that can be found.

B. This issue content

The contents of this issue are intended to show what we want the journal to be, in this instance focusing on the area of multi-robot systems. Hence, the issue includes various theoretical papers, such as the contribution by Bruno Lacerda et al. [9], alongside others that are much more applied, for example the piece by Patrick Beeson et al. [3]. An overview of the field is also included, in the shape of the invited paper by Lynne Parker [10], and a more applied contribution from Y. Elmaliach et al. [6] , emphasizing the problem of human-robot interaction in this domain.

JOPHA FUTURE

Our goals as a community at this point are simply to consolidate this publication as a quarterly journal[2]. We do not think that issues will be very large. We assume that an average issue will comprise 5 papers. On the other hand, we do not want to set maximum limits on the number of pages in a paper. If an author does need more space we are willing to offer it, obviously on condition that the relevant editor is in agreement.

[2]Note that 2008 will be the first complete year, and may be we just reach three issues

We are making a bid to distribute the contents of the journal as *libre* (free) material, as well as to avoid any cost for authors. All of the contents are intended to become available on the Internet under a Creative Commons licence. However, any publishing effort of this kind needs some financial support. We would encourage all potential readers to subscribe themselves, or to ask their departments, universities, schools, or libraries to subscribe.

Finally, this is not an Internet or virtual journal. We prefer it to be a paper-based journal and wish to keep it in print form for many reasons. This is the traditional method for the dissemination of research, which we favour. Paper journals are very convenient, as they can be read anywhere (even in the absence of a net connection, or power), and libraries are a well-established and reliable method for storing and indexing knowledge.

ACKNOWLEDGMENT

I should like to recognize explicitly the great efforts made by all of the organizers of the annual sessions of the Spanish WAF (Workshop on Physical Agents), which have helped to create a local community around the "Physical Agent" concept in Spain and have given birth to this journal. In particular, I should like to express my extreme gratitude to Miguel A. Cazorla for his support and tireless work.

REFERENCES

[1] T. Arai, E. Pagello, and L.E. Parker, *Advances in multi-robot systems*. IEEE Transactions on Robotics and Automation, Vol. 18(5), pp. 655-661, 2002.

[2] Tucker Balch, and Lynne Parker, *Robot Teams: From Diversity to Polymorphism*. A.K. Peters, 2002.

[3] Patrick Beeson, Jack O'Quin, Bartley Gillan, Tarun Nimmagadda, Mickey Ristroph, David Li, and Peter Stone. *Multiagent Interactions in Urban Driving*. JoPhA Vol 2(1), 2008.

[4] Y. Uny Cao and Alex S. Fukunaga and Andrew B. Kahng. *Cooperative Mobile Robotics: Antecedents and Directions*, Autonomous Robots, Vol 4 (1), pp. 7-23, 1997.

[5] G. Dudek and M. Jenkin and E. Milios and D. Wilkes. *A Taxonomy for Multi-Agent Robotics*, Autonomous Robots, Vol. 3(4), pp. 375-397, 1996.

[6] Yehuda Elmaliach and Gal A. Kaminka. *Robust Multi-Robot Formations under Human Supervision and Control*. JoPhA Vol 2(1), 2008.

[7] L. Iocchi, D. Nardi, and M. Salerno, *Reactivity and Deliberation: a Survey on Multi-Robot Systems*, Lecture Notes in ComputerScience, Vol. 2103, pp. 9-34, 2001

[8] Editors: Amir Shapiro and Gal A. Kaminka. Special Issue on Multi-Robot Coverage, Search, and Exploration. Annals of Math and Artificial Intelligence (AMAI) 2008 (to appear).

[9] Bruno Lacerda, Pedro Lima. *Time Temporal Logic Control of Discrete Event Models of Cooperative Robots*. JoPhA Vol 2(1), 2008.

[10] Lynne E. Parker. *Distributed Intelligence: Overview of the Field and its Application in Multi-Robot Systems*. JoPhA Vol 2(1), 2008.

[11] M. Sridharan and Peter Stone. *Structure Based Color Learning on a Mobile Robot under Changing Illumination*. Autonomous Robots, Vol. 23(3). pp. 161-182, 2007.

[12] Michael P. Wellman, Amy Greenwald, and Peter Stone. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*, MIT Press, 2007.

[13] Grzegorz Wieczerzak, and Krzysztof Kozkowski. *Agents that Live in Robots: How are successful applications built?*. Fourth International Workshop on Robot Motion and Control, pp. 97-102, Puszczykowo (Poland) 2004.

[14] M. Wooldridge, and N.R. Jennings, *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review, Vol. 10(2), pp. 115-152, 1995.

Distributed Intelligence: Overview of the Field and its Application in Multi-Robot Systems

Lynne E. Parker

***Abstract*—This article overviews the concepts of distributed intelligence, outlining the motivations for studying this field of research. First, common systems of distributed intelligence are classified based upon the types of interactions exhibited, since the type of interaction has relevance to the solution paradigm to be used. We outline three common paradigms for distributed intelligence — the bioinspired paradigm, the organizational and social paradigm, and the knowledge-based, ontological paradigm — and give examples of how these paradigms can be used in multi-robot systems. We then look at a common problem in multi-robot systems — that of *task allocation* — and show how the solution approach to this problem is very different depending upon the paradigm chosen for abstracting the problem. Our conclusion is that the paradigms are not interchangeable, but rather the selection of the appropriate paradigm is dependent upon the specific constraints and requirements of the application of interest. Further work is needed to provide guidance to the system designer on selecting the proper abstraction, or paradigm, for a given problem.**

***Index Terms*—Distributed intelligence, multi-robot systems, multi-agent systems, task allocation.**

I. Introduction to Distributed Intelligence

DISTRIBUTED Intelligence refers to systems of *entities* working together to reason, plan, solve problems, think abstractly, comprehend ideas and language, and learn. Here, we define an *entity* as any type of intelligent process or system, including agents, humans, robots, smart sensors, and so forth. In these systems, different entities commonly specialize in certain aspects of the task at hand. As humans, we are all familiar with distributed intelligence in teams of human entities. For example, corporate management teams consist of leaders with particular specialties such as Chief Executive Officer (CEO), Chief Operating Officer (COO), Chief Financial Officer (CFO), Chief Information Officer (CIO), and so forth. Oncology patient care teams consist of doctors that specialize in various areas, such as surgical oncology, medical oncology, plastic and reconstructive surgery, pathology, etc. Distributed intelligence is also exhibited in military applications, such as special forces A-Teams, where team members specialize in weapons, engineering, medicine, communications, and so forth. Another military example includes personnel on an aircraft carrier flight deck, who are segmented into the catapult crew, the landing signal officers, ordnancemen, plane handlers, etc. As humans have clearly learned, these teams can very efficiently solve complex tasks by making use of specialists who work together productively.

The objective of distributed intelligence in computer science (and related fields) is to generate systems of software agents, robots, sensors, computer systems, and even people and animals (such as search and rescue dogs) that can work together with the same level of efficiency and expertise as human teams. Clearly, such systems could address many important challenges, including not only urban search and rescue, but also military network-centric operations, gaming technologies and simulation, computer security, transportation and logistics, and many others.

As a research topic, the study of distributed intelligence has gained much popularity in recent years. Figure 1 shows data from the Web of Science resulting from a keyword search on the terms “distributed intelligence”, “distributed AI”, “distributed artificial intelligence”, “multiagent”, “multi-agent”, “distributed robot”, “multirobot”, and “multi-robot”. Each year’s results show the number of publications that appeared containing these keywords in that year. The search begins in year 1980 — the most recent year with no publications containing any of these keywords — up through 2006. While this is an admittedly incomplete survey of this area of research, the data clearly shows the significantly increasing interest in this research area, as investigators and application developers are recognizing the potential power of distributed intelligence.

What is the potential promise of distributed intelligence? Certainly, some applications can be better solved using a distributed solution approach — especially those tasks that are inherently distributed in space, time, or functionality. Further, if a system is solving various subproblems in parallel, then it offers the potential of reducing the overall task completion time. Any system consisting of multiple, sometimes redundant entities, offers possibilities of increasing the robustness and reliability of the solution, due to the ability for one entity to take over from another failing entity. Finally, for many applications, creating a monolithic entity that can address all aspects of a problem can be very expensive and complex; instead, creating multiple, more specialized entities that can share the workload offers the possibility of reducing the complexity of the individual entities.

Of course, these advantages of distributed intelligence are, to some extent, offset by some disadvantages. For example,

Lynne Parker is with the Distributed Intelligence Laboratory in the Department of Electrical Engineering and Computer Science at University of Tennessee, Knoxville, Tennessee, USA.

E-mail: parker@cs.utk.edu

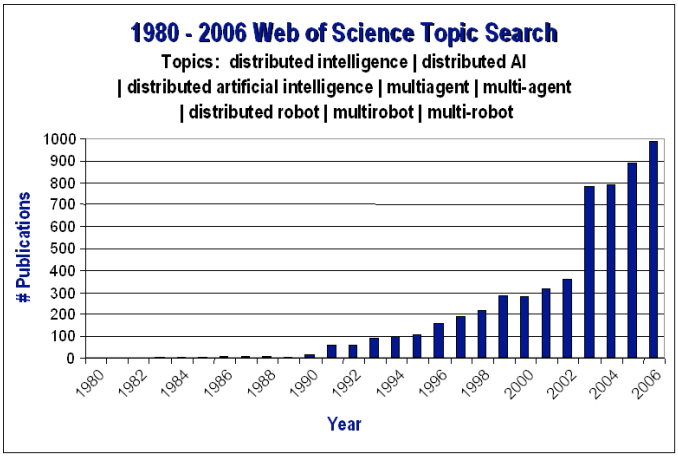


Fig. 1. Web of Science data showing the number of publications appearing per year on topics related to distributed intelligence.

even though the individual entity cost and complexity may be less, determining how to manage the complete system may be more difficult and complex, because of the lack of centralized control or of a centralized repository of global information. Further, distributed intelligent systems may require more communication to coordinate all the entities in the system. Increasing the number of entities can lead to increased interference between entities, as they must act without complete knowledge of the other entities' intents. Finally, systems of multiple entities will typically experience increased uncertainty about the state of the system as a whole.

Overall, however, as new solution approaches are being developed and validated, the research and user community is finding that it is often possible to realize the advantages of distributed intelligence systems while countering many of the possible disadvantages. The challenge is determining how best to properly design the system so as to achieve global coherence through the local interactions of individual entities.

II. The Domain Space of Distributed Intelligence

As researchers are discovering, there are many possible solution strategies, or paradigms, for achieving distributed intelligence. Not all of these paradigms are appropriate for all types of distributed intelligence. Thus, it is important to understand the various types of distributed intelligence that can occur in different application settings.

There are many ways to classify distributed intelligent systems and multi-robot systems, such as those proposed in [28], [17], [24], [102]. While these existing classifications provide important insight to various characteristics of multi-robot systems, they do not focus specifically on the types of of *interactions* exhibited by these systems. For our purposes, we believe it is helpful to view the domain space in distributed intelligence in terms of the types of interactions that can take place between the entities in the system. To help make the distinctions between the various types of interactions clear, we find it useful to define a minimal set of variables (or characteristics) whose values, together, can help us easily categorize the different types of interactions exhibited in multi-robot systems.

Toward this end, as illustrated in Figure 2, we view the types of interactions along three different axes — the *types of goals*, whether entities have *awareness of others* on the team, and whether an entity's actions *advance the goals of others* on the team. In terms of types of goals, we classify systems into two types — those in which each entity has *individual* goals, and those in which the entities have *shared* goals. For the *awareness of others* axis, we divide the systems into those that are *aware* and those that are *not aware*. By *aware* in this context, we refer to whether entities reason about the actions and intentions of their teammates. Robots that are *not aware* may sense the presence of local entities and move so as to maintain a certain distance, for example, but otherwise perform no other reasoning to understand the intent or future plans of the teammates. Often, these "un-aware" systems operate based on the principle of *stigmergy*, in which communication between entities is not direct, but rather through changes made in the environment.

Finally, we segment systems into those in which an entity's actions do advance the goals of others on the team (*yes*) and those that do not (*no*). An example of an entity advancing the goals of others with its actions is a floor cleaning robot, as a member of a floor cleaning robot team. Each robot's actions of cleaning a bit of the floor are helpful to the other teammates, who do not have to repeat the floor cleaning in that particular spot.

Obviously, these segmentations of the domain space are approximate, yet we believe they are helpful in quickly understanding and categorizing the primary types of interactions that can occur in typical applications. Different areas of this subspace represent common types of interactions seen in systems of distributed intelligence. These common forms of interaction are:

- Collective
- Cooperative
- Collaborative
- Coordinative

In the following paragraphs we describe these types of interactions in more detail, discussing their relevance and application to multi-robot systems.

Perhaps the simplest type of interaction is the *collective* interaction, in which entities are not aware of other entities on the team, yet they do share goals, and their actions are beneficial to their teammates. An example of this type of interaction in multi-robot systems is the swarm robotics work of many researchers (e.g., [73], [71], [59]). This work focuses on creating systems of robots that can perform biologically-relevant tasks, such as foraging, swarming, flocking, herding, formation-keeping, and so forth. Robots in these systems typically perform relatively simple local control laws which, when combined with larger numbers of robots, result in the global goal being achieved, often as an emergent property of the local interactions.

The second type of interaction is the *cooperative* interaction, in which entities are aware of other entities, they share goals, and their actions are beneficial to their teammates. In multi-robot systems, an example of this type of interaction is multiple robots working together and reasoning about each

other's capabilities in order to accomplish a joint task, such as pushing a box (e.g., [39]), cleaning up a worksite (e.g., [82]), performing search and rescue (e.g., [77]), or extra-planetary exploration (e.g., [103]). In these systems, robots may at times be working on different parts of the higher level goal, and thus may at times have to ensure that they share the workspace without interfering with each other. However, the majority of the work of the robots is focused on working together to achieve a common goal.

A third type of interaction in systems of distributed intelligence occurs when robots have individual goals, they are aware of their teammates, and their actions do help advance the goals of others. This part of the domain space is typically called *collaborative*, and is characterized by entities helping each other to achieve their individual, yet compatible, goals. While closely associated to the cooperative domain space, we make a distinction here to focus on the ability of entities to work together to help others better achieve their individual goals. In human research teams, we are familiar with the concept of collaboration, in which each person brings unique expertise that helps the team as a whole achieve a broader objective. Each team member has his/her own goal of performing his/her own aspect of the research, but by working together with others with complementary expertise, each can help the other members better achieve their individual goals. Of course, most of these collaborations are also cooperative, and it is possible to turn a collaborative team into a cooperative team by simply viewing the team goals from a higher perspective. A multi-robot example of a collaborative team is a group of robots that each must reach specified goal positions that are unique to each member. If robots are unable to reach their goal positions independently, due to sensor limitations, they could work together with other robots by sharing sensory capabilities to help all team members reach their individual goal locations. This type of collaboration is sometimes called *coalition formation*, and has been illustrated in multi-robot systems in [85], [119].

Finally, the fourth type of interaction relevant to distributed intelligence is what we call *coordinative*. In these systems, entities are aware of each other, but they do not share a common goal, and their actions are not helpful to other team members. In multi-robot systems, these situations often occur when robots share a common workspace. The robots must work to coordinate their actions to minimize the amount of interference between themselves and other robots. Multi-robot path planning techniques (e.g., [56], [43], [1], [25], [86], [88], [19], [107], [53], [93], [63]) or traffic control techniques (e.g., [40], [54], [66], [3], [125], [121]) are commonly used in these domains.

As a side note, we could have extended the third axis of our domain space to categorize systems based on whether they (1) *positively* affect the goals of other entities, (2) have *no effect* on the goals of other entities, or (3) *negatively* effect the goals of other entities. Then, we could create a new type of interaction in which entities have individual goals, they are aware of each other, but their actions have a negative effect on others' goals. This defines the *adversarial* domain, in which entities actively work against each other. In multi-robot systems, this topic is studied extensively in the multi-robot soccer application domain (e.g., [55], [15], [118], [101]). This form of interaction also has clear relevance for many security and military applications.

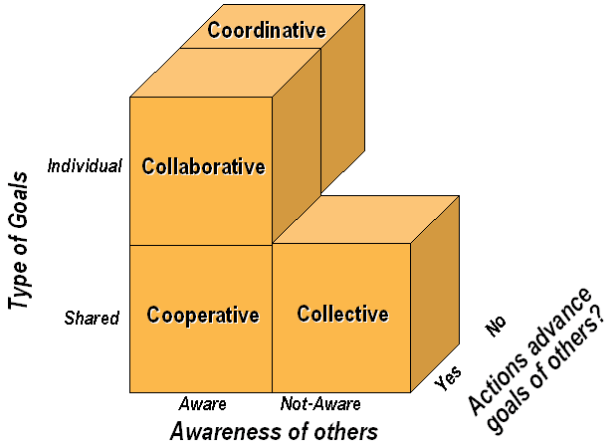


Fig. 2. Categorization of types of interactions in systems of distributed intelligence.

Understanding the types of interactions that we want to achieve in a distributed intelligence system can provide insights into the appropriate solution strategy. The following section outlines some common paradigms for distributed intelligence that can achieve these varying types of interactions in multi-robot systems.

III. Paradigms for Distributed Intelligence

Just as there are many types of interactions in systems of distributed intelligence, there are also many paradigms for achieving distributed intelligence. Each paradigm abstracts the problem space in a different way, enabling the system designer to view the system from a perspective that sheds light on proper solution strategies. Often, these paradigms take inspiration from societies of insects, or societies of humans. Not all paradigms are appropriate for all types of interaction dynamics. In this section, we outline some of the more common paradigms for distributed intelligence, especially focusing on their relevance to multi-robot systems. Note that a fundamental challenge in all of these paradigms is determining how best to achieve global coherence from the interaction of entities at the local level. By abstracting the problem in different ways, alternative solution strategies become apparent that can help address this challenge.

Three commonly used paradigms for building systems of distributed intelligence include:

- Bioinspired, emergent swarms paradigm,
- Organizational and social paradigms, and
- Knowledge-based, ontological, and semantic paradigms.

The following subsections outline these paradigms in more detail.

A. Bioinspired, emergent swarms paradigm

The behavioristic approach to autonomous robot control that gained popularity beginning in the 1980's [13] has its roots

in the observations of animal behavior. Animals, particularly the lower animals, are existence proofs that interesting results can be achieved without the need for a complex, human-level architecture. Many animals appear to be "hard-wired" for certain behaviors, producing very stereotypical reactions to particular stimuli. For instance, a robin begins defending its territory when it sees the red breast of another robin, or even a bunch of red feathers [26]. A pregnant Three-spined Stickleback fish approaches a male Stickleback with a red belly, or even a crude model of a Stickleback, as long as it is painted red underneath [114]. A male grayling butterfly flies up to mate rather large, dark, close, dancing objects, which could include not only female graylings, but also birds, falling leaves, and shadows [115].

Applying animal observations to the realm of autonomous robotics, interesting and seemingly intelligent activities can be obtained by layering behaviors that react to stimuli from the world according to the robot's current internal state [13]. Rather than decomposing the robot control system based on information processing functions, the behavioristic approach decomposes the control into task achieving behaviors, such as obstacle avoidance, exploration, and map building. The result was a series of autonomous robots that can survive in a dynamic world, avoiding obstacles, exploring the environment, following walls, building maps, climbing over uneven terrain, and so forth [14].

But this same approach — the observation of animal behavior — that has been used for inspiration in the development of individual robots is just as easily used to gain insight into the creation of groups of robots that cooperate toward attaining some goal [80]. By learning how various species of animals function as groups, ideas can be obtained for building a cooperating team of autonomous robots.

1) Two Types of Animal Societies: Since there are so many varieties of social behavior in the animal kingdom, a classification of animal societies is useful. One such classification, proposed by Tinbergen [114], is of particular interest for current robotics research in multi-robot systems, as it parallels two possible approaches to achieving multi-robot systems. According to Tinbergen, animal societies can be grouped into two broad categories: those that *differentiate*, and those that *integrate*.

Societies that *differentiate* are realized in a dramatic way in the social insect colonies [124]. These colonies arise due to an innate differentiation of blood relatives that creates a strict division of work and a system of social interactions among the members. Members are formed within the group according to the needs of the society. In this case, the individual exists for the good of the society, and is totally dependent upon the society for its existence. As a group, accomplishments are made that are impossible to achieve except as a whole.

On the other hand, societies that *integrate* depend upon the attraction of individual, independent animals to each other. Such groups do not consist of blood relatives that "stay together", but instead consist of individuals of the same species that "come together" by integrating ways of behavior [90]. These individuals are driven by a selfish motivation which leads them to seek group life because it is in their own best interests. Interesting examples of this type of society are wolves, or the breeding colonies of many species of birds, in which hundreds or even thousands of birds congregate to find nesting partners. Such birds do not come together due to any blood relationship; instead, the individuals forming this type of society thrive on the support provided by the group. Rather than the individual existing for the good of the society, we find that the society exists for the good of the individual.

2) Parallels in Multi-Robot Systems: In analyzing research in multi-robot systems, a parallel can be drawn with the classifications of animal societies discussed above. A large body of work in robotics, including much of the earliest work (e.g., [21], [112], [98], [23], [70], [10], [59], [99], [33], [122]), involves the study of emergent interactions in colonies of robots — an approach comparable to *differentiating* animal societies. This research emphasizes the use of large numbers of identical robots that individually have very little capability, but when combined with others can generate seemingly intelligent group behavior. This intelligent group behavior is achieved as a side-effect of the individual robot behaviors. This type of interaction is typically called *collective*.

In this paradigm, the need for communication between entities is greatly reduced by assuming the ability of the entities to sense relevant information in their local environments (i.e., *stigmergy*). The application requirements in these problems allow for simple action protocols, or control rules, that are identical on each entity, and that lead to the desired group behavior. An example local control rule under this paradigm that can cause all the agents/robots to aggregate (as in a swarm) is [71]:

```
Aggregate:
  If agent is outside aggregation
      distance
  then turn toward aggregation
      centroid and go.
  Else
     stop.
```

This is a powerful paradigm for those applications that require the same task to be performed across a distributed workspace, where the task does not require complex interactions of entities and all entities are interchangeable. Research challenges include developing tools that can predict the global behavior given a set of local control rules, as well as the inverse problem, in which we want to derive the local control rules, given a desired global behavior. This paradigm is relevant for many spatially distributed applications, including flocking, schooling, and formations (e.g., [92], [70], [5], [81], [105], [73], [75], [76], [8]); foraging, coverage, and search (e.g., [87], [31], [120], [34], [4], [104], [94], [106], [16], [69]); target tracking and observation (e.g., [7], [123], [65], [62], [57], [51], [111]); sorting and clumping [8], and so forth. While earlier approaches to these applications were based on human-generated local control rules that were demonstrated to work in practice, more recent work is based on control theoretic principles, with a focus on proving stability and convergence properties in multi-robot team behaviors. Examples

of this work include [47], [9], [116], [29], [68], [35], [36], [108], [2].

Other types of interactions, however, require more complex solution paradigms. Thus, a second approach parallels the *integrative* societies in the animal kingdom. This research aims to achieve higher-level, "intentional" cooperation or collaboration amongst robots. Rather than beginning with robots having very low-level behaviors, individual robots that have a higher degree of "intelligence" and capabilities are combined to achieve purposeful cooperation or collaboration. The goal is to use robots that can accomplish meaningful tasks individually, and yet can be combined with other robots with additional skills to complement one another in solving tasks that no single robot can perform alone. To be purely analogous to the integrative animal societies, robots in this type of interaction would have individual, selfish, motivations which lead them to seek cooperation or collaboration [72]. Such interactions would be sought because it is in the best interests of each robot to do so to achieve its mission. Of course, the possession of a selfish motivation to cooperate or collaborate does not necessarily imply consciousness on the part of the robot. It is doubtful that we would attribute consciousness to all the integrative societies in the animal kingdom; thus, some mechanism must exist for achieving this type of productive interaction without the need for higher-level cognition. A large body of multi-robot systems research falls into this category (e.g., [82], [123], [78], [12], [15], [67], [101], [103], [119], [1]). The next two paradigms discussed in this article — namely, the organizational/social paradigm and the knowledge-based/ontological/semantic paradigm — are additional techniques that have been used to create similar higher-level, intentional cooperation and/or collaboration in multi-robot teams.

The type of approach one should use for the multi-robot solution is dependent upon the applications envisioned for the robot team. The differentiating interaction approach is useful for collective tasks requiring numerous repetitions of the same activity over a relatively large area (relative to the robot size), such as waxing a floor, cleaning barnacles off of ships, collecting rock samples on a distant planet, and so forth. Such applications would require the availability of an appropriate number of robots to effectively cover the work area while continuing to maintain the critical distance separation.

On the other hand, the intentional interaction approach would be required for cooperative or collaborative interactions in applications requiring several distinct tasks to be performed, perhaps in synchrony with one another. Throwing more robots at such problems would be useless, since the individual tasks to be performed cannot be broken into smaller, independent subtasks. Examples of this type of interaction include container management in ports [1], some aspects of extra-planetary exploration [103], some types of search and rescue [49], some aspects of mineral mining [95], transportation [113], industrial and household maintenance [83], construction [96], hazardous waste cleanup [82], security [27], [44], agriculture [89], and warehouse management [45].

Of course, there is overlap in the relevance of these approaches to various applications, and in some instances the differences are a matter of degree. For instance, if large numbers of robots are too expensive or are not available to be applied to, say, planetary exploration, then more purposive interaction (i.e., cooperation or collaboration) is required to achieve the goal of the mission. Combinations of the approaches are also possible by using intentionally interacting robots to guide the activities of smaller groups of swarm robots in a coordinated way (e.g., [84]).

B. Organizational and social paradigms

Organizational and social paradigms are typically based on organizational theory derived from human systems. Knowledge from the fields of sociology, economics, and psychology, and related areas, have proven valuable for understanding how to create systems of intelligent artifacts that can work together to solve complex problems. In these approaches, agent/robot interactions are designed by modeling individual and group dynamics as part of an organization. These approaches reduce the communications requirements among entities by making use of models drawn from these fields. This type of approach is commonly used for *cooperative* and *collaborative* types of distributed intelligence. Three examples of organizational theory applied to multi-robot systems are the use of *roles*, the use of *market economies*, and the use of *teamwork models*.

Roles are often used to divide the application into manageable portions of the work that can each be assigned to a different robot in the team. An easy division of work is achieved by assigning roles according to the skills and capabilities of the individual team members. For instance, in multi-robot soccer [100], [67], [118], positions played by the different robots are often defined as roles, such as goal keeper, left defender, right defender, left forward, right forward, and so forth. The robot best suited, and perhaps in closest proximity, to the available roles/positions then selects to perform that role.

Market economies are used in multi-robot systems as a paradigm for *task allocation*, which we discuss further in the next section. In brief, task allocation is the problem of mapping tasks to robots, such that the most suitable robot is selected to perform the most appropriate task, leading to all tasks being satisfactorily completed. Market-based approaches to task allocation (e.g., [22], [126], [39], [12]) make use of the theory of market economies to determine how best to allow robots to negotiate on responsibilities in the mission. Market economy approaches define methods for how to manage bids, how to handle multiple bids in parallel, how to consider multiple tasks at once, and so forth. More discussion on this topic is given in the next section.

Teamwork models allow agents/robots to explicitly reason about coordination and communication. In dynamic environments, the ability to reason about the interactions of agents/robots can enable the team members to reorganize themselves as needed to address new situations that arise. Many teamwork models, such as Tambe's STEAM model [109], are based on the concept of *joint intentions* [20], which models the joint mental state of the team. The idea is that a team is jointly committed to a task if its individual team

members are committed to the task and believe that they are executing the task. Protocols for establishing team member commitments are defined as part of this general model. A related concept is that of *shared plans* [41], which represent agents' intentions to perform particular tasks.

C. Knowledge-based, ontological, and/or semantic paradigm

A third paradigm commonly used for developing systems of distributed intelligence is the *knowledge-based, ontological, and/or semantic* paradigm. The focus in these approaches is on knowledge sharing between heterogeneous robots/agents, with the objective of easily allowing these entities to share and understand knowledge from disparate sources. Often, knowledge is defined as an *ontology*, which specifies a common vocabulary and semantics for the knowledge in the system. Such approaches require a language for representing knowledge, such as the Knowledge Interchange Format (KIF) [37], as well as a language for communicating knowledge, such as the Knowledge Query and Manipulation Language (KQML) [30]. This paradigm achieves communication reduction by making use of shared assumptions of vocabulary and semantics.

This type of paradigm can be used for many types of interactions, including cooperative, collaborative, and coordinative. However, while this paradigm has become perhaps the dominant paradigm in multi-agent systems, it is not commonly used in multi-robot systems, at least in the form of full-fledged ontologies. More than likely, this is because physical robot systems are more challenged by noise and uncertainty in sensing and actuation, as well as low-bandwidth communications, limited power, and limited computation. As such, the limiting bottleneck in multi-robot systems is not typically the semantics of the shared knowledge, but rather dealing with these uncertainties. However, this does not mean that multi-robot systems do not use knowledge-based approaches. On the contrary, many approaches do model information about the system and about the teammates in order to more effectively cooperate, collaborate, and coordinate.

IV. Contrasting Paradigms for a Typical Multi-Robot Challenge: Task Allocation

Having explored three common paradigms in systems of distributed intelligence, we now compare and contrast these paradigms in their approach to a common challenge in multi-robot systems — that of *task allocation*. As previously introduced, task allocation arises in many multi-robot applications in which the mission of the team is defined as a set of tasks that must be completed. Each task can usually be addressed by a variety of different robots; conversely, each robot can usually work on a variety of different tasks. Independent tasks can be achieved concurrently, while dependent tasks must be achieved according to their interdependence constraints. Once the set of tasks has been defined, the challenge is to determine the preferred mapping of robots to tasks that optimizes some objective function. This is the task allocation problem. (See Gerkey [38] for a taxonomy and formal analysis of the computational complexity of variants of the Multi-Robot Task Allocation (MRTA) problem.) The general task allocation problem is known to be NP-hard [38], meaning that optimal solutions cannot be found quickly for large problems. Therefore, solutions to this problem are typically approximations that are acceptable in practice. The following subsections examine how each of the paradigms we have discussed would handle the multi-robot task allocation problem.

A. Bioinspired approach to task allocation

The bioinspired approach to task allocation typically assumes large numbers of homogeneous robots that are all interchangeable. In this situation, any robot that is available and senses the need for a task to be performed can select to perform that task (i.e., the task is *allocated* to that robot). Because of *stigmergy*, robots do not have to explicitly communicate to decide which task to undertake. Or, alternatively, robots may broadcast minimal information about their state or their environment without undertaking any conversations with other robots about which robot should perform which tasks. Robots that fail can be replaced by any other available robot. If all robots operate under this principle, then the entire mission is typically accomplished.

An example of this type of task allocation is the work of Balch and Arkin [6]. While this work focused on the effect of communication on a society of robots, their work also achieved implicit task allocation through stigmergy and minimal inter-robot communication of robot state and/or goal information. The tasks studied in this work were *Forage*, *Consume*, and *Graze*. The *Forage* task requires robots to wander in the environment, seeking out an *attractor* object. Once found, the robot attaches to the attractor and returns it to home base. The *Consume* task is similar to the *Forage* task, except that the robot performs work in place on the object, rather than return it home. Finally, the *Graze* task requires the robots to collectively visit all areas of the environment. To accomplish these tasks, the robots were programmed with a schema-based reactive control system that enabled robots to avoid obstacles, detect attractors, move to a desired goal destination (e.g., Home or an attractor), etc. Through a variety of simulations that varied the settings of parameters in the system, Balch and Arkin showed that the robots could successfully complete their set of tasks through stigmergy, when available, or through the use of minimal state communication when sensing through the environment (i.e., stigmergy) was not possible.

Most other bioinspired approaches to achieving robot collectives also implicitly achieve task allocation through stigmergy. Examples of these approaches include [59], [60], [70], [74], [87], [99], [105], [106].

B. Organizational approach to task allocation

An organizational approach to task allocation could make use of *roles*, as we described previously for multi-robot soccer. Each role encompasses several specific tasks, and robots select roles that are best suited for their capabilities. In this case, robots need not be homogeneous, but instead can have a variety of different sensing, computation, and effector capabilities.

An example of role-based task allocation is the work of Simmons, et al. [96], who have developed the Distributed Robot

Architecture (DIRA), which allows autonomy in individual robots and facilitates explicit coordination among robots. Their approach is based on a layered architecture, in which each robot's control architecture consists of a *planning* layer that decides how to achieve high-level goals; an *executive* layer that synchronizes agents, sequences tasks and monitors task execution; and a *behavioral* layer that interfaces to the robot's sensors and effectors. Each of these layers interacts with those above and below it. Additionally, robots can interact with each other via direct connections at each of the layers. Task allocation is achieved through a role-based approach, in which each robot takes on one of the roles necessary for the task. This architecture has been demonstrated in a team of three robots — a crane, a roving eye, and a mobile manipulator — performing a construction assembly task. This task requires the robots to work together to connect a beam at a given location. Each robot fulfills a critical role of the overall task. In these demonstrations, a software agent acting in the "foreman" role decides which robot should move the beam at which times. Initially, the crane moves the beam to the vicinity of the emplacement based on encoder feedback. The foreman then sets up a behavioral loop between the roving eye and the crane robot to servo the beam closer to the point of emplacement. Once the beam is close enough, the foreman tasks the roving eye and the mobile manipulator to servo the arm to grasp the beam. After contact is made, the foreman tasks the roving eye and the mobile manipulator to coordinate to servo the beam to the emplacement point, thus completing the task.

Teamwork models for cooperation and collaboration in multi-robot/multi-agent teams often also use roles to achieve task allocation. For example, in the work of [109], roles are used to simplify the assignment and execution monitoring of the activities of agents on the team during the mission. Other role-based approaches include [101], [117], [48], [18], [79].

An alternative organizational approach to task allocation is the market-based approach. In these approaches, robots explicitly communicate to bid for tasks according to their expected contribution to those tasks. Assignments are typically made by greedily assigning each task to the robot that can perform it with the highest utility. The fundamental paradigm for interaction in this case is based upon the Contract Net Protocol [97], which was the first to address the problem of how agents can negotiate to collectively solve a set of tasks. The use of a market-based approach specifically for multi-robot task allocation was first developed in the M+ architecture [12]. In the M+ approach, robots make incremental choices of tasks to perform from the set of *executable* tasks. Tasks are considered executable if all of their predecessor tasks have been accomplished or are underway. For these tasks, robots create their own individual plans, estimating their costs for executing these tasks. The robots then compare their costs to offers announced by other robots. The robot selects to perform the task of lowest cost that it can perform that is better than the cost announced by any other robot. Other aspects of the M+ architecture allow robots to negotiate with other teammates to incrementally adapt their actions to suit the team as a whole, through the use of social rules that facilitate the merging of plans.

A significant amount of recent work addresses more powerful market-based approaches for task allocation [39], [58], [96], [117], [61], [11], [91], [127], [42], [52], [64], [32], [50], [85]. These approaches can deal with more complex task representations, more coupling between tasks, dynamic events, combinatorial auctions, and so forth. See [22] for a complete survey of these approaches.

C. Knowledge-based approach to task allocation

The knowledge-based approach is also used for task allocation in multi-robot teams, through the modeling of teammate capabilities. Many variations are possible, such as in the ALLIANCE approach by Parker [82], in which robots model the ability of team members to perform the tasks of the system by observing team member performance and collecting relevant task quality statistics, such as time to task completion. Robots then use these models to select tasks to perform that benefit the group as a whole. In this approach, explicit communication is not required for the selection of task assignments.

Other techniques are also possible that make use of learned models of teammate capabilities. For example, the COBOS work of Fua and Ge [32] maintains a *task suitability matrix*, which is a team model representing the suitability of each robot on the team to perform each task. Tasks can be broken down into subtasks, and the suitability of a robot to perform a macro task can be calculated from its suitability for performing lower-level *basis* tasks. In this work, suitability is calculated based on the compatibility of a robot's *intrinsic* abilities fundamental to the task; *extrinsic* factors, such as time and distance, can also be incorporated into the task description.

Work by Parker and Tang [85], [110] in the development of ASyMTRe is an example of a *semantic* approach to task allocation. In ASyMTRe, each robot team member possesses a set of building-block capabilities, called *schemas*. Examples of schemas include *perceptual schemas*, which extract information from sensors, and *motor schemas*, which convert information from perceptual schemas to motor command outputs. In this approach, each schema's inputs and outputs represent semantic *information types* that must be obtained (in the case of inputs), or are generated (in the case of outputs). This semantic information can be obtained from any source, such as from a schema on the same robot, or a schema on a different robot. By determining a valid information flow through a combination of schemas within, and across, robot team members, the team as a whole is able to compose *coalitions* for solving given tasks. The use of semantic information in this approach enables sensor sharing across heterogeneous team members in a manner that is flexible and extendible to a variety of team compositions and team tasks.

D. Summary of contrasting paradigms for task allocation

As we see through these task allocation examples, a specific problem in multi-robot systems can be addressed in many different ways, based upon the paradigm selected for abstracting the problem at hand. Each paradigm has its own advantages and disadvantages, which may be specific to the application. The paradigms are therefore not interchangeable for many

applications, with the most suitable approach depending upon the relevant constraints and requirements of the application.

V. Conclusions

In this paper, we have outlined aspects of the field of distributed intelligence, focusing on the types of interactions that can occur in such systems, and some common paradigms used to achieve distributed intelligence. To explore the challenges, we have used examples from the field of multi-robot systems to illustrate, compare, and contrast the alternative interactions and paradigms. The main message of these discussions is that the choice of paradigm is not always obvious, and is dependent upon the requirements of the application to be addressed. We also note that complex systems of multiple robots can make use of several different paradigms simultaneously. For example, a large-scale exploration, mapping, deployment, and detection problem, such as that described in [46], can make use of an organizational paradigm to define roles for the high-level abstraction, an application-specific knowledge-based approach for multi-robot mapping, a knowledge-based modeling approach for mobile network deployment, and a bioinspired approach for creating a mobile sensor network. The challenge as system designers is to create and make use of the appropriate paradigms that best address the specific constraints and challenges of the application at hand.

References

[1] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the MARTHA project. *Robotics and Automation Magazine*, 5(1):36–47, 1998.

[2] G. Antonelli and S. Chiaverini. Kinematic control of platoons of autonomous vehicles. *IEEE Transactions on Robotics*, 22(6):1285–1292, 2006.

[3] H. Asama, K. Ozaki, H. Itakura, A. Matsumoto, Y. Ishida, and I. Endo. Collision avoidance among multiple mobile robots based on rules and communication. In *Proceedings of IEEE/RJS International Conference on Intelligent Robots and Systems*. IEEE, 1991.

[4] T. Balch. The impact of diversity on performance in robot foraging. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 92–99. ACM Press, 1999.

[5] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998.

[6] T. Balch and R. C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1995.

[7] R. W. Beard, T. W. McLain, and M. Goodrich. Coordinated target assignment and intercept for unmanned air vehicles. In *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 2002.

[8] R. Beckers, O. Holland, and J. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In R. Brooks and P. Maes, editors, *Proceedings of the 14th International Workshop on Synthesis and Simulation of Living Systems*, pages 181–189. MIT Press, 1994.

[9] C. Belta and V. Kumar. Abstraction and control for groups of robots. *IEEE Transactions on Robotics*, 20(5):865–875, 2004.

[10] G. Beni and J. Wang. Swarm intelligence in cellular robotics systems. In *Proceedings of NATO Advanced Workshop on Robots and Biological System*, 1989.

[11] M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt. Robot exploration with combinatorial auctions. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1957–1962. IEEE, 2003.

[12] S. Botelho and R. Alami. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1234–1239. IEEE, 1999.

[13] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[14] Rodney A. Brooks. New approaches to robotics. *Science*, 253:1227–1232, September 1991.

[15] B. Browning, J. Bruce, M. Bowling, and M. Veloso. STP: Skills, tactics and plays for multi-robot control in adversarial environments. *IEEE Journal of Control and Systems Engineering*, 219:33–52, 2005.

[16] Z. J. Butler, A. A. Rizzi, and R. L. Hollis. Cooperative coverage of rectilinear environments. In *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 2000.

[17] Y. Cao, A. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:1–23, 1997.

[18] L. Chaimowicz, B. Grocholsky, J. F. Keller, V. Kumar, and C. J. Taylor. Experiments in multirobot air-ground coordination. In *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 2004.

[19] C. M. Clark, S. M. Rock, and J.-C. Latombe. Motion planning for multiple mobile robot systems using dynamic networks. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4222–4227, 2003.

[20] P. R. Cohen and H. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

[21] J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario. Self-organizing collection and transport of objects in unpredictable environments. In *Japan-U.S.A. Symposium on Flexible Automation*, pages 1093–1098, Kyoto, Japan, 1990.

[22] B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, July 2006.

[23] A. Drogoul and J. Ferber. From Tom Thumb to the Dockers: Some experiments with foraging robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 451–459, 1992.

[24] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3:375–397, 1996.

[25] M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[26] W. Etkin. *Social Behavior from Fish to Man*. The University of Chicago Press, Chicago, 1964.

[27] H. R. Everett, R. T. Laird, D. M. Carroll, G. A. Gilbreath, T. A. Heath-Pastore, R. S. Inderieden, T. Tran, K. J. Grant, and D. M. Jaffee. Multiple Resource Host Architecture (MRHA) for the Mobile Detection Assessment Response System (MDARS). In *SPAWAR Systems Technical Documen 3026, Revision A*, 2000.

[28] A. Farinelli, L. Iocchi, and D. Nardi. Multi-robot systems: A classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics B*, 34(5):2015–2028, 2004.

[29] J. A. Fax and R. M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, 49(9), 2004.

[30] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *in Software Agents*. MIT Press, 1995.

[31] M. Fontan and M. Matarić. Territorial multi-robot task division. *IEEE Transactions on Robotics and Automation*, 15(5):815–822, 1998.

[32] C.-H. Fua and S. S. Ge. COBOS: Cooperative Backoff Adaptive Scheme for Multirobot Task Allocation. *IEEE Transactions on Robotics*, 21(6):1168–1178, 2005.

[33] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organizing robots based on cell structures — CEBOT. In *Proceedings of IEEE International Workshop on Intelligent Robots and Systems*, pages 145–150. IEEE, 1988.

[34] Douglas Gage. Randomized search strategies with imperfect sensors. In *Proceedings of SPIE Mobile Robots VIII*, pages 270–279. SPIE, September 1993.

[35] V. Gazi. Swarm aggregations using artificial potentials and sliding-mode control. *IEEE Transactions on Robotics*, 21(6):1208–1214, 2005.

[36] S. S. Ge and C.-H. Fua. Queues and artificial potential trenches for multirobot formations. *IEEE Transactions on Robotics*, 21(4):646–656, 2005.

[37] M. R. Genesereth and R. E. Fikes. *Knowledge Interchange Format, Reference Manual*. Computer Science Department, Univ. of Stanford, 1992.

[38] B. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

[39] B. P. Gerkey and M. J. Matarić. Sold! auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.

[40] D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, 4:491–497, 1988.

[41] B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.

[42] J. Guerrero and G. Oliver. Multi-robot task allocation strategies using auction-like mechanisms. In *Proc. of Sixth Congr. Catalan Association for Artificial Intelligence*, pages 111–122, 2003.

[43] Y. Guo and L. E. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2002.

[44] Y. Guo, L. E. Parker, and R. Madhavan. Towards collaborative robots for infrastructure security applications. In *Proceedings of International Symposium on Collaborative Technologies and Systems*, pages 235–240, 2004.

[45] C. Hazard, P. R. Wurman, and R. D'Andrea. Alphabet soup: A testbed for studying resource allocation in multi-vehicle systems. In *Proceedings of AAAI Workshop on Auction Mechanisms for Robot Coordination*, pages 23–30. AAAI, 2006.

[46] A. Howard, L. E. Parker, and G. S. Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment, and detection. *International Journal of Robotics Research*, 25:431–447, 2006.

[47] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6), 2003.

[48] J. Jennings and C. Kirkwood-Watts. Distributed mobile robotics by the method of dynamic teams. In T. Leuth, R. Dillman, P. Dario, and H. Worn, editors, *Proc. of Fourth International Symposium on Distributed Autonomous Robotic Systems*. Springer, 1998.

[49] J. S. Jennings, G. Whelan, and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *Proceedings of the 8th International Conference on Advanced Robotics*, pages 193–200, 1997.

[50] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coupled tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 570–575. IEEE, 2006.

[51] B. Jung and G. Sukhatme. Tracking targets using multiple mobile robots: The effect of environment occlusion. *Autonomous Robots*, 13(3):191–205, 2002.

[52] N. Kalra, D. Ferguson, and A. Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2005.

[53] K. Kant and S. W. Zucker. Toward efficient trajectory planning: the path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

[54] S. Kato, S. Nishiyama, and J. Takeno. Coordinating mobile robots by applying traffic rules. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1535–1541, July 1992.

[55] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matasubara. Robocup: A challenge problem of ai. *AI Magazine*, 18(1):73–86, 1997.

[56] S. Kloder and S. Hutchinson. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics*, 22(4):650–665, 2006.

[57] A. Kolling and S. Carpin. Multirobot cooperation for surveillance of multiple moving targets – a new behavioral approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1311–1316. IEEE, 2006.

[58] H. Kose, U. Tatlidede, C. Mericli, K. Kaplan, and H. L.Akin. Q-learning based market-driven multi-agent collaboration in robot soccer. In *Proceedings of the Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 219–228, 2004.

[59] C. R. Kube and H. Zhang. Collective robotics: From social insects to robots. *Adaptive Behavior*, 2(2):189–219, 1993.

[60] Masao Kubo and Yukinori Kakazu. Learning coordinated motions in a competition for food between ant colonies. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 487–492. MIT Press, 1994.

[61] M. Lagoudakis, E. Markakis, D. Kempe, P. Keshinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems I*. MIT Press, 2005.

[62] S. M. LaValle, H. H. Gonzalez-Banos, C. Becker, and J-C. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 731–736. IEEE, 1997.

[63] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Transactions on Robotics and Automation*, 14:912–925, 1998.

[64] L. Lin and Z. Zheng. Combinatorial bids based multi-robot task allocation method. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1145–1150. IEEE, 2005.

[65] S. Luke, K. Sullivan, L. panait, and G. Balan. Tunably decentralized algorithms for cooperative target observation. In *Proceedings of the fourth international joint conference on Autonomous Agents and Multiagent Systems*, pages 911–917. ACM Press, 2005.

[66] V. J. Lumelsky and K. R. Harinarayan. Decentralized motion planning for multiple mobile robots: The cocktail party model. *Autonomous Robots*, 4(1):121–135, 1997.

[67] S. Marsella, J. Adibi, Y. Al-Onaizan, G. Kaminka, I. Muslea, and M. Tambe. On being a teammate: Experiences acquired in the design of RoboCup teams. In O. Etzioni, J. Muller, and J. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 221–227, 1999.

[68] J. A. Marshall, M. E. Broucke, and B. R. Francis. Formations of vehicles in cyclic pursuit. *IEEE Transactions on Automatic Control*, 49(11), 2004.

[69] M. Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 19(2-3):323–336, 1997.

[70] M. J. Matarić. Designing emergent behaviors: From local interactions to collective intelligence. In J. Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 432–441. MIT Press, 1992.

[71] M. J. Matarić. Issues and Approaches in the Design of Collective Autonomous Agents. *Robotics and Autonomous Systems*, 16:321–331, 1995.

[72] D. McFarland. Towards robot cooperation. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 440–444. MIT Press, 1994.

[73] J. McLurkin. Stupid robot tricks: Behavior-based distributed algorithm library for programming swarms of robots. In *M.S. Thesis, Massachusetts Institute of Technology*, 2004.

[74] J. McLurkin and J. Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In *Symposium on Distributed Autonomous Robotic Systems*. Springer, 2004.

[75] A. I. Mourikis and S. I. Roumeliotis. Optimal sensor scheduling for resource-constrained localization of mobile robot formations. *IEEE Transactions on Robotics*, 22(5):917–931, 2006.

[76] A. I. Mourikis and S. I. Roumeliotis. Performance analysis of multirobot cooperative localization. *IEEE Transactions on Robotics*, 22(4):666–681, 2006.

[77] R. R. Murphy. Marsupial robots for urban search and rescue. *IEEE Intelligent Systems*, 15(2):14–19, 2000.

[78] R. R. Murphy, C. Lisetti, R. Tardif, L. Irish, and A. Gage. Emotion-based control of cooperating heterogeneous mobile robots. *IEEE Transactions on Robotics and Automation*, 18(5):744–757, 2002.

[79] E. Pagello, A. D'Angelo, and E. Menegatti. Cooperation issues and distributed sensing for multirobot systems. *Proceedings of the IEEE*, 94:1370–1383, 2006.

[80] L. E. Parker. Adaptive action selection for cooperative agent teams. In Jean-Arcady Meyer, Herbert Roitblat, and Stewart Wilson, editors, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 442–450. MIT Press, 1992.

[81] L. E. Parker. Designing control laws for cooperative agent teams. In *Proceedings of the IEEE Robotics and Automation Conference*, pages 582–587. IEEE, 1993.

[82] L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[83] L. E. Parker and J. Draper. Robotics applications in maintenance and repair. In S. Nof, editor, *Handbook of Industrial Robotics*, pages 1023–1036. Wiley Publishers, 2nd edition, 1999.

[84] L. E. Parker, B. Kannan, F. Tang, and M. Bailey. Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2004.

[85] L. E. Parker and F. Tang. Building multi-robot coalitions through automated task solution synthesis. *Proceedings of the IEEE, special issue on Multi-Robot Systems*, 94(7):1289–1305, 2006.

[86] D. Parsons and J. Canny. A motion planner for multiple mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 8–13, 1990.

[87] K. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine*, 22(3):52–67, 2002.

[88] M. Peasgood, C. Clark, and J. McPhee. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, to appear, 2008.

[89] T. Pilarski, M. Happold, H. Pangels, M. Ollis, K. Fitzpatrick, and A. Stentz. The demeter system for automated harvesting. In *Proceedings of the 8th International Topical Meeting on Robotics and Remote Systems*, 1999.

[90] A. Portmann. *Animals as Social Beings*. The Viking Press, New York, 1961.

[91] G. Rabideau, T. Estlin, S. Schien, and A. Barrett. A comparison of coordinated planning methods for cooperating rovers. In *Proceedings of AIAA Space Technology Conference*, 1999.

[92] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21:25–34, 1987.

[93] M. Rude. Collision avoidance by using space-time representations of motion processes. *Autonomous Robots*, 4:101–119, 1997.

[94] P. Rybski, S. Stoeter, C. Wyman, and M. Gini. A cooperative multi-robot approach to the mapping and exploration of mars. In *Proceedings of AAAI/IAAI-97*. AAAI, 1997.

[95] G. Shaffer and A. Stentz. A robotic system for underground coal mining. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 633–638, 1992.

[96] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In *Proc. of the ISER Seventh International Symposium on Experimental Robotics*. Springer-Verlag, 2000.

[97] R. G. Smith. The Contract Net Protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12), December 1980.

[98] L. Steels. Cooperation between distributed agents through self-organization. In Yves Demazeau and Jean-Pierre Muller, editors, *Decentralized A.I.* Elsevier Science, 1990.

[99] D. Stilwell and J. Bay. Toward the development of a material transport system using swarms of ant-like robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 766–771, 1993.

[100] P. Stone and M. Veloso. A layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.

[101] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[102] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[103] A. Stroupe, A. Okon, M. Robinson, T. Huntsberger, H. Aghazarian, and E. Baumgartner. Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance. *Autonomous Robots*, 20(2):113–123, 2006.

[104] K. Sugawara and M. Sano. Cooperative behavior of interacting simple robots in a clockface arranged foraging field. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Distributed Autonomous Robotic Systems*, pages 331–339. Springer, 2002.

[105] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotic Systems*, 13(3):127–139, 1996.

[106] S. Sun, D. Lee, and K. Sim. Artificial immune-based swarm behaviors of distributed autonomous robotic systems. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3993–3998. IEEE, 2001.

[107] P. Svestka and M. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23:125–152, 1998.

[108] P. Tabuada, G. Pappas, and P. Lima. Motion feasibility of multi-agent formations. *IEEE Transactions on Robotics*, 21(3):387–392, 2005.

[109] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[110] F. Tang and L. E. Parker. A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[111] Z. Tang and U. Ozguner. Motion planning for multitarget surveillance with mobile sensor agents. *IEEE Transactions on Robotics*, 21(5):898–908, 2005.

[112] Guy Theraulaz, Simon Goss, Jacques Gervet, and Jean-Louis Deneubourg. Task differentiation in Polistes wasp colonies: a model for self-organizing groups of robots. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 346–355, 1990.

[113] C. Thorpe, T. Jochem, and D. Pomerleau. The 1997 automated highway free agent demonstration. In *Proceedings of IEEE Conference on Intelligent Transportation System*, pages 496–501, 1997.

[114] N. Tinbergen. *Social Behavior in Animals*. Chapman and Hall LTD, Great Britain, 1953.

[115] N. Tinbergen. *Animal Behavior*. Time-Life Books, New York, 1965.

[116] C. M. Topaz and A. L. Bertozzi. Swarming patterns in two-dimensional kinematic model for biological groups. *SIAM Journal of Applied Math*, 65(1):152–174, 2004.

[117] D. Vail and M. Veloso. Multi-robot dynamic role assignment and coordination through shared potential fields. In A. Schultz, L. E. Parker, and F. Schneider, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings of 2003 International Workshop on Multi-Robot Systems*, pages 87–98. Springer, 2003.

[118] M. Veloso, P. Stone, and K. Han. The CMUnited-97 robotic soccer team: Perception and multiagent control. *Robotics and Autonomous Systems*, 29(2-3):133–143, 1999.

[119] L. Vig and J. A. Adams. Multi-robot coalition formation. *IEEE Transactions on Robotics*, 22(4):637–649, 2006.

[120] I. Wagner, M. Lindenbaum, and A. M. Bruckstein. Mac vs. PC – determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *International Journal of Robotics Research*, 19(1):12–31, 2000.

[121] Jing Wang. Fully distributed traffic control strategies for many-AGV systems. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, pages 1199–1204. IEEE, 1991.

[122] Jing Wang and Gerardo Beni. Distributed computing problems in cellular robotic systems. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, pages 819–826, Tsuchiura, Japan, 1990.

[123] B. B. Werger and M. J. Matarić. Broadcast of local eligibility for multi-target observation. In L. E. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotic Systems 4*, pages 347–356. Springer, 2000.

[124] E. Wilson. *The Insect Societies*. The Belknap Press, Cambridge, 1971.

[125] S. Yuta and S. Premvuti. Coordinating autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1566–1574. IEEE, 1992.

[126] R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research*, 25(1):73–101, January 2006.

[127] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3016–3023. IEEE, 2002.

Multiagent Interactions in Urban Driving

Patrick Beeson, Jack O'Quin, Bartley Gillan, Tarun Nimmagadda, Mickey Ristroph, David Li, Peter Stone

***Abstract*—In Fall 2007, the US Defense Advanced Research Projects Agency (DARPA) held the Urban Challenge, a street race between fully autonomous vehicles. Unlike previous challenges, the Urban Challenge vehicles had to follow the California laws for driving, including properly handling traffic. This article presents the modular algorithms developed largely by undergraduates at The University of Texas at Austin as part of the Austin Robot Technology team. We emphasize the aspects of the system that are relevant to multiagent interactions. Specifically, we discuss how our vehicle tracked and reacted to nearby traffic in order to allow our autonomous vehicle to safely follow and pass, merge into moving traffic, obey intersection precedence, and park.**

***Index Terms*—autonomous vehicles, interactive systems, sensor fusion**

I. Introduction

THE DARPA Urban Challenge successfully demonstrated the possibility of autonomous vehicles driving in traffic. The main difference between the Urban Challenge and previous demonstrations of autonomous driving was that in the Urban Challenge, robots needed to be prepared to interact with other vehicles, including other robots and human-driven cars. As a result, robust algorithms for *multiagent* interactions were essential. This article introduces Austin Robot Technology's autonomous vehicle (Figure 1), one of 89 entries in the Urban Challenge. The main contribution is a detailed description of the multiagent interactions inherent in the DARPA Urban Challenge and how our team addressed these challenges.

Austin Robot Technology's entry in the Urban Challenge had two main goals. First, the team aimed to create a fully autonomous vehicle that is capable of safely and robustly meeting all of the criteria laid out in the DARPA Technical Evaluation Criteria document [1], including the multiagent interactions that we emphasize in this article. Second, and almost as important, the team aimed to educate and train members of the next generation of computer science and robotics researchers by encouraging and facilitating extensive participation by undergraduate programmers.

Fig. 1. **The vehicle platform is a 1999 Isuzu VehiCross.** All actuators were developed and installed by the Austin Robot Technology team volunteers.

This article emphasizes the first goal; however, the second goal biases our algorithms to be as straightforward as possible. Nonetheless, the algorithms described here are reliable enough for our team to have placed among the top twenty-one teams at the Urban Challenge National Qualifying Event (NQE). With slightly more luck from prototyped hardware and with a bit more time for testing and verifying code, we believe our autonomous vehicle could have competed well in the final race along with the eleven finalists.

The remainder of this article is organized as follows. Section II provides a brief history of the DARPA autonomous driving challenges. Section III summarizes our specific vehicle platform, including both the hardware and software systems. Section IV presents the main contribution, namely our approach to the multiagent challenges of driving in traffic. Section V summarizes our experience at the Urban Challenge event itself, and Section VI concludes.

II. Background

The first DARPA Grand Challenge was held in 2004 as a competition between academics, military contractors, and amateurs to win a 150 mile autonomous race through the desert. DARPA offered prize money in an effort to spur technological advancements that would lead to one-third of the United States' ground military vehicles being autonomous by 2015. That year, none of the teams made it further than 8 miles. In 2005, the Grand Challenge was held again, and the

Peter Stone is an Associate Professor at The University of Texas at Austin, Department of Computer Sciences. He instructed a Spring 2007 undergraduate course "cs378, Autonomous Vehicles - Driving in Traffic" where undergraduate students became familiar with the robotics problems related to the Urban Challenge.

Patrick Beeson is a doctoral candidate at UT Austin, Department of Computer Sciences. He is the project lead in charge of overseeing undergraduate research and software development on the autonomous vehicle platform. He is teaching the Spring 2008 cs378 autonomous driving course.

Jack O'Quin is a retired IBM operating systems developer and a regular contributor to open source audio programs for Linux. He is a volunteer ART team member that provided immeasurable time and energy designing, programming, and testing most of the control infrastructure. He is also a UT Austin alumnus.

Bartley Gillan, Tarun Nimmagadda, Mickey Ristroph, and David Li were students in Dr. Stone's Spring 2007 course. They all participated in the 2007 DARPA Urban Challenge National Qualifying Event.

course was completed by five teams with Stanford University's team finishing first [2].

Austin Robot Technology (ART) formed as a collection of technologists interested in participating in the 2005 race. In their spare time and with much of their own money, they created an autonomous vehicle that made it to the semi-finals in 2005.

In 2007, DARPA held the Urban Challenge in an attempt to have vehicles race to complete a 60 mile "urban" course in under 6 hours. Carnegie Mellon's Tartan racing team won the race, and six teams completed the course, though only four of these finished under the 6 hour deadline. Unlike the previous races, this race simulated urban (actually more suburban) driving, where the autonomous vehicles had to obey traffic laws and interact with other vehicles on the road.

For the 2007 competition, Austin Robot Technology teamed up with The University of Texas at Austin (UT Austin) via Peter Stone's undergraduate course on autonomous driving. This partnership provided the team with Artificial Intelligence expertise as well as a group of excited undergraduate programmers. It provided the university with an interesting platform on which to offer invaluable undergraduate research opportunities.

The ART team made it to the National Qualifying Event (semi-finals) again in 2007, but was not among the top eleven teams chosen for the final race. Mostly this was due to a shortened development schedule and a few hardware glitches during the qualifying events. In particular the algorithms for interacting with other agents, described in this article, performed well at the event. These other agents may be other robots, or they may be human drivers. Our algorithms make no distinction.

III. Vehicle Overview

Here we give a quick overview of the vehicle's hardware and software before discussing the specific aspects of interacting with other vehicles in Section IV. More hardware details, along with an overview of the software developed by the undergraduate class, can be found in the technical report submitted to DARPA as part of the quarter-final site visit [3].

A. Hardware

The Austin Robot Technology vehicle, in its present configuration, is shown in Figure 1. It is a stock 1999 Isuzu VehiCross that has been upgraded to run autonomously. Austin Robot Technology team members spent much of their time in 2004 and 2005 adding shift-by-wire, steering, and braking actuators to the vehicle. Control of the throttle was achieved by interfacing with the vehicle's existing cruise control system.

In addition to actuators, the vehicle is equipped with a variety of sensing devices. Differential GPS, an inertial measurement unit, and wheel revolutions are combined together by the Applanix POS-LV for sub-meter odometry information. SICK LMS lidars are used for precise, accurate planar range sensing in front and behind the vehicle. A Velodyne High Definition Lidar (HDL) provides 360° 3D range information (see Figure 2).

The vehicle contains three machines with a total of ten AMD Opteron cores, which provides more than enough processing power for the Urban Challenge domain. Additionally a 24V alternator provides power to computers, heavy-duty actuators, and some perception devices, while the vehicle's existing 12V system powers many plug in devices such as network switches, the safety siren, and the safety strobe lights.

B. Software

At the DARPA Urban Challenge, teams were given models of the roadways via Route Network Definition Files (RNDFs). An RNDF specifies drivable road segments, GPS waypoints that make up the lanes in each segment, valid transitions between lanes, stop sign locations, and lane widths. The direction of travel for each lane is provided through the ordering of the GPS waypoints for that lane. An RNDF may also specify any number of "zones", which are open areas of travel with no specific lane information provided. A polygonal boundary consisting of GPS waypoints is given for each zone. Zones are used to describe parking lots, and can have designated parking spots at specified locations. Each RNDF file has one or more associated Mission Data Files (MDFs) that provide an ordered list of GPS waypoints that the vehicle must drive over [5].

Thus, the challenge is to design a system that interacts with the perceptual and actuator devices at the low-level in order to produce a vehicle that can follow the high-level map given by the RNDF and MDF (Figure 3). The vehicle must do so while traveling in legal lanes, using blinkers, following traffic laws, and not colliding with any obstacles.

Key to our success in the DARPA Urban Challenge is the observation that avoiding collisions and obstacles while navigating within a lane is a simpler problem then generic robot navigation. For example, consider driving down a curvy road with another vehicle traveling towards you in an oncoming lane. The other vehicles will pass within perhaps a meter of you, yet the risk of collision is minimal. Without using lane information, the slightest curve in the road could cause even a highly accurate tracking algorithm to predict a head-on collision, necessitating complex path planning. However, under the assumption that any obstacle being tracked will stay within its lane, or within a lane to which it can legally switch, it is safe to continue on in one's own lane. While seemingly risky, this is the implicit assumption of human drivers, and a necessary assumption for any autonomous agent expected to drive like a human.

We discuss navigation and perception using lane information in Section IV. We also discuss the behaviors used for non-lane navigation in parking areas. But first we introduce the software architecture to ground the concepts and ideas that we utilize in modeling other vehicles on the road.

1) Commander: The Commander module operates at the highest level of reasoning. Using the graph built from the RNDF, it determines the optimal route from the current location of the vehicle to the next two goals given by the MDF. We use A* search to find the shortest time path between waypoints, penalizing graph edges that we know are at intersections or that go through parking lots.

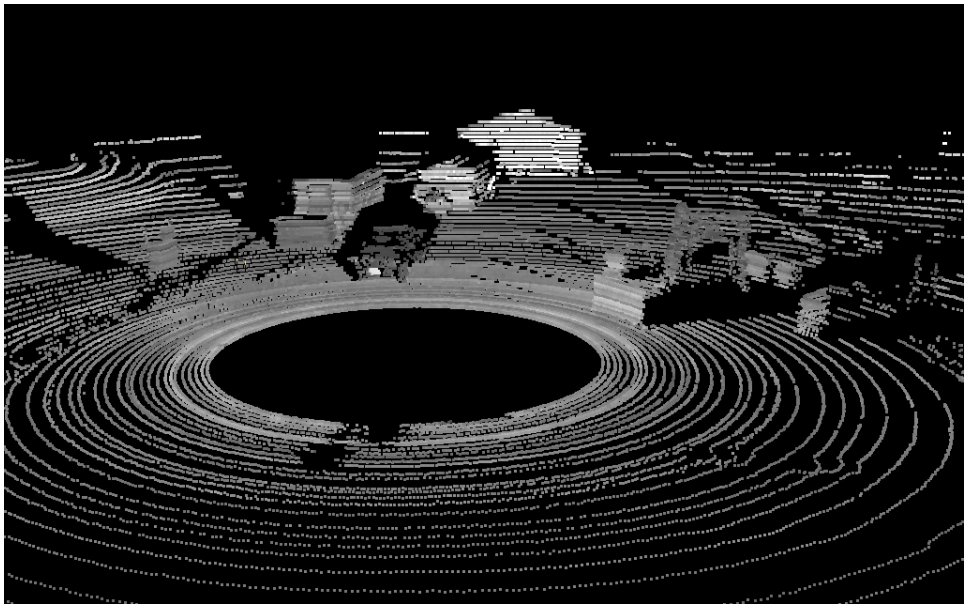


Fig. 2. **Sample snapshot of Velodyne HDL data.** The Velodyne HDL uses lidar technology to return a 360° 3D point cloud of data over a 24° vertical window. Intensity of distance returns are correlated with pixel intensity. Notice that buildings and a truck can be seen among the obstacles.

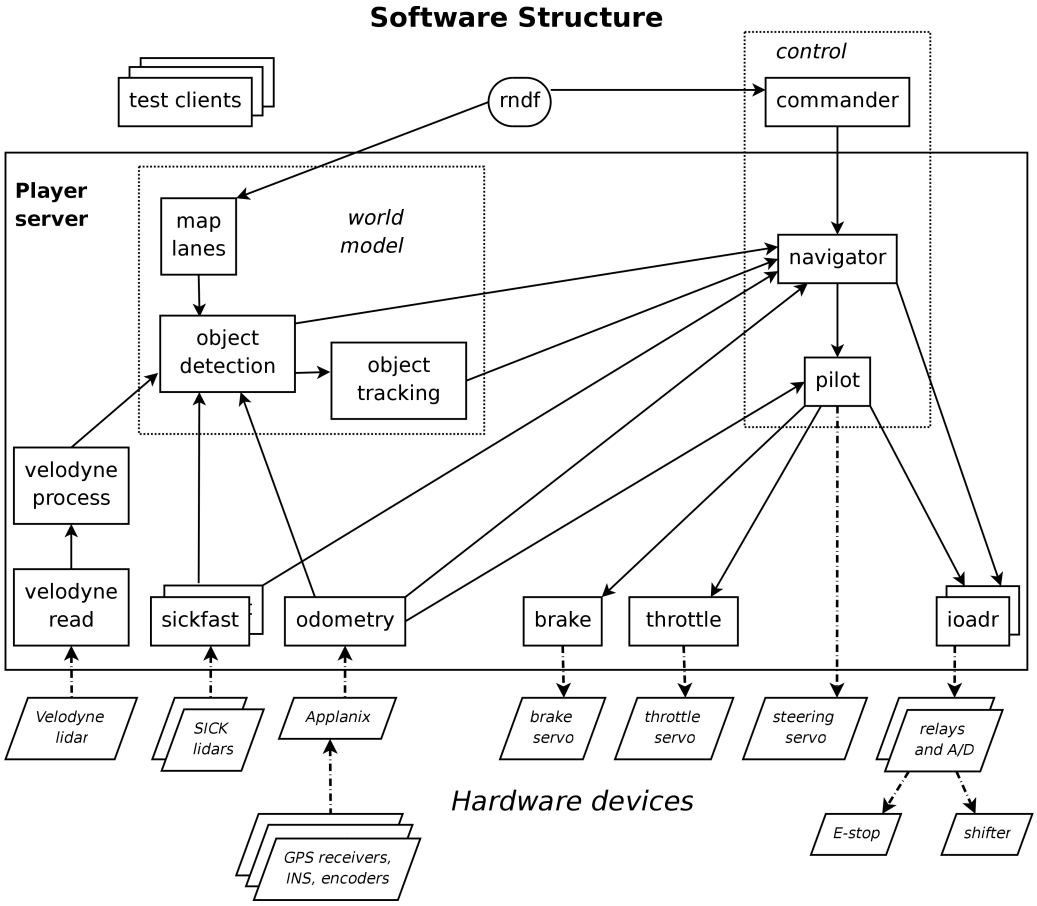


Fig. 3. **Software architecture.** We utilize the Player [4] robot server as our interface infrastructure. Here each module is a separate server, running in its own thread. There is a single client which connects to the top-level server to start the system. Hardware interfaces are shown at the bottom. Perceptual filters and models are on the left side of the diagram while control—planning (commander), behaviors (navigator), low-level actuation control (pilot)—are shown on the right side.

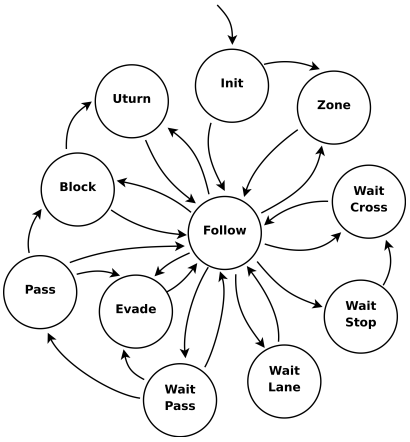


Fig. 4. **A simplified illustration of the vehicle's Run state machine.** Most of the time, the vehicle is in the Follow state. The Navigator module can decide to pass a stalled vehicle, if a passing lane exists, without having the higher-level Commander module replan. Entering other states depends on current traffic conditions or whether the vehicle must enter a parking zone. Many of these are detailed in Section IV.

2) Navigator: The Navigator module is essentially a hierarchical state machine that runs a variety of behaviors on the vehicle. At the highest level is a simple run/pause/disable machine. The Run state contains another state machine, which is illustrated in Figure 4. When running, Navigator uses the next several waypoints in the plan it receives from Commander and runs the appropriate behavior.

Most of the time the vehicle is following its current lane, though many of the interesting behaviors from a multiagent point-of-view occur in the other control states. Due to a shortened development time[1], we did not utilize a traditional model-based route planner for most behaviors. Instead each behavior here uses a snapshot of the world at each cycle to quickly compute a desired travel and turning velocity. The *Pilot* module transforms this velocity command into low-level throttle, brake pressure, and steering angle commands.

3) Velodyne HDL Processing: The Velodyne High Definition Lidar (HDL) provides around one million points of data every second. Following our design principle of trying simple algorithms first, we use "height-difference" maps to identify vertical surfaces in the environment without the need for computationally intensive algorithms for 3D, real-time modeling [6]. Our solution can be thought of as a "slimmed down" version of the terrain labeling method performed by the 2005 Grand Challenge Stanley team [2]. At each cycle (i.e. every complete set of 360° data), we create a 2D (x, y) grid map from the 3D point cloud, recording the maximum and minimum z (vertical) values seen in each grid cell.

Next, a simulated lidar scan is produced from the 2D grid—the algorithm casts rays from the sensor origin, and an obstacle is "detected" whenever the difference between the max and min z values is above a threshold. The result is a 360° 2D simulated lidar scan, which looks very similar to the data output by the SICK lidar devices (see Figure 5); however, this 2D lidar scan is non-planar and only returns the distances to the closest obstacles that have a predetermined vertical measure (currently, 25 cm tall with at least a 45° slope).

[1]Because the undergraduate class' code was designed as a prototype to pass the regional site visit, we overhauled the software completely between July and the October NQE events.

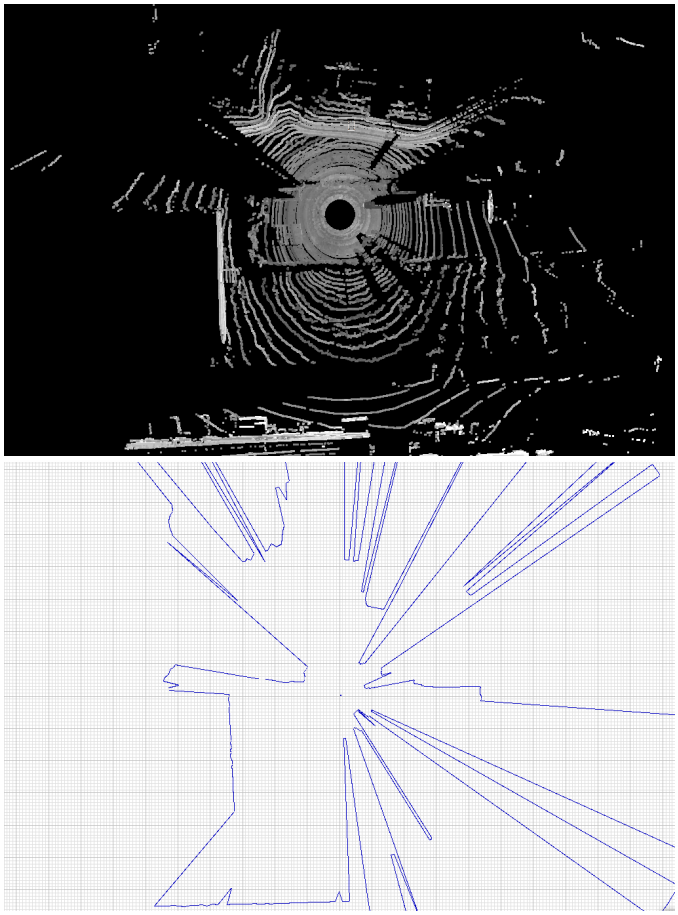


Fig. 5. **Processed Velodyne lidar information.** Raw Velodyne HDL point cloud (bird's eye view of Figure 2 is shown) gets processed into a 2D scan. Notice corresponding features between the two data formulations. This method creates possible occlusions, but allows fast, efficient processing of the million points per second the Velodyne HDL transmits.

4) MapLanes: Initially conceived as a temporary substitute for visual lane recognition, the MapLanes module has become an important piece of our current software infrastructure. MapLanes is designed to parse an RNDF and to create a lane map in the global Cartesian coordinate system provided by the Applanix odometry. Its dual purposes are i) to create lane information useful for vehicle navigation and ii) to provide a way of classifying range data as being in the current lane, in an adjacent lane, or off the road entirely.

The MapLanes road generation algorithm uses standard cubic splines [7], augmented with a few heuristics about roadways, to connect the RNDF waypoints (Figure 6). We first create a C1 continuous Hermite spline [7] from the discrete series of waypoints that define a lane in the RNDF. We chose the Hermite form because its representation allows us to control the tangents at the curve end points. We can then specify the derivatives at the waypoints in such a way that the spline that we create from these curves has the continuity

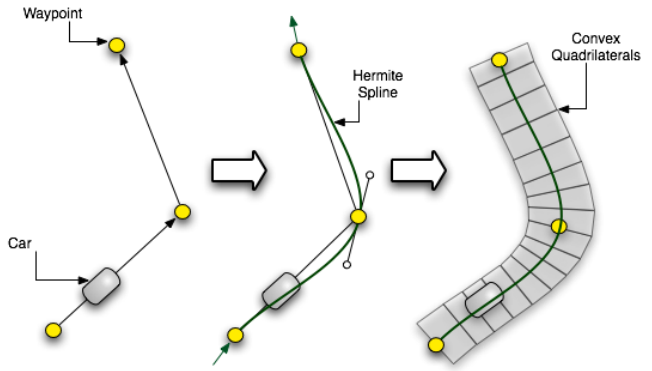


Fig. 6. **Guessing the road shape.** Given waypoints that define a lane, a cubic spline gives a rough approximation of the road. We utilize a few non-standard heuristics to detect straight portions of roadway, which improves the spline tangents at each point. The quadrilaterals that we utilize for the MapLanes module are built on top of the curves. The collection of these labeled quadrilaterals are called *polygons* in our current implementation jargon.

properties we desire.

We then convert the spline from a Hermite basis to the Bézier basis. This step allows us to use any of a large number of algorithms available to evaluate Bézier curves. At this time, we express the curve in terms of nth degree Bernstein polynomials which are defined explicitly by:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \qquad i = 0, \ldots, n.$$

Any point on the curve can be evaluated by:

$$b^n(t) = \sum_{j=0}^{n} b_j B_j^n(t).$$

We set $n = 3$. The coefficients b_j are the Bézier control points.

This spline, along with the lane widths defined in the RNDF, gives the vehicle an initial, rough guess at the shape of the roadway (see Figures 7&8). Each lane is then broken into adjacent quadrilaterals (referred to as polygons in our software) that tile the road model. These quadrilaterals are passed through a Kalman filter where vision-based lane detection can fine tune the lane model or overcome incorrect GPS offsets.[2]

5) Polygon Operations: Each polygon created by MapLanes is placed into a data structure that contains, among other information, the Cartesian coordinates of its four corners, the midpoint of the polygon, the length, the width, the heading of the lane at the polygon's midpoint, the waypoints which the polygon lies between (thus the lane the polygon lies on), the type of lane markings which lie on its boundaries, and a unique ID. An ordered list of polygons for each lane is maintained and published to the perceptual and control modules discussed in Section IV.

We created a polygon library that provides numerous methods for extracting information from the ordered list of polygons. This library performs the bulk of the computation pertaining to the current state of the world surrounding the vehicle. Examples include: filtering out range data not on the road, determining distances along curvy lanes, and determining which lanes will be crossed when passing through an intersection.

[2]For the NQE event, the decision was made to run without vision, as issues such as false positives (shadows, sun flares) and illumination changes are still not adequately handled by our software. To our knowledge only two of the six teams to complete the final course strongly relied on visual lane tracking.

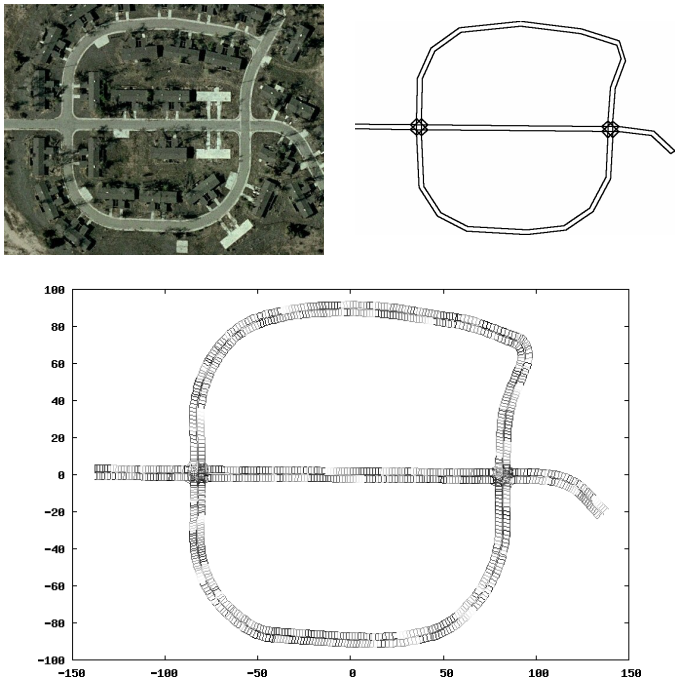


Fig. 7. **MapLanes model of the NQE Area C.** The MapLanes module estimates the continuous shape of the roadway from a simple connected graph of the course extracted from the provided RNDF. Above left is a satellite image of the course. Above right is a graph of the connected RNDF waypoints. The bottom diagram illustrates the MapLanes data structure.

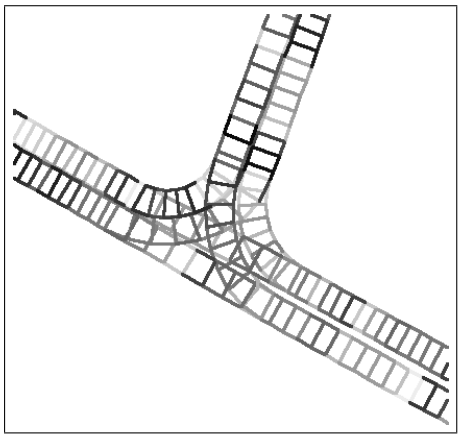


Fig. 8. **Polygons at an intersection.** Lane polygons model each lane while overlapping transition polygons fill in the connections between lanes. This overlapping is how the vehicle determines which lanes to observe when pulling into an intersection: transition polygons of the current lane overlap polygons of lanes that must be assigned obstacle trackers.

IV. Multiagent Interactions

The overall hardware and software architecture sketched in Section III forms the substrate system for the main research reported in this article, namely the vehicle's multiagent interactions. There are two main components to the multiagent

interactions that we consider. First, in Section IV-A, we focus on the ability to *perceive* and represent other vehicles. Second, in Section IV-B, we detail our vehicle's *behaviors* based on those perceptions.

A. Perception

We describe the perception necessary for multiagent interaction in two parts. First, we describe obstacle tracking in a lane using the range data received by lidar sensors. Second, we define a set of *observers*, each of which instantiates an obstacle tracker on the appropriate set of nearby lanes, and reports the situation to the control modules. In a sense, each observer is like a "back-seat driver" that continuously calls out the appropriate information: "unsafe to merge left," "your turn to go," etc.

1) Obstacle Tracking: For autonomous driving, a robot needs good approximations of the locations and velocities of surrounding traffic. Recent approaches to obstacle tracking have often utilized a Cartesian-based occupancy grid [8] for spatial and temporal filtering of obstacles. This occupancy grid is used to estimate the state $X = (x, y, \theta)$, $\dot{X} = (\dot{x}, \dot{y}, \dot{\theta})$, and sometimes the shape or extent of surrounding obstacles [9].

Our design differs from omni-directional tracking in that we utilize the MapLanes model of the roadway to solve the obstacle tracking problem. The key insight in simplifying the problem of obstacle tracking in the urban driving domain is that the vehicle only needs to track obstacles that are within lanes.[3] We further reduce the dimensionality of the problem by observing that it is sufficient to track the velocity of each obstacle only along the lane.

By partitioning the world into lanes and defining an order on the quadrilaterals comprising each lane, we impose a linearization on the space. Thus we can easily track the distance to the closest obstacle in each direction of a lane. The distance function in this space is not Euclidean but rather an approximation of the lane distance as reported by the polygon library. The distance computation between two points first projects each point onto the midline of the lane begin tracked. The lane distance is approximated by using the summation of piecewise line segments connecting the lane polygons.

For a particular lane and direction (in front of or behind our vehicle), we build an obstacle tracker using lidar data. The obstacle tracker for each lane receives a laser scan which specifies the positions of all obstacles that are within its lane. It then iterates through all these obstacles to find the closest one in the specified direction. It maintains a history of these nearest observations using a queue of fixed size. We then filter out noise using acceleration and velocity bounds and estimate the relative velocity from the queue of recent observations. Figure 9 illustrates an experiment where our vehicle was sitting still and tracking another vehicle driving in an adjacent lane.

One advantage of this obstacle tracking approach is that it scales linearly with respect to the number of lanes the vehicle attends to, not the number of obstacles. We found that, even though it had a lossy model of the world, it was powerful enough to complete the various requirement that DARPA outlined in the Technical Evaluation Criteria, and therefore sufficient for most urban driving tasks. During the NQE event, we used a 10 frame queue of distances, which reduced the risk of inaccurate measurements from sensor noise, but introduced a lag of about 1 second in the measurements given the 10Hz lidar updates (see Figure 10). We accepted this lag as a trade-off in favor of robust, safe driving over aggressive behavior.

2) Observers: We define an observer as an object focusing on a subset of MapLanes polygons and lidar range data to determine whether that specific area of the world is deemed free from traffic by our vehicle. Think of an observer as a back-seat driver in charge of reporting whether a specific section of the road is safe to enter or occupied by traffic or obstacles. Each observer sends its report to Navigator every cycle. Navigator chooses which observers are appropriate to use for decision making given its current plan. The primary information each observer provides is a single bit, which represents whether its area is clear or unclear. When an observer reports "unclear," it also provides useful quantitative data such as estimated time to the nearest collision.

Our system uses six observers: Nearest Forward, Nearest Backward, Adjacent Left, Adjacent Right, Merging, and Intersection Precedence. In order for the observers to track vehicles and other objects on the road, they need information about the nearby road lanes. Using the current vehicle odometry and the polygon library, each observer determines whether it is applicable or not based on whether lanes exist relative to the vehicle's pose.

a) Nearest Forward Observer: The Nearest Forward observer reports whether the current lane is clear forward of the vehicle's pose. This perception data is from the Velodyne HDL and the front SICK lidars. This observer reports potential collisions in the current lane.

b) Nearest Backward Observer: The Nearest Backward observer is just like the Nearest Forward observer except that it looks behind the vehicle's current pose. This observer is rarely used by Navigator, as often our vehicle ignores anything approaching from behind in the current lane.

c) Adjacent Left Observer: The Adjacent Left observer reports whether the lane immediately to the left of the vehicle's current lane is safe to enter, for example to pass a stopped vehicle. If a vehicle is in the left lane but the time to collision is larger than 10 seconds, the lane is considered to be clear. Unlike the Nearest Forward and Nearest Backward observers, this observer has two trackers, one in front and one behind. It reports the most imminent threat to Navigator.

d) Adjacent Right Observer: The Adjacent Right observer is just like the Adjacent Left observer except that it looks to the lane immediately right of the vehicle. If there is no lane to the right of the observer, then the observer reports that it is not applicable.

e) Merging Observer: The Merging observer is used to check when it is safe for the vehicle to proceed across any type of intersection: driving through an intersection or turning into/across traffic. Like other observers, it uses trackers to estimate relative velocity of obstacles in lanes and makes a binary safe/unsafe decision based on collision time: $t = d/v$,

[3]The 2007 Urban Challenge specifically ruled out pedestrians or any other obstacles that might move into traffic from off the roadway.

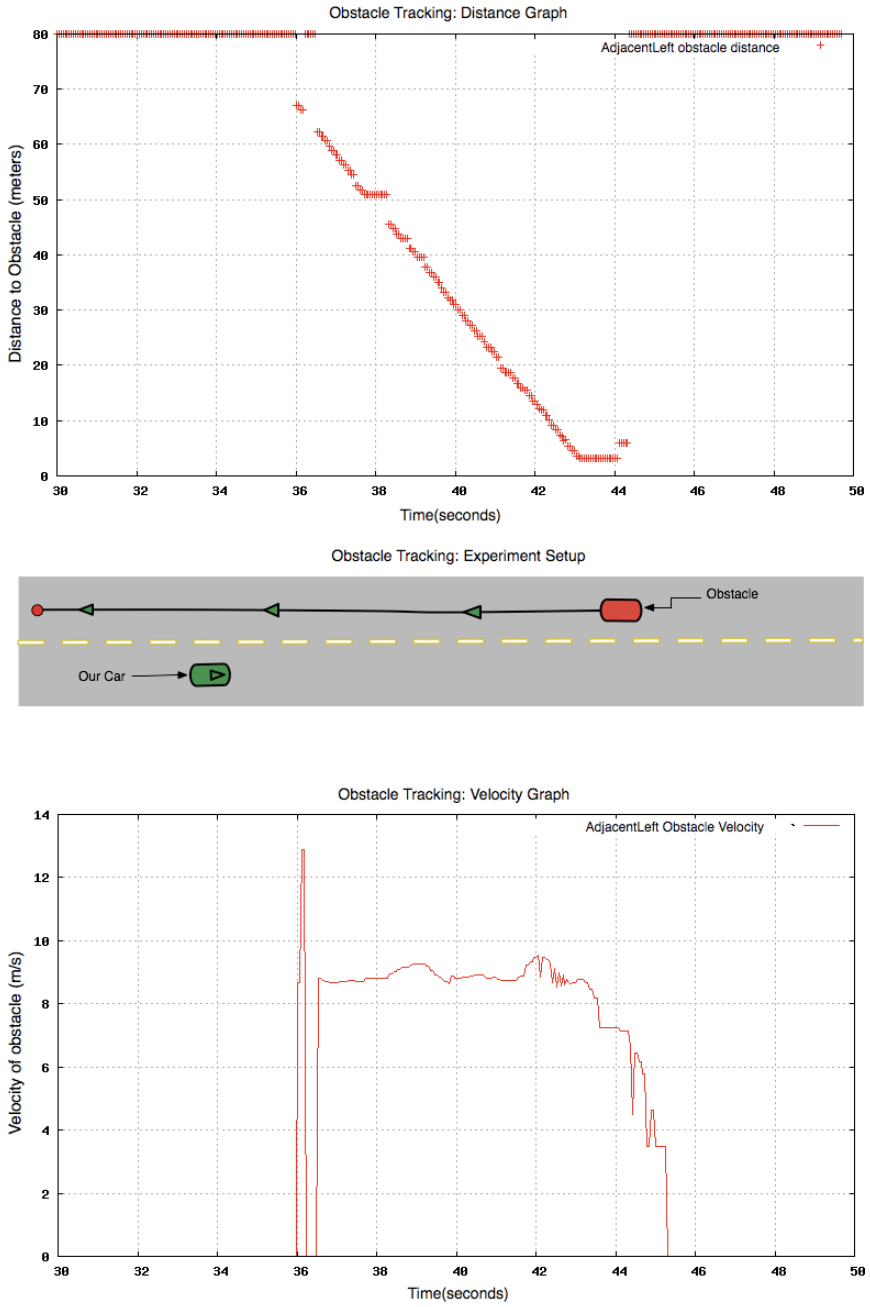


Fig. 9. **Obstacle Tracking Experiment Results.** In this experiment, our vehicle was stopped on a road and tracking another vehicle in the lane to the left. The driver of the tracked vehicle reported an estimated speed of 9 m/s. Other than a brief initial transient, the obstacle tracker accurately models the oncoming vehicle starting from about 60 meters away.

where d is the lane distance, which as described above can be computed between two obstacles in different lanes.

In merging scenarios, the observer currently checks whether all lanes the vehicle *can possibly* traverse are safe. For example, at an intersection, if it is unsafe to turn left, the vehicle will wait, even if it plans to turn right. This behavior is necessitated by the modular design of our system, in that the observers are not aware of the vehicle's plan of action. Note that the effect is more conservative behavior, which may be appropriate given the fact that other vehicles can also be autonomous, thus perhaps unpredictable.

Choosing the set of lanes to assign trackers to is a critical task. A general polygons-based solution was developed for this purpose that looks at *lane* versus *transition* polygons. Lane polygons are the quadrilaterals that are adjacent along the lane (illustrated in Figure 6), while transition polygons connect up different lanes (see Figure 8). We build a table that holds information regarding which lanes overlap each other in the nearby surround. We know a lane crosses another if any transition polygon in the current lane intersects any polygon (lane or transition) of the other lane. Thus a simple point-in-polygon algorithm can be looped over polygons to build the table.

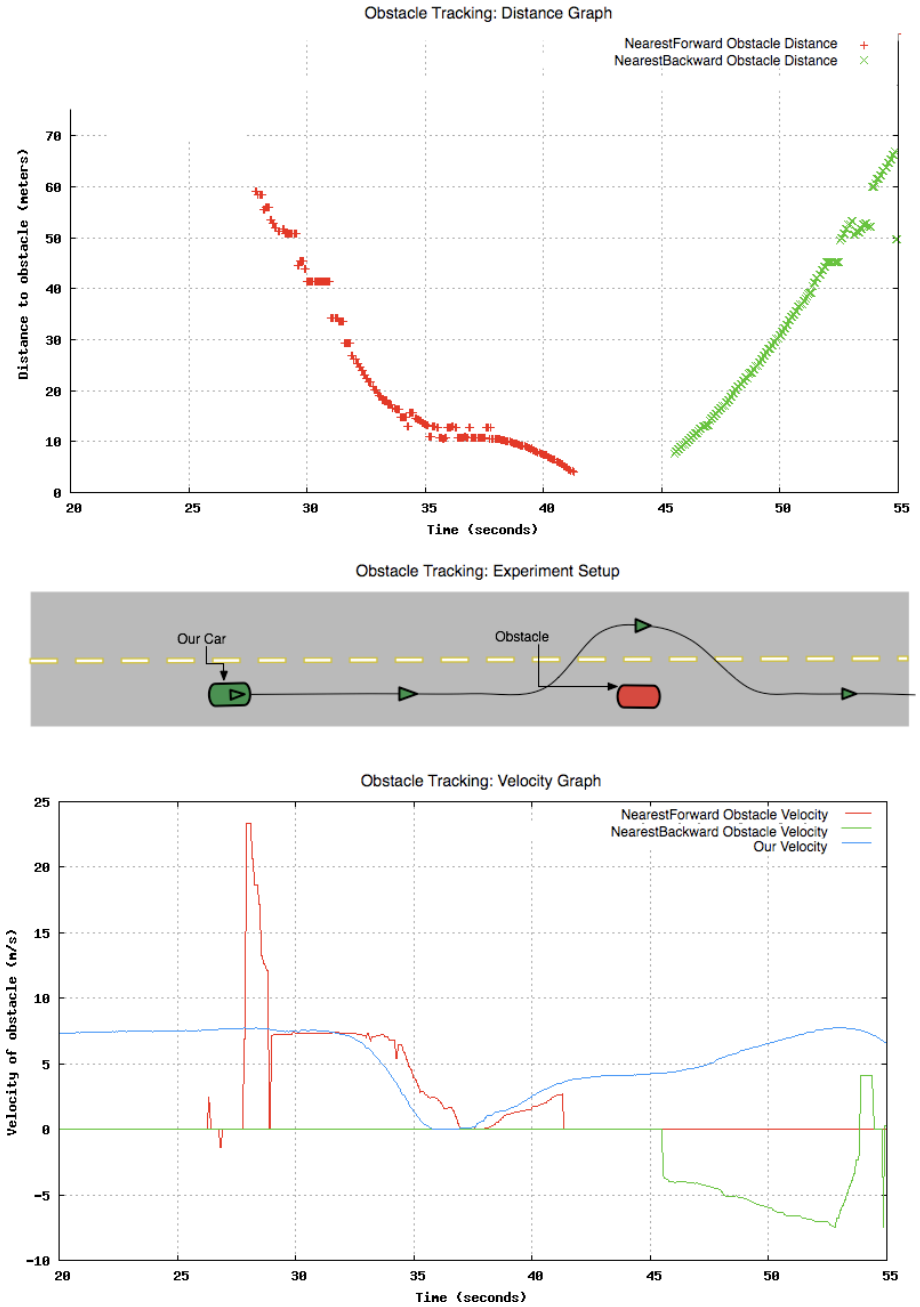


Fig. 10. **Obstacle Tracking Experiment Results.** In this setup, our vehicle drove up to another vehicle that was stopped in the same lane. Our vehicle then passed before returning to the original lane of travel. The above graphs show the relative state of the obstacle estimated by two trackers: one forward and one behind in the current lane.

f) Intersection Precedence Observer: Upon stopping at a stop sign (defined as a special waypoint in the RNDF and marked by a white stop line on the roadway), the vehicle needs to yield to other vehicles that are already stopped at the intersection. The vehicle must then proceed through the intersection when its turn arrives, even if other vehicles have arrived at the intersection later.

The Intersection Precedence observer is used to indicate whether or not it is appropriate to start moving through the intersection. By default, the observer reports that the vehicle should wait. Once it becomes the vehicle's turn to move, the observer reports that it is time to go. This observer does not rely on obstacle tracking as it can be difficult to separate vehicles that pass each other in the intersection. Instead the simple yet effective idea behind this observer is to notice which lanes at the intersection have obstacles (presumably other vehicles) waiting at the instant that our vehicle arrives, and then to notice when these obstacles disappear from each such lane.

There are 3 main components to our intersection precedence observer.

1) *Locate the other stop signs at the same intersection.* The RNDF does not explicitly define the structure of intersections. It can be deduced through complex modeling;

however, we simply locate all stops that are within 20 meters of our vehicle's stop waypoint and are in front of our vehicle.[4]

2) *Determine which stop signs have vehicles with higher priority.* Immediately when the vehicle stops, it needs to determine which stop signs are "occupied."
First, we filter all range readings that fall outside lanes that have stop signs at the intersection. Then we throw out any readings further than 2 meters of a stop sign location—stopped vehicles are supposed to be within 1 meter of the stop sign. Next, we bin all remaining range data for forward facing angles into 180 bins, each representing 1°, and we determine the closest reading in each bin to the vehicle.
The resulting 180 element vector, μ, is our model of the intersection. Most elements in μ will have a zero range (because most range readings have been filtered out); non-zero elements are assumed to indicate vehicles that arrived at the intersection before us, and thus have the right of way. To reduce false positives/false negatives, we collect this data over several cycles immediately upon arriving at the stop sign.
3) *Determine when all higher priority vehicles have gone.* On each subsequent cycle, the vehicle gathers new range readings, and produces a new 180 element vector of the closest ranges at 1° intervals, ν. We update the model of the intersection by declaring that if $\nu_i - \mu_i \geq 2$, then $\mu_i = 0$. Note that once an element of μ is zero, is remains zero. Once all elements of μ equal zero, it is our vehicle's turn to go. This procedure works even when multiple vehicles are queued up in other lanes due to the fact that a small gap always exists between a vehicle leaving and another vehicle pulling up. Note that we actually produce ν using several cycles of range data in order to handle false negatives.

B. Behaviors

Perception and tracking of other vehicles as described in Section IV-A are necessary prerequisites for designing effective multiagent behaviors. This section details the behaviors that we created to interact with traffic in the Urban Challenge domain. With good input from the perception subsystem, these behaviors can be relatively straightforward.

The behaviors are all implemented in the Navigator thread using a hierarchy of controllers. Every 50 msec cycle, Navigator creates a Pilot command indicating a desired velocity v and yaw rate ω for the vehicle. Pilot, running at the same rate in a separate thread, translates those commands into steering, brake, throttle and shifter servo motor instructions.

Each controller in the Navigator module provides an interface that can modify the next Pilot command according to the current situation. Some controllers are finite state machines, others simple code sequences. Every control method also returns a "result" code, which the calling controller often uses to determine its future states and activities.

[4]This does not handle degenerate cases, but it is suitable for most environments.

Figure 4 illustrates how these behaviors are connected in the main Navigator state machine, which is itself a controller. Its states and state transitions all have associated actions, which support the same controller interface. Most Navigator controllers follow lanes in the road network, utilizing MapLanes data to orient the vehicle within its lane. The major exception is the "Zone" controller, which operates in unstructured areas such as parking lots.

1) Follow Lane: The FollowLane controller, designed to follow an open lane, is the behavior that is executed most often. It is executed in the Follow state which appears at the center of Figure 4. Due to a shortened development period between the regional site visit and the NQE, we chose not spend time developing a planner that models vehicle dynamics. Instead we implemented a simple linear spring system.

The spring system is based on an assumed constant velocity v, the lateral offset with respect to the center of the lane Δl, and the heading offset with respect to the lane $\Delta\theta$. The value of v is set before this computation based on local obstacles, the distance to a stop sign, or the curvature of the lane ahead. We gather the lane heading and location using the closest lane polygon just past the front bumper of the vehicle. The vehicle steers in the lane using the following equation:

$$\omega = -k_\theta \Delta\theta - \frac{k_l}{v}\Delta l,$$

where both k_θ and k_l were experimentally determined to be 0.5.

Avoidance of obstacles near the edge of lanes is accomplished by simply changing Δl to edge the vehicle to the other side of the lane. When the obstacle is far into the lane, the vehicle stops with the Blocked result code, which may eventually lead to passing in another lane or performing a U-turn.

2) Follow at a safe distance: While following a lane, this controller checks whether there is another vehicle or obstacle ahead, matching speeds while maintaining a safe distance. Note that the obstacle may be stationary, in which case the vehicle will stop at an appropriate distance behind it, with the controller returning a Blocked result. This behavior can be used in smooth-flowing traffic to maintain at least the standard 2-second following distance, or in stop-and-go traffic. The pseudocode for follow safely is in Algorithm IV.1.

3) Intersection Precedence: When the Follow controller reaches a stop sign way-point, it returns Finished, causing a transition to the WaitStop Navigator state. This transition runs ActionToWaitStop(), followed by ActionInWaitStop() in each subsequent cycle. Algorithm IV.2 gives the pseudocode for these two subcontrollers.

The guidelines for the Urban Challenge specify that if another vehicle fails to go in its turn, the vehicle should wait 10 seconds before proceeding cautiously. Our implementation uses two timers. The *stop_line_timer* gives the Intersection Precedence observer one second to gather information about other vehicles already stopped at this intersection. Meanwhile, the *precedence_timer* starts counting up to 10 seconds each time the number of vehicles ahead of us changes.

When there are none left or the *precedence_timer* expires, we set the *pending_event* class variable to Merge, which

Algorithm IV.1: FOLLOWSAFELY($speed$)

$result \leftarrow$ OK
$distance \leftarrow$ range of closest obstacle ahead in this lane
if $distance \geq maximum\_range$
  then return $(result)$
$following\_time \leftarrow distance/speed$
if $\begin{cases} following\_time \leq min\_following\_time \textbf{ or} \\ distance \leq close\_stopping\_distance \end{cases}$
  then $\begin{cases} speed \leftarrow 0 \\ \textbf{if} \text{ already stopped} \\ \quad \textbf{then } result \leftarrow \text{Blocked} \end{cases}$
  else if $following\_time < desired\_following\_time$
  then decrease *speed* to match obstacle
  else if $\begin{cases} \text{already stopped } \textbf{or} \\ following\_time > desired\_following\_time \end{cases}$
  then increase *speed* to match obstacle
if $\begin{cases} \text{Nearest Forward observer reports obstacle approaching } \textbf{and} \\ \text{closing velocity is faster than our current velocity} \end{cases}$
  then $result \leftarrow$ Collision
return $(result)$

Algorithm IV.2: WAITSTOP($speed$)

procedure ACTIONTOWAITSTOP($speed$)
 set turn signal depending on planned route
 start 1 second *stop_line_timer*
 start 10 second *precedence_timer*
 $prev\_nobjects \leftarrow -1$
 return (ACTIONINWAITSTOP($speed$))

procedure ACTIONINWAITSTOP($speed$)
 $speed \leftarrow 0$
 if *stop_line_timer* expired **and** Intersection observer reports clear
  then $pending\_event \leftarrow$ Merge
 $obs \leftarrow$ current Intersection observation data
 if $obs.applicable$ **and** $obs.nobjects \neq prev\_nobjects$
  then $\begin{cases} prev\_nobjects \leftarrow obs.nobjects \\ \text{start 10 second } precedence\_timer \end{cases}$
 if *precedence_timer* expired
  then $pending\_event \leftarrow$ Merge
 return (OK)

triggers a transition in the next cycle, in this case to the WaitCross state, which handles Intersection Crossing.

4) Intersection Crossing: When the vehicle has reached an intersection and is ready to proceed, Navigator changes to its WaitCross state. As the state transition diagram in Figure 4 shows, this may either happen from the WaitStop state after intersection precedence is satisfied, or directly from the Follow state if there is no stop sign in our lane (e.g. turning left across traffic). In either case, the vehicle has already stopped. It remains stopped while waiting for the intersection to clear.

The WaitCross control simply activates the appropriate turn signal based on the planned route and waits until the Merging observer reports at least a 10 second gap in surrounding traffic. It then transitions to the Follow controller, which guides the vehicle through the intersection and cancels the turn signals after reaching the next lane of the desired route.

5) Pass: The intersection state transitions provide a simple introduction to the more complex transitions involved in passing a stopped vehicle or other obstacle blocking the desired travel lane.

Our current implementation never passes moving vehicles. In the Follow state, Navigator matches speeds with any vehicle ahead in our lane. As described in section IV-B2, it only returns a Blocked result after the vehicle comes to a complete stop due to a stationary obstacle, which could be a parked vehicle or a roadblock.

A Blocked result in the Follow state initially triggers a transition to the WaitPass state (Algorithm IV.3). Next, Navigator attempts to find a passing lane using the polygon library. If none is available, the situation is treated as a roadblock, causing a transition to the Block state, and initiating a request for the Commander module to plan an alternate route to our next checkpoint, usually beginning with a U-turn. Because a roadblock is a static obstacle not representing a multiagent interaction, we focus more deeply on the case where a passing lane exists, allowing us to pass a parked vehicle blocking our lane.

When waiting to pass, two things can happen. If the obstacle moves within several seconds, Navigator immediately returns to the Follow state. If the obstacle remains stationary, Navigator changes to the Pass state as soon as no vehicle is approaching in the passing lane.

In the Pass state (Algorithm IV.4), Navigator saves the polygon list of the selected passing lane, and invokes the Passing controller. This controller uses the same linear spring system as the FollowLane controller to follow the polygons in the passing lane. It returns Finished when it detects that the vehicle has passed all obstacles in the original lane. Navigator then returns to the Follow state, where the FollowLane controller uses the polygons for the original travel lane to guide the vehicle back.

6) Evade: This controller runs when the main Navigator state machine is in the Evade state. We reach that state after some other controller returns a Collision result, having noticed that the Nearest Forward observer saw something driving towards our vehicle in the current lane. Having a closing velocity with respect to an obstacle does not imply a collision event. Only if the relative velocity is significantly greater than our vehicle's velocity, do we decide to evade.

The Evade controller's job is to leave the lane to the right, wait until there is no longer a vehicle approaching, then return Finished. Navigator then returns to the Follow state. Other evasion techniques could be used. Our approach implements the recommended behavior in the DARPA Technical Evaluation Criteria document [1].

This controller has a simple state machine of its own. Algorithm IV.5 gives pseudocode for each state, the appropriate procedure being selected by the *state* variable in each 20Hz Navigator cycle. When reset on transition to Evade, it begins in the Init state. The Leave state invokes a private leave_lane_right() method, also shown. It calls the LaneEdge controller as long as the lane to the right is clear. That controller steers the vehicle outside the right lane boundary to avoid a head-on collision.

7) Obstacle Avoidance in Zones: In our compressed development schedule, driving in zones was largely put off until

Algorithm IV.3: WAITPASS(*speed*)

procedure ACTIONTOWAITPASS(*speed*)
if FIND_PASSING_LANE()
then { set turn signal for passing direction; start 5 second *passing_timer*; **return** (ACTIONINWAITPASS(*speed*))
else { *pending_event* ← Block; *speed* ← 0; **return** (Blocked)

procedure ACTIONINWAITPASS(*speed*)
if *passing_timer* expired **and** observer reports passing lane clear
then *pending_event* ← Pass
result ← FOLLOWSAFELY(*speed*)
if *result* = OK
then *pending_event* ← FollowLane
else if *result* = Collision
then *pending_event* ← Collision
speed ← 0
return (*result*)

Algorithm IV.4: PASS(*speed*)

procedure ACTIONTOPASS(*speed*)
if SWITCH_TO_PASSING_LANE()
then { reset Passing controller; **return** (ACTIONINPASS(*speed*))
else { *pending_event* ← Block; *speed* ← 0; **return** (Blocked)

procedure ACTIONINPASS(*speed*)
result ← PASSING(*speed*)
if *result* = Finished
then { *pending_event* ← FollowLane; *result* ← OK
else if *result* = Blocked **and** blockage has lasted a while
then *pending_event* ← Block
else if *result* = Collision
then *pending_event* ← Collision
speed ← 0
return (*result*)

Algorithm IV.5: EVADE(*speed*)

procedure EVADE_INIT(*speed*)
set right turn signal on
state ← Leave
return (EVADE_LEAVE(*speed*))

procedure EVADE_LEAVE(*speed*)
if still in lane
then { **if** Nearest Forward observer reports vehicle approaching
then *result* ← LEAVE_LANE_RIGHT(*speed*)
else { *speed* ← 0; set left turn signal on; *result* ← Finished
else { set both turn signals on; start *evade_timer*; *state* ← Wait; *result* ← EVADE_WAIT(*speed*)
return (*result*)

procedure EVADE_WAIT(*speed*)
speed ← 0
if *evade_timer* expired
then { set left turn signal on; *state* ← Return; **return** (EVADE_RETURN(*speed*))
return (OK)

procedure EVADE_RETURN(*speed*)
cancel *evade_timer*
speed ← 0
if Nearest Forward observer reports vehicle approaching
then return (OK)
else return (Finished)

procedure LEAVE_LANE_RIGHT(*speed*)
limit *speed* to 3 meters/second
result ← Unsafe
if Adjacent Right observer reports clear
then *result* ← LANEEDGE(*speed*, outside right)
return (*result*)

just before the NQE. Not having implemented a model-based planner for lane navigation left us with a large piece missing when it came to driving in the less restricted zone areas. Rather than writing a model-based planner, we utilized an off-the-shelf skeleton algorithm called EVG_Thin [10] to get a coarse route that the vehicle could follow between the zone entry, the zone exit, and any parking spots.

Inside of a zone, we use the perimeter polygon given by the RNDF, the parking waypoints, and any observed obstacles to generate a new skeleton every cycle (see Figure 11). The thinning-based skeleton is an approximation of the Voronoi graph [11], thus it connects points that are maximally far from any two obstacles (a criterion we find quite nice in its aversion to danger). Because we use a Voronoi-style skeleton, we also have the distance to the closest obstacle for each point. We call this distance the point's *safety radius*.

Our controller relies on the fact that if two points are within

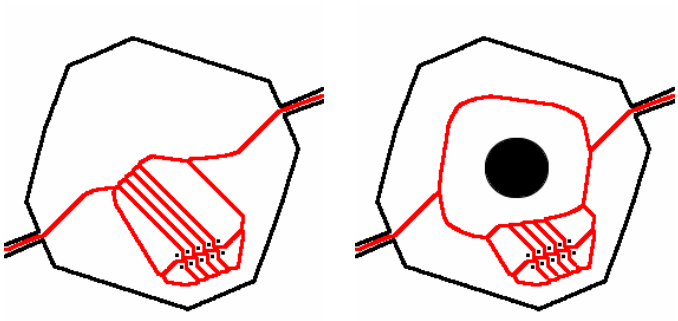


Fig. 11. **Thinning-based skeleton of parking zone.** The black lines connect the perimeter points that define the zone boundaries. The black dots represent the locations of five parking spots. The zone boundaries and parking spots are seen as obstacles along with any obstacles (e.g. the circle in right image) detected by the lidar sensors. The lighter line illustrates the thinned skeleton given local obstacles. This coarse path that avoids obstacles is created at every timestep.

Algorithm IV.6: ZONECONTROLLER($zone, obstacles, current\_location, goal$)

```
procedure PATHTHROUGHZONE()
 graph ← EVG_THIN(zone, obstacles)
 start_node ← NULL
 end_node ← NULL
 for each node ∈ graph
   do { if node.safety_radius ≤ minimum_safety_radius
        then remove node from graph
        else { if { current_location within node.safety_radius of node and
                  { goal within node.safety_radius of node
                then return (empty path)
             if { current_location within node.safety_radius of node and
                  { node is closer to curent_location than start_node
                then start_node ← node
             if { goal within node.safety_radius of node and
                  { node is closer to goal than end_node
                then end_node ← node
 path ← A*(graph, start_node, end_node)
 return (path)

procedure NEXTAIMPOINTINZONE()
 path ← PATHTHROUGHZONE()
 if path is empty
   then { comment: Straight shot to goal.
        { return (goal)
 node ← last node in path
 while start_location is not within node.safety_radius of node
   do node ← previous node in path
 if goal is within node.safety_radius of node
   then aim ← goal
   else aim ← node
 return (aim)
```

the safety radius of the same skeletal node, the straight line between those two points will not cross any obstacles. This fact allows us to find potentially far away nodes along the skeleton which the vehicle can aim straight for without running into obstacles. In this manner, we ensure the vehicle avoids all obstacles, without going unreasonably far out of its way to follow the Voronoi diagram precisely. Algorithm IV.6 details the procedures that are called 10 times per second, attempting to constantly move the vehicle directly towards the furthest safe point.

This controller does not consider the exact vehicle dynamics when planning a path, and does not respect parking lot navigation conventions, such as passing an approaching vehicle on the right. However, by using the safety radius information to aim at far away points, we still get reasonably smooth control.

8) Park: Some MDFs require the autonomous vehicle to park before it exits a zone. A parking spot is defined in the RNDF by two GPS waypoints (see Figure 12) and a spot width. One waypoint defines the location of the entry to the spot, and the other indicates the desired location of the front bumper when the vehicle is properly parked.

There are three main components to the parking behavior. First, the vehicle must get close to the parking spot. This step is done by using the Voronoi zone planner to get to a point near the entry to the parking spot.

Second, the vehicle must determine the exact location of the spot. Given no surrounding obstacles, the vehicle simply uses its GPS-based odometry to define the spot location. With obstacles nearby, the spot location is fine-tuned. The fine-tuning is done by simply defining a rectangle that corresponds to the width of the spot and the length of the spot (the two waypoints plus some extra room determines the length). A discrete search over a predetermined area (1x1 meter offset) is performed to find the spot location that keeps farthest from nearby obstacles.

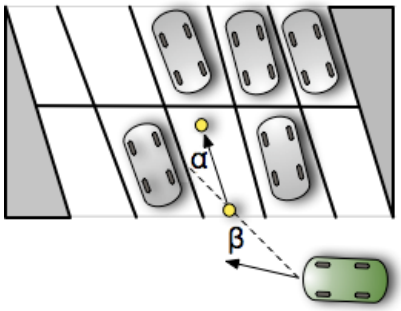


Fig. 12. **Geometric constraints used in parking.** Behavior-based parking tries to minimize both the bearing offset between the front of the vehicle and the beginning of the spot (β) and the heading offset of the vehicle with the parking spot ($\gamma = \alpha + \beta$).

Third, the vehicle must pull into the spot then reverse out of the spot. This can be broken into four sub-behaviors. i) The vehicle ensures its front bumper is over the GPS waypoint at the spot entry and that it is aligned with the spot. ii) It then pulls straight into the spot until the front bumper is over the

Algorithm IV.7: ALIGN WITH SPOT($spot\_pose$)

if too far away from spot entry
then { **comment:** go back to zone controller; **return** (NotApplicable)
comment: In Figure 12, $\gamma = \alpha + \beta$
$\beta \leftarrow$ Bearing to $spot\_pose$
$\gamma \leftarrow$ Heading offset wrt $spot\_pose$
comment: Angles are normalized to be between (-180,180]
if $|\beta| < 15^\circ$ **and** $|\gamma| < 15^\circ$
then { **comment:** aligned with spot entry; **return** (Done)
comment: The space around vehicle is divided into 4 quadrants:
in front, in back, left, and right of vehicle.
comment: front $\equiv$ [-45°,45°), left $\equiv$ [45°,135°),
back $\equiv$ [135°,225°), right $\equiv$ [225°,315°)
comment: v is forward velocity; ω is rotational velocity
$v \leftarrow 0.5$ (m/s)
$\omega \leftarrow \beta$ (degrees/s)
if β is in front quadrant
then { **if** γ is in front quadrant **and** $|\gamma| > |\beta|$ **then** $\omega \leftarrow \gamma$
if β is in left quadrant
then { **if** γ is in front quadrant **or** γ is in right quadrant **then** $v \leftarrow -v$; **else if** γ is in left quadrant **and** $|\gamma| > |\beta|$ **then** $\omega \leftarrow \gamma$
if β is in right quadrant
then { **if** γ is in front quadrant **or** γ is in left quadrant **then** $v \leftarrow -v$; **else if** γ is in right quadrant **and** $|\gamma| > |\beta|$ **then** $\omega \leftarrow \gamma$
if β is in back quadrant
then { **if** γ is in front quadrant **then** { $v \leftarrow -v$; $\omega \leftarrow 0$ }; **else if** γ is in left quadrant **or** γ is in right quadrant **then** { $v \leftarrow -v$; $\omega \leftarrow \gamma$ }
comment: η is a tuning parameter
$\omega \leftarrow \eta \cdot \omega$
if (v, ω) is unsafe
then { $v \leftarrow -0.5$; $\omega \leftarrow \eta \cdot \gamma$; **return** (Done)

second waypoint. iii) It reverses straight back until the front bumper is again over the entry waypoint. iv) Finally it reverses further, turning to face the appropriate direction to continue its plan.

Pulling into and out of the spot once aligned with the spot is straightforward. The pseudocode in Algorithm IV.7 explains the more complex behavior that gets the vehicle aligned to pull directly into the spot.

9) Escape (or Traffic Jam): In cases where the vehicle cannot make progress, it must get unstuck. To do this, we construct a "zone" on the fly. This temporary zone is large enough to encompass our vehicle, the next waypoint in the current plan, and nearby obstacles. We then invoke the same zone controller used in parking lots. In this manner, we continue to make forward progress, though the vehicle may leave the road if there is no alternative path forward. If forward progress is unsafe, the vehicle attempts to reverse. If no safe action can be taken, the vehicle waits until a safe action is applicable.

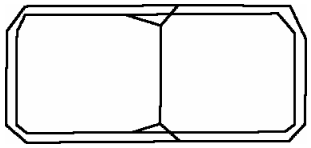


Fig. 13. **Waypoint graph of Area A.** Vertices correspond to GPS waypoints from the RNDF, and edges denote explicit connections from the RNDF.

V. THE URBAN CHALLENGE NQE EXPERIENCE

The multiagent behaviors described in Section IV were put the test at the Urban Challenge National Qualifying Event (NQE), where our vehicle was placed in several challenging multiagent environments. This section describes our experiences at the October NQE as one of the 35 teams invited to participate after having successful site visits in July.

The NQE had three areas meant to test the abilities necessary to safely and effectively navigate the Urban Challenge final course. Area A required the vehicle to merge into and turn across a continuous stream of traffic (eleven human-driven cars operating at 10 mph). This area was the most challenging and was deemed "the essence of the Urban Challenge" by DARPA director Dr. Tony Tether. The challenges in Area B were focused on parking, avoiding static obstacles on the road, and long-term safe navigation. No moving vehicles or dynamic obstacles were encountered in this area. Area C required vehicles to pass a combination of intersection precedence scenarios with human drivers. Roadblocks were added in the middle of the course to force vehicles to perform U-turns and replan routes. This area was similar to the site visit test, which teams were required to complete before being invited to the NQE.

The algorithms described above were reliable enough for our team to place in the top twelve to twenty-one teams at the NQE. With a bit more time for integration testing, we believe that we could have done better. After diagnosing an Ethernet cable failure and tracking down a memory leak in third-party software, we believe we could have competed well in the final race along with the eleven finalists.

A. Area A

The Area A course consisted of a two lane road in a loop with a single lane bypass running north-south down the middle (see Figure 13). Eleven human-driven cars made laps in both directions on the outer loop, while the autonomous vehicles were commanded to perform counter-clockwise laps on the east half of the course. The autonomous vehicle was required to turn across traffic when turning into the bypass, as well as merging back into the main loop when exiting the bypass.

Key to this course was the ability to successfully judge when a sufficient opening was available in traffic. Being overconfident meant cutting off the human drivers and getting

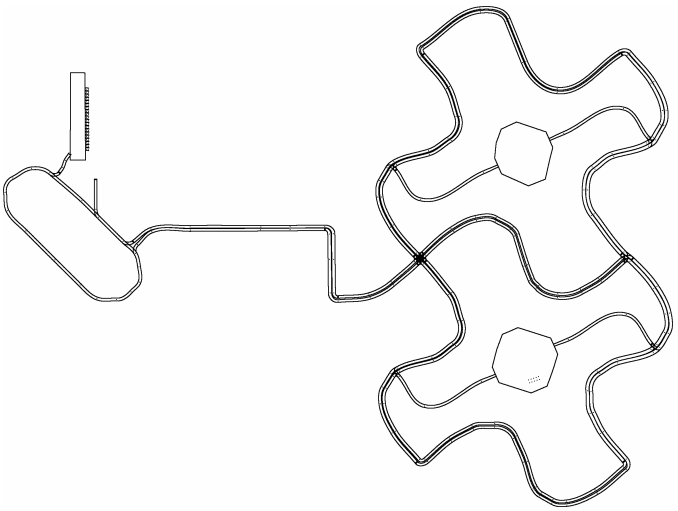


Fig. 14. **MapLanes outline of Area B.** Note the starting chutes in the top left. There are also two large zones, the lower of which has defined parking spaces.

penalized, while being too cautious could result in waiting indefinitely. Our observers were able to successfully detect openings in traffic, resulting in very good multiagent behavior. Our vehicle was able to complete seven laps in the first half-hour run without contact with either vehicles or barriers.

The south edge of the course was lined with k-rails (3 feet high concrete barriers) at the curb for the safety of the judges and spectators. The proximity of the k-rails to the road caused problems for many teams when turning out of the bypass. When making this sharp left turn, these k-rails were seen by our vehicle as obstacles in our path, resulting in our low-level safety controller returning a Blocked event, and thus prohibiting completion of the turn before merging back into the loop (the center bottom of Figure 13). The vehicle eventually called the Escape controller, backed up, and continued down the lane.

This problem was easily overcome by turning down the safety thresholds slightly. However, by the time of our second run in Area A, the race officials had moved these barriers away from the lane boundary by a few feet, since many teams were having problems with this one turn. The vehicle was only able to complete two partial laps during our second run due to a defective Ethernet cable, which dropped most of the data packets from our Velodyne lidar unit. This was an unfortunate hardware failure that was unrelated to our trackers or behaviors.

B. Area B

In Area B, each autonomous vehicle was randomly assigned to one of the start chutes that were used for the start of the final race. The vehicles needed to make their way out of the start zone, down a narrow corridor, and around a traffic circle before proceeding west to the main section of the course, which tested parking and avoiding many static obstacles in the road. Figure 14 illustrates the course.

Our vehicle successfully exited the start zone and made it through the corridor, which had given several prior teams trouble. It then followed the course through several turns before arriving at the first parking lot. The lot contained three parked vehicles surrounding the designated parking spot. Our vehicle parked into and reversed out of the parking space flawlessly. As the vehicle was turning from the path leading out of the parking lot to the main road, a previously undetected memory leak in third-party software caused our control program to crash, ending our run.

Unfortunately, this memory leak occurred in both of our Area B runs, both times just after leaving the parking test area. We eventually traced the root cause back to building Player/Stage [4] with a buggy version of the C++ Standard Template Library.[5] The parking lot was large enough to trigger a memory leak in the STL vector code, which we had never seen in testing. Recompiling Player with the current STL libraries eliminated this memory leak, but unfortunately this solution was not discovered until after both of our Area B runs.

C. Area C

Area C consisted of a large loop with a road running east-west through the center, forming two four-way intersections (Figure 7). Challenges on this course included intersection precedence as well as roadblocks that required replanning the route to the next checkpoint. Human-driven cars provided the traffic at the intersections; however, there was no traffic on other sections of the course. The human drivers retreated to driveways alongside the road when not needed.

Our vehicle made a perfect run, successfully handling intersections with two, three, and four cars queued up when it arrived. Intersections with either two or three cars queued required the vehicle to wait until they had all cleared the intersection before proceeding. The intersection test with four cars had two of the cars queued up in the same lane, which required our vehicle to only wait for the first three cars to clear the intersection and proceed without waiting for the fourth car.

VI. Conclusion and Future Work

This article presented the autonomous vehicle developed by Austin Robot Technology for the 2007 DARPA Urban Challenge. Specifically we focused on the effective multiagent algorithms programmed in part by UT Austin students in only a few months time. We detailed the perceptual algorithms necessary to model other vehicles and the behaviors necessary to drive in the presence of other vehicles. We provided algorithms used in the system to merge, pass, follow, park, track dynamic obstacles, and obey intersection laws.

While our system presents a novel autonomous robot platform, there are many ways to improve the current state by incorporating more sophisticated robotics research into our software. To start, we plan to utilize more of the Velodyne HDL 3D data in order to attempt real-time 3D scene reconstruction and object recognition. Similarly, our system tracks dynamic objects within road lanes, but it does not distinguish

[5] g++-4.1 was updated in Ubuntu Dapper to fix this bug after Player was built on our vehicle's computers.

between vehicles and non-vehicles, nor does it track obstacles outside of the road which may still be on a collision path with our vehicle. Thus, the vehicle cannot yet deal appropriately with pedestrians crossing the road. For more robust navigation, vision needs to be integrated in order to correct MapLanes information due to GPS drift and inaccurate curve estimation. Vision is also needed for dealing with traffic signals, the other main omission from the Urban Challenge scenario (in addition to pedestrians). Finally, we aim to include a model-based planner to provide more human-like local control in open zones and for getting unstuck.

These avenues for future work notwithstanding, we now have an autonomous vehicle that is fully capable of driving in traffic, including the complex multiagent interactions that doing so necessitates. All in all, the research presented in this article takes a large step towards realizing the exciting and imminent possibility of incorporating autonomous vehicles into every day urban traffic.

ACKNOWLEDGMENTS

The authors would like to thank the many members of Austin Robot Technology (ART) for building and supporting the vehicle. Arturo Martin-de-Nicolas started Austin Robot Technology in 2004. His brothers Jorge and Juan have provided invaluable support for both hardware and testing. Don McCauley is the electronics and electrical wizard that gave the vehicle a physical brain. Jon Brogden implemented the microcontroller interface to the vehicle's cruise control motor. Dave Tuttle was the 2007 team leader. Dr. Michael Quinlan provided valuable general robotics expertise in the months leading up to the NQE.

Many (non-author) students helped contribute to the project as well, including Ryan Madigan, Richard Edwards, Srinivas Ashok, David Reaves, Evan Archer, Chris Billen, and many more.

The undergraduate course was made possible through the support of the Freshman Research Initiative, directed by Dr. Sarah Simmons, in the College of Natural Sciences at UT Austin. This research is supported in part by NSF CAREER award IIS-0237699 and ONR YIP award N00014-04-1-0545.

REFERENCES

[1] "DARPA Urban Challenge technical evaluation criteria," 2007. [Online]. Available: http://www.darpa.mil/GRANDCHALLENGE/docs/Technical_Evaluation_Criteria_031607.pdf

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L. E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Winning the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[3] P. Stone, P. Beeson, T. Meriçli, and R. Madigan, "Austin Robot Technology DARPA Urban Challenge technical report," 2007. [Online]. Available: http://www.darpa.mil/grandchallenge/TechPapers/Austin_Robot_Tech.pdf

[4] B. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the International Conference on Advanced Robotics (ICAR)*, 2003.

[5] "Route Network Definition File (RNDF) and Mission Data File (MDF) formats," 2007. [Online]. Available: http://www.darpa.mil/grandchallenge/docs/RNDF_MDF_Formats_031407.pdf

[6] D. Wolf, A. Howard, and G. S. Sukhatme, "Towards geometric 3D mapping of outdoor environments using mobile robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[7] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An introduction to splines for use in computer graphics & geometric modeling*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.

[8] A. Elfes, "Occupancy grids: A probabilistic framework for robot perception and navigation," Ph.D. dissertation, Carnegie Mellon University, 1989.

[9] J. Modayil and B. Kuipers, "Bootstrap learning for object discovery," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[10] P. Beeson, "EVG-Thin software," 2006. [Online]. Available: http://www.cs.utexas.edu/users/qr/software/evg-thin.html

[11] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, March 1984.

Robust Multi-Robot Formations under Human Supervision and Control

Yehuda Elmaliach and Gal A. Kaminka

***Abstract*—There is considerable interest in real-world formation-maintenance tasks, where robots move together while maintaining a geometric shape. This interest is motivated by promise of robustly and efficiently moving multiple robots along a path, guided by a human operator. This paper presents a comprehensive set of techniques that fulfill this promise: (i) a novel method for fusing open- and closed- loop controllers, for robust formation-maintenance; (ii) an ecological display, allowing a human operator to monitor and guide robots, while improving their performance and reducing the failure rate; and (iii) a set of methods for interacting with the formation in the case of a disconnect in the formation. We evaluate each of these contributions in extensive experiments, including 25 human operators. We show significant improvements in performance (in terms of movement time), robustness (both in number of failures, as well as failure rate), and consistency between operators.**

I. Introduction

THERE is considerable commercial interest in formation-maintenance tasks, where a team of robots move together while maintaining a geometric shape. This interest is motivated by promise of robustly and efficiently moving multiple robots along a path chosen for them by a human operator. Examples of such applications include cooperative object carrying [40], scouting [5], platooning and efficient convoys [7], groups of unmanned aerial vehicles [38], and spacecraft formation flying [1]. Realizing the potential of formation maintenance in real-world multi-robot applications requires addressing several open challenges.

First, many real-world applications require a human operator to monitor the state of the robots [14]. Previous approaches to monitoring multiple robots use individual robot displays that are independent of each other. For instance, the operator may monitor all robots in parallel, via a split display showing each robot's individual state; or the operator may switch between such displays [2], [14], [41]. However, independent individual displays lead to difficulties in monitoring *coordinated tasks*, requiring tight, continuous coordination between the robots. The problem is that the operator needs to infer the state of the formation based on individual displays, and (as we show) this decreases significantly from the operator's abilities, reducing the team's performance and robustness.

Second, formation maintenance requires a robot to know of the location of at least one other robot—called a *target*[1]. There are many methods that rely on *closed-loop control* to maintain the formation by tracking the target(s); these rely on the robots' sensors (typically, vision and range sensors) to supply continuous, uninterrupted, feedback about the target's location [5], [6], [9], [10], [13], [15]. However, this feedback is easily interrupted in the presence of obstacles, and when traversing rough terrains. While it can be possible to then re-link the formation [20], persistent interruptions slow the group's progress, and add to the load on the operator.

Finally, if and when an interruption occurs, and the formation becomes disconnected, the human operator must be able to effectively interact with the robots when needed (for example, when the formation becomes disconnected). Robots that require the operator's assistance initiate (or are issued) *call-requests*, which are queued for the operator. Traditionally, the operator switches control between robots, and uses single-robot teleoperation with individual robots to resolve the call requests in some (prioritized) sequence (e.g., [2], [12], [14], [28], [34], [37]). This method works well in settings where the task of each robot is independent of its peers, and thus the resolution of a call request is independent of others. However, in multi-robot formations, the robots are tightly-coordinated, and individual call-request handling means that all robots remain idle, while the operator tends to any single operator. This, despite the additional information that the otherwise idle robots may be able to offer the operator, based on their knowledge of the formation (which allows them to estimate the position of any robot).

This paper presents a comprehensive set of techniques for addressing the challenges above, including: (i) an ecological display, allowing a human operator to monitor and guide robots, while improving their performance and reducing the failure rate; (ii) a set of methods for interacting with the formation in the case of a disconnect, where a robot can no longer proceed; and (iii) a novel method for fusing open- and closed- loop controllers, for robust formation-maintenance that minimizes disconnects.

The paper is organized in three parts:

- First, we discuss tools for single-operator monitoring and guidance of multi-robot formations (Section III). This part explores a novel ecological display allowing a single operator to control multiple robots in formation. We report on extensive experiments with 25 human subjects. The results show that the use of the ecological display (i) reduces the number of failures and task completion time in these tasks; (ii) reduces the number of failures per second; and (iii) reduces the variance in controlling robots, thus leading to more consistent performance across operators.
- The next part (Section IV) explore ways to *multiplex* and/or *fuse* closed-loop and open-loop formation control, to limit the number of disconnects that occur in forma-

The MAVERICK Group, Computer Science Department and the Multidisciplinary Brain Research Center, Bar Ilan University, Israel.
E-mail: {elmaley,galk}@cs.biu.ac.il.

[1]Typically, additional conditions must hold as well.

tions. The idea is to utilize communications to overcome sensor failures, and thus limit the number of interruptions that require human assistance. Experiments show that the techniques are able to significantly reduce the number of undetected obstacles, and thus increase the robustness of the formation to obstacles, even with limited sensing.

- In the final part of the paper (Section V) we report on first steps towards allowing coordinating robots, in a formation task, to use their knowledge of the coordination to autonomously assist the operator in resolving call requests involving a disconnected robot. Experiments with up to 25 human operators show that this call-request resolution method leads to shorter failure-recovery times.

Before presenting these three parts, we discuss relevant relevant literature and motivation for our work (Section II). Section VI concludes.

II. Background and Motivation

This paper touches on a vast body of literature, as multi-robot formations is a well known canonical task, that has been investigated for many years. We therefore focus our review on the most closely related investigations. We first briefly discuss multi-robot formations, specifically in the context of obstacles that interfere with the formation (Section II-A). We then discuss investigations of human operation and monitoring of multiple robots (Section II-B).

A. Robust Formations

Balch and Arkin [5] examine three fundamental techniques for formation maintenance, in experiments with up to four (4) homogeneous robots:

1) *Unit-center-referenced* is a technique where the robots place themselves according to X, Y coordinates, relative to their peers in the formation, and subject to the geometric shape to be maintained. This technique relies on the ability of robots to sense the locations of all others.
2) To address this requirement, the *leader-referenced* technique instead allows robots to position themselves relative to the position of only a single robot, which acts as a leader. However, all robots must orient themselves with respect to the same leader.
3) Finally, the *neighbor-reference* technique relaxes this requirement further. Here, each robot positions and orients itself with respecting to a single robot—called the *target*—but different robots can choose different targets.

It was shown that the last two categories in Balch and Arkin's work (*Leader-Referenced* and *Neighbor-Referenced*) are both related to a general method for formation maintenance, called *Separation-Bearing Control* (SBC) [9], [10], [13], [15]. In SBC, a single robot is chosen as the leader of the formation. Each robot (but the leader) must maintain connectivity—a given distance (separation) and angle (bearing)—with respect to an assigned target. There must be a path of such connected robots from every robot in the team to the leader. It was shown that SBC controllers are sufficient to maintain stable formations.

SBC is arguably the most practical formation-maintenance technique today for real-world settings. This is likely due to its simple requirements of sensing (monitoring) only one other robot, and to the wealth of opportunities it presents for optimizing sensor usage [20], [27] and robot role assignment [24], [26].

There have been several works addressing the robustness of SBC-based formations. Fredslund and Matarić [15] describe an algorithm for generating SBC monitoring rules for robots in a given formation. The robots are assumed to have supporting sensing capabilities, and the position of the leader is given. The monitoring rules are supplemented by communications for robustness against robot death.

Kaminka et al. [20] describe an algorithm that generates SBC monitoring rules based on the sensor configuration of the robots, and dynamically adjusts these rules to overcome sensor failures. They show that this leads to significantly improved robustness, as long as alternatives exist to prevent a robot from becoming completely disconnected. Their approach is susceptible to latencies of the communication protocol used to switch between different monitoring rules.

Our approach relies on fusing SBC control with open-loop, communication-based control of the formation, which relies on the localization of the robots and their ability to accurately estimate their own movements. In this, we complement the techniques outlined above, rather than compete with them.

Most previous work on formation maintenance in the presence of obstacles has assumed that obstacles are detectable in some unspecified fashion. Using the techniques presented below, robots can use their sensors to detect obstacles, to a greater extent than they do when they have to utilize their sensors to maintain the formation. In this, we facilitate the use of techniques which rely the use of obstacle-detection sensors, and that are difficult to use in sensor-impoverished robots that utilize their robots for formation maintenance.

For example, Chen and Li [8] propose a technique where obstacles are recognized by the leader robot, which builds a path for the formation to avoid the obstacles. Thus the leader is responsible for detecting any obstacles. Our approach complements this technique, by allowing other robots to also detect obstacles.

Similarly, Ogren and Leonard [30] describe an approach for allowing a group of robots moving in formation to avoid known obstacles. They show how to calculate a path for each robots that best maintains the formation while avoiding obstacles. Our work is complementary: The multiplexing technique we present is focused on detection of unknown obstacles; but we do not provide a method for calculating obstacle-avoiding paths.

Balch and Hybinette [6] use social potential fields which use attraction and repulsion to position robots within their relative positions in a defined formation. This technique is robust to obstacles in the path of the robots, in the sense that the geometric shape maintained by the formation is dynamically stretched to account for obstacles. However, the techniques assumes that robots know of the positions of obstacles. The technique we present in this paper frees up the robots' sensors for this purpose.

B. Operator Interaction and Monitoring of Formations

The literature on human-robot interaction is vast, but most investigations address a single operator controlling a single robot. Below, we focus on closely related works, which utilize an ecological approach to monitoring the state of robots, as well as methods for interacting (e.g., commanding) the robots. We refer the reader to a comprehensive survey by Goodrich and Schultz [17] for additional related investigations.

Traditionally, human-robot interaction addressed interaction with a single robot. Nevertheless, important lessons can be learned from single-robot displays.

Ricks et al. [31] and Nielsen et al. [29] discuss the ecological approach to displaying information in single-robot navigation tasks. This approach focuses on explicit display of the key constraints of the task [39]. We argue that the *relation tool* we present is indeed such a display for tightly-coordinated tasks, in that it explicitly displays the state of coordination of the robots.

Other ecological displays in single-robot interaction support this approach. Johanson et al. [18] propose a discrete geodesic dome called a Sensory EgoSphere (SES). The SES is a "two dimensional data structure centered on the robot's coordinate frame" that provides the operator with a pointer to an objects on a map and the robot's sensor state. The relation tool similarly centers the display on the lead robot in the formation, and shows all other robots' in relation to it.

Yanko et al. at [41] describe techniques for making human operators aware of pertinent information regarding the robot and its environment. They tested this technique in a rescue-robot competition. Based on their study, they recommend providing user interfaces that support multiple robots in a single display, minimizing the use of multiple windows. The ecological coordination-monitoring display fills these requirements by giving the operator a single view of the controlled robots and their coordination state.

Possibly as a result, the bulk of existing work on controlling multiple robots puts the operator in a centralized role in attending to robots, and does not often distinguish between different task types on the basis of the coordination involved.

Indeed, many existing approaches implicitly assume that the sub-tasks assigned to different robots are independent of each other (e.g., exploring different sub-areas). In such settings, a centralized control scheme does not interfere with task execution, and the monitoring of each robot can be done individually, i.e., by reverting back to a single-operator/single-robot paradigm.

Adams et al. [2] investigated the use of a three-dimensional GUI that has selectable operation modes to switch control between robots, teleoperate a robot, create a navigation plan for the robot, or replay the last few minutes of the robot's task execution (for diagnosis of failures). Our work contrasts sharply with this approach, as we focus on a display that abstracts away the details of the robots' local surroundings, focusing instead on displaying their relative state, not their absolute state with respect to their environment.

Keskinpala et al. at [22], [23] developed a system for controlling a robot from a PDA (Personal Digital Assistant). PDA platforms are small, light-weight and mobile interaction devices, but their size and limited resources pose significant challenges the user-interface design. The system Keskinpala et al. present includes three screens: vision-only, sensor-only and vision with sensory overlay. These methods stem from the requirement to provide the minimally necessary data to the operator needs, because of lack of space in the PDA's display. In our work we suffer from a similar problem, because we monitor multiple robots, and duplicating the displays for each quickly exhausts the available screen resources.

An exception to the single-operator/single-robot paradigm, Skubic et al. [34] reports on an investigation of a sketch-based interface for controlling one or more robots through a known map, on which sketches are drawn grouping robots together for group movement. Waypoints are then plotted and the robots navigate to the waypoints. This display does not keep track of coordinated movement. Instead, it shows where each *individual* robot is located, with respect to a global coordinate system, in contrast to our method, which displays information about the relative position of robots. However, similarly to our approach, all robots are commanded together.

Indeed, monitoring the status of robots and their tasks is only one component of the interaction of a human operator with a team of robots. An additional important component is the ability of the operator to issue commands to the robots, or interact with them when their require assistance. Fields [12] discusses unplanned interactions between a human and multiple robots in battlefield settings, where otherwise-autonomous robots send *call requests* to the human operator to ask for assistance. These call requests are queued, and the operator resolves the problems one by one.

Fong et al. [14] propose a *collaborative control* system that allows robots to individually initiate and engage in dialog with the human operator, one robot at a time. This approach requires significant autonomy by the robots, and assumes that their monitoring need not be continuous. The call requests are queued based on priority, and resolved serially.

Myers and Morely [28] discusses an architecture called TIGER that uses a coordinating agent that mediates between the operator and autonomous software agents. This agent centralizes the information from all agents, and can present it to the operator (or provide it to other agents). The agent is also responsible for translating operators instructions to the team. This approach thus assumes that call requests may be resolved autonomously by the robots, given appropriate high-level commands to the team. In contrast to this approach, we believe (as others do [12], [14]) that often, the operator must directly interact with a failing robot or its teammates to resolve a call request. We thus allow the operator to directly interact with any single robot, while others assist.

Rybski et al. [32] describes an architecture for allowing one operator to control multiple (miniature) robots. The main idea is to increase robot autonomy and allow the operator to interrupt the robot behavior with high-level commands. In addition, they supply an interface that supports mission design, and mission execution, where the operator can view mission status and teleoperate the robots. Our interface is fundamentally different from this interface in that we show a state view of all the robots, rather than only an individual

robot's.

ACTRESS (Actor-based Robots and Equipment Synthesis System) [4], [35], [42] is an architecture including an interface for monitoring and controlling multiple robots. The operator may issue commands that affect groups or individual robots; information is presented to the operator based on both explicit requests (from the operator to individual robots), as well as based on gathering of information exchanged by the robots. However, ACTRESS does not focus on visual presentation of the coordination, in contrast to our work. Moreover, ACTRESS does not utilize collaboration between the operator and robots in resolving call requests. The operator may issue commands to robots that assist in such resolution, but the robots are otherwise idle.

In contrast to the above centralized approaches, we believe that in tight-coordination settings, resolving call-requests is in the interest of all robots currently coordinating with the robot requiring assistance—and thus they should actively collaborate with the operator to resolve the call request. Other work has also examined distributed paradigms for human/robot interaction.

Tews et al. [37] describe a scalable client/server architecture that allows multiple robots and humans to queue call requests (*service requests*) for one another. Scerri et al. [33] describe an architecture facilitating teamwork of humans, agents and robots, by providing each member of the team with a proxy and have the proxies act together as a team. Our work differs from both of these investigations in that we do not attempt to put humans and robots on equal ground. Instead, the human initiates an controls the call-request resolution. However, once initiated, the task is carried out by all members of the robotic team and the operator.

Ali [3] compares different classes of human-robot team interaction (*Direct manual control, supervisor control, individual* and *group control*). The parameters measured are effectiveness (in term of task completion and speed of completion), safety (both for the robots and their environment), and ease of use. While we similarly evaluate different interaction methods, we focus only on the case of one operator and multiple robots. However, within those, we distinguish several different types. Moreover, we provide new distributed resolution types.

III. A Socially-Attentive Ecological Display

We first introduce the ecological display approach for human operator monitoring of formations, which focuses on explicitly displaying the state of coordination the team (Section III-A). We then describe the display in detail (Section III-B). We empirically evaluated this approach in extensive systematic experiments with up to 25 human operators. The results show (Section III-C) that the use of the ecological display (i) reduces the number of failures and task completion time in these tasks; (ii) reduces the number of failures per second; and (iii) reduces the variance in controlling robots, thus leading to more consistent performance across operators. To our best knowledge, this is one of the largest studies done with human operators controlling multiple robots.

A. Monitoring the State of Coordination

A key component in real-world applications of multi-robot formation maintenance tasks is allowing the operator to monitor the progress of the team, and the status of the robots. Previous approaches to monitoring multiple robots use individual robot displays that are independent of each other, as discussed above. For instance, the operator may monitor all robots in parallel, via a split display showing each robot's individual state; or the operator may switch between such displays [2], [14], [41].

However, independent individual displays lead to difficulties in monitoring *coordinated tasks*, requiring tight, continuous coordination between the robots; i.e., where robots are highly inter-dependent. Here, the operator must monitor the state of coordination—the relative state of robots—in addition to the state of each robot. Such monitoring is called *socially-attentive* because it focuses on inter-agent relations [21].

Formation-maintenance is an example of such a task, requiring tight continuous coordination between robots. Such a task can be executed by a single operator, by guiding or teleoperating the lead robot, and allowing the others to maintain the formation autonomously. To maintain the formation, the operator must monitor the formation itself—slowing down or speeding up the lead as necessary—in addition to monitoring the movement of the team towards its goal. Such monitoring can be done, in principle, by showing the camera view of each robot. However, it might be much easier to do if the operator has a bird's eye view of the formation, showing the *relative* positions of robots. Unfortunately, a bird's eye view is not always possible, e.g., for lack of a global-view camera.

To address this challenge, we develop a socially-attentive *ecological display* component—called *relation tool*—that explicitly displays the state of coordination in a team, complementing individual display. Ecological interface design emphasizes visual cues that focus on the key constraints in the user's task [39]. For coordinated tasks, these include the coordination constraints in the team [21]. The relation tool allows the operator to visualize the robots' state with respect to each other, and thus visually identify coordination failures. Since the relative state of robots may not be known directly, the relation tool fuses sensor readings from multiple robots, and reconstructs from these the state of coordination between them. In doing this, it must overcome the uncertainty and noise inherent in robot sensor data.

The graphical *socially-attentive* display complements existing displays. It allows the operator to visualize the robots' coordination—their state with respect to each other—and thus visually identify coordination failures before they become catastrophic. By showing the operator an explicit visualization of the coordination state of the team, her cognitive load would be reduced, and her performance would increase.

B. The Relation Tool

Ecological interface design emphasizes explicit visualization of key constraints in the user's task [39]. Socially-attentive monitoring emphasizes that in coordinated tasks, these include the relative state of robots [21]. To show these constraints,

we developed the *relation tool*, a 2D display that shows the relative state of robots by drawing a geometric shape corresponding to their state. Colored dots denote different robots. The positions of the dots denote their states, and thus the shape they make up—their relative positioning—denotes their relative states. In principle, every application requires its own method of projecting robot state onto a 2D plane, and a target shape that defines normative coordination.

The key is that the operator should be able to see, at a glance, whether the shape being maintained corresponds to correct coordinated execution. When the shape deviates from ideal, the operator can easily identify coordination faults within the monitored team, with little or no need for inferring this information from the other displays. This eases the cognitive load on the operator in coordinated tasks.

We investigate the use of the relation tool concretely in two popular formation maintenance tasks (triangle and line). We created human-controlled versions of these tasks, and implemented them using the Tekkotsu software [36] for Sony AIBO robots. Each robot has an on-board video camera and a infra-red distance sensor pointing at the direction of the camera. They transmit their video and sensor readings to the operator's station for monitoring. The operator uses the mouse as a joystick, moving the controlled robot in the direction and speed chosen.

We begin by examining the line formation task, which we refer to here as *cooperative pushing*, as it has two AIBO robots jointly push a light-weight bar across the floor (Figure 1). One robot is teleoperated, while the other pushes the bar while maintaining a straight line with the human-controlled robot. The bar is color-marked, such that a robot can identify its position with respect to the bar. If the mark moves too much to the side, this would indicate a drift, i.e., the robot is either lagging behind or is pushing too quickly ahead. Here, we follow traditional sensor-based formation-maintenance techniques; the robots do not communicate with each other. Section IV examines the use of communications to maintain formations.

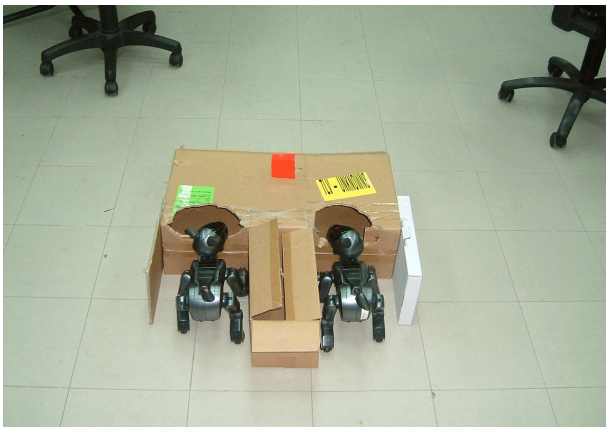
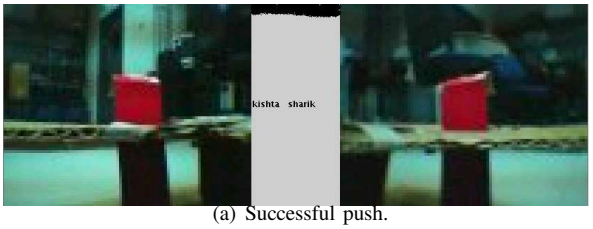


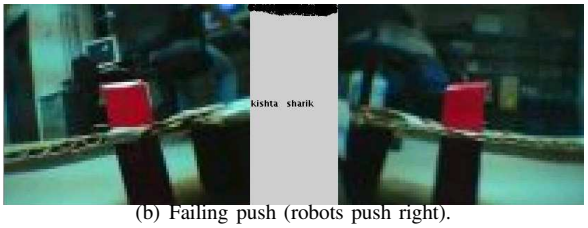
Fig. 1. Cooperative pushing (line formation) by AIBO robots.

The coordination between the robots involves a single dimension—the robots are to maintain equal velocities. One possible visualization of this relationship consists of a horizontal line that connects two dots, representing the robot. The horizontal position of the dots remains fixed, while the Y axis denotes the angle of the color mark within their view.

Figures 2 and 3 show the interfaces when executing this task. Figure 2 shows the split-camera view from the individual robots, as presented to the operator, in a successful case (Figure 2-a), and in a failure case, where the box drifts to one side (Figure 2-b). Figure 3 shows the respective relation tool displays in both cases: The successful push (Figure 3-a) and the failing push (Figure 3-b). As can be seen, it can be difficult to differentiate the split-view displays in cases of success and failure (Fig. 2). However, by showing the relative velocities of the two robots (Figure 3) the failing push is easily detected.

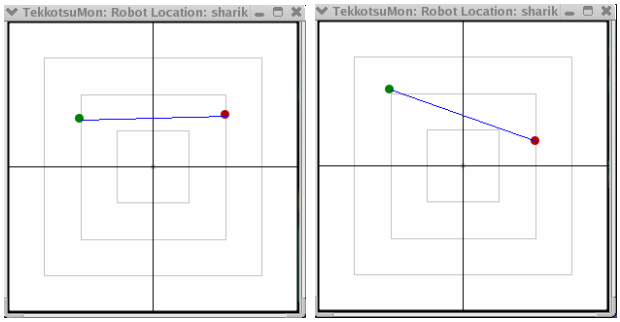


(a) Successful push.



(b) Failing push (robots push right).

Fig. 2. Cooperative pushing (line formation): Split camera view.



(a) Successful push.

(b) Failing push (robots push right).

Fig. 3. Cooperative pushing (line formation): Relation tool display.

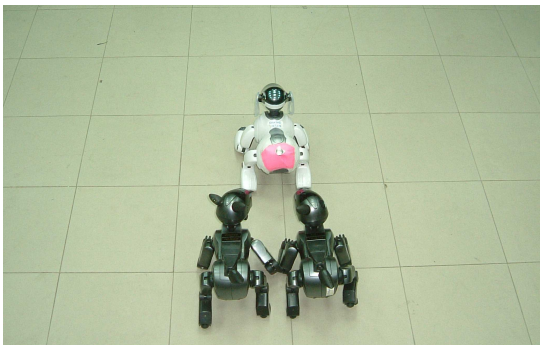
Of course, providing a visualization of the relative states of robots is trivially done when a global world-view camera exists, or perfect global localization data is available. However, this is not often the case in real-world applications.

Thus a key challenge in developing the relation tool lies in integrating the information needed for the visualization, from the robots themselves. The approach we take analyzes the robots' own sensor readings (including camera positioning,

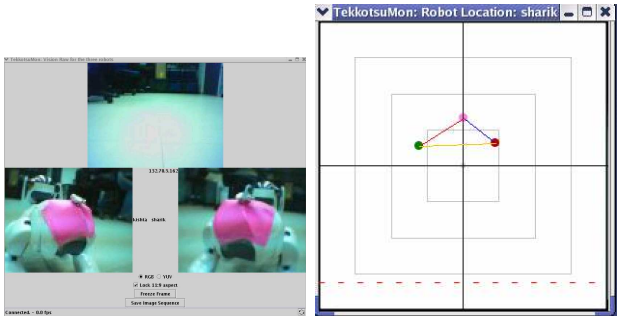
infra-red range sensor readings, detected objects) to reconstruct the position of the robot with respect to others, from its own perspective. As a side-effect, we expose the relation tool display to the uncertainty and noise inherent in robot perception. This must be countered by noise-filtering processes within the display. In our case, a moving average filter was used on the distance and angle data to create a stable display.

The relation tool may be used to draw the attention of the operator to specific robots that are responsible for any mis-coordination. We use the formation task to demonstrate. Here, the objective is to navigate a triangular formation (three robots), through a short obstacle course. To allow a human operator to control the formation, the lead (front) robot teleoperated by the operator, while the two follower robots maintain fixed angles and distances to this robot using their sensors. Again, the robots do not utilize any communications for maintaining the formation.

Figures 5 and 4 show this task in action. Figure 4 shows an example of perfect formation, while Figure 5 shows a failed formation situation. In both figures, sub-figure (a) shows the actual position of the robots on the ground; (b) shows the split-camera view from each of the individual robots; and (c) shows a screen snapshot of the relation tool.



(a) Ground truth.



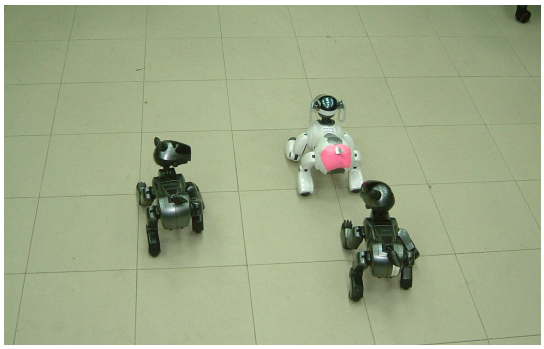
(b) Split-camera view. (c) Relation tool.

Fig. 4. Successful Triangle Formation.

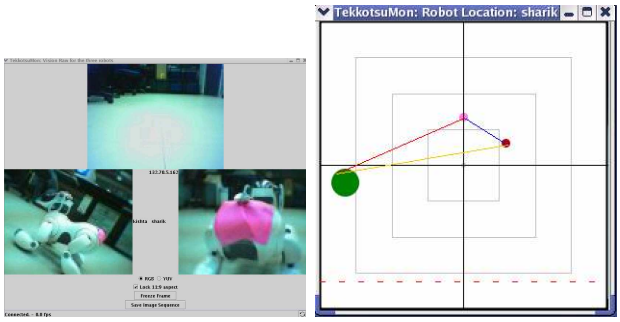
The figures contrast the information presented to the operator with the relation tool and using existing approaches. Unlike the cooperative pushing task, the split-camera view (sub-figure (b) in Figures 5 and 4) does indeed provide indication of whether the formation is maintained. However, it is difficult to see from the split camera view to what degree the formation is maintained (i.e., the magnitude of the failure), and which robots are responsible for it (i.e., the location of the failure).

In contrast, the relation tool makes it easy, at a glance, to see not only whether the formation is maintained, but also the magnitude and location of any failures. We chose polar coordinates to describe the formation. The X axis denotes the angle to the leader, while the Y axis denotes the distance to the leader. The position of the leader is always fixed. We connected the points (that represent the robots) with lines to create a shape easily recognizable by the operator.

The choice of the polar coordinates separates distance and angle for the operator. By glancing at the shape, one can fairly quickly determine whether the formation is breaking because a robot is lagging behind (distance too great), or its angle with respect to the leader is too sharp (e.g., because of a sharp turn).



(a) Ground truth.



(b) Split-camera view. (c) Relation tool view.

Fig. 5. Failing Triangle Formation.

Indeed, to further assist the operator in localizing coordination problems, the display uses additional mechanisms to draw the attention of the operator where its most needed. One such fault-feedback mechanism uses the size of the dots, representing robot positions, to draw the operator's attention to failing robots. We use three sizes: regular, medium, and large. Regular size is used when the associated robot lies fulfills the constraints of the formation. Medium size is used when the robot begins to report intermittent failures in following the lead, as these are indicative of an impending formation failure. The large size is used when the formation is essentially broken, e.g., when the robot in question completely lost track of the lead robot, and is unable to proceed.

Another fault-feedback mechanism is the dashed line drawn across the bottom of the display. This line signifies the maximum distance sensed by the robots' sensors, and thus the

position in which they are likely to lose track of the leader. The operator may use this line to estimate how far it can let the robots stray away from the leader, while not getting into catastrophic failures.

C. Experimental Evaluation of the Relation Tool

We evaluate the effectiveness of the relation tool in the triangle formation maintenance and the cooperative pushing (line formation) tasks. Our goal is to explore the generality of the method. In the triangle formation, the operator leads the robots in a triangular formation towards a target destination, while avoiding obstacles. If the operator causes the lead robot to turn too sharply, or move too quickly, the formation may break, as the SBC controller in the follower robots will lose sight of the leader. However, the operator seeks to minimize the time it takes to reach the destination. In the line formation (box pushing), the operator controls the velocity of one of the robots, while the other is pushing autonomously. The operator must be careful not to push too quickly for the other robot, nor to lag behind.

We believe that the relation tool should be used to complement, rather than replace, existing display (which focus on individual robot state). We thus conducted experiments contrasting different combinations (see below) of the socially-attentive display with individual robot display. We ran multiple experiments with novice operators, age 20–30.

19 operators were tested in the pushing task (18 males/one female, 18 students—including the female—of which 15 are in computer science). 25 operators were tested in the formation task (23 male/two female, 22 students–including the two females—of which 19 are in computer science). The students in both groups were either graduate students or undergraduates in their final year. None had previous experience controlling multiple robots of any kind.

Each operator tried all combinations available in the task she operated (a within-subjects design). However, to avoid ordering effects, the order in which each operator tried each combination was randomized (in both sets of experiments). In no setting were the operators able to see the robots while operating them. In all cases, operators were given an approximate 25-minute training session in operating a single and multiple robots (including the formation and pushing tasks), until they reported they felt comfortable controlling the robots. Overall, the results below represent almost 100 hours of human operation.

1) Cooperative Pushing (Line Formation) Experiments: The first experiment examined the use of the relation tool in the cooperative pushing task. We contrasted three interfaces: a split-camera view only (representing existing approaches), a combination split-camera and relation tool, and the relation-tool alone. We remind the reader that all 19 human operators were tested on all three interfaces, randomizing the order of their introduction to the different interfaces to prevent biasing the results due to human learning. Their performance was measured as the average absolute angle deviation from the imaginary horizontal line connecting the robots when they maintain ideal relative velocity. This angle was sampled at 20Hz during task execution.

Figure 6 shows the results of this experiment—the average absolute angle error—averaged across all operators. Clearly, both combinations that use the relation tool are significantly superior to the interface relying on camera alone. Moreover, the surprising result here is that the relation tool by itself is sufficient (in fact, even slightly better than its combination with the split view). This is due to this task being essentially a pure-coordination task: The operator does not need to worry about where the pushed object is going, as long as the relative velocity of the robots is 0 (i.e., their velocities are equal). Thus even a socially-attentive display by itself is sufficient. On the other hand, the non-social split-camera view (by itself), is difficult to use for coordination. A one-tailed t-test (assuming unequal variance) shows that the difference between using the tool by itself, and using the split-camera view, is statistically significant (we use a 0.05 significance level). The probability of the null hypothesis is $p < 0.014$ when looking at the difference in the number of failures.

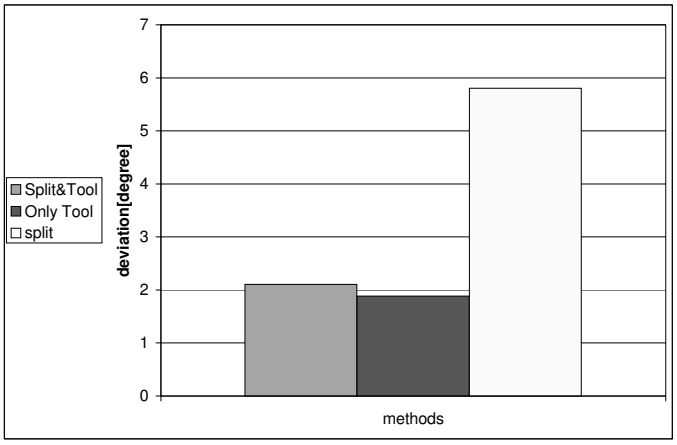


Fig. 6. Line formation: Total number of failures.

We also examine the angle error results with versus task completion time. Figure 7 shows these results as average absolute angle error versus time to complete a 180cm walk. The Y axis is the average angle error and the X axis is the time to complete the task. The fastest time for completing the 180cm walk was by the method that does not use our socially-attentive display (the *split view*). However, we can see that this method also had the maximum average angle error. We believe that this is the result of the operator, having no idea of the relative state of the two robots, just pushed the teleoperated robot as quickly as possible, finishing the course as quickly as possible (but poorly).

These results should only be interpreted qualitatively. The exact distance traveled by robots in each trial is technically difficult to measure precisely (as the robots' own odometry is inherently inaccurate). We thus allowed the operator to always proceed for the estimated distance (180cm), and measured the time it took. Because of the distance is estimated in this trial, the timing may not be accurate.

Finally, we examine the the standard deviation of the distribution of number of failures in these experiments. Lower standard deviation indicates more consistent performance of the different subjects, i.e., less differences between the results

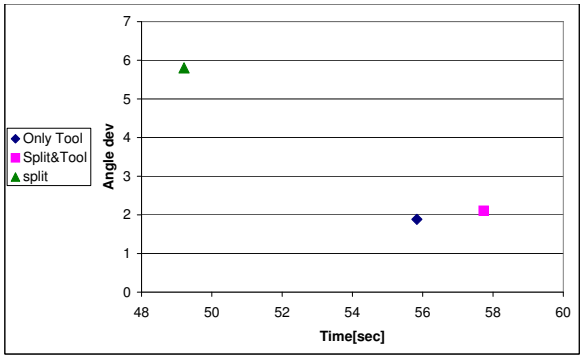


Fig. 7. Line formation: Failures vs. task completion time.

Interface	Std. Deviation
Split-view	7.11
Split+Tool	**0.93**
Tool alone	**0.79**

TABLE I
LINE FORMATION: STANDARD DEVIATION OF THE NUMBER OF FAILURES.

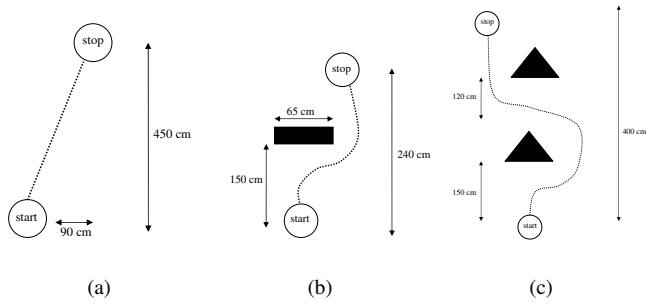


Fig. 8. Triangle formation obstacle courses.

of different operators. This, in turn, typically indicates more intuitive and more generally-applicable interfaces. The standard deviation results appear in Table I. They indicate that interfaces using the relation tool unequivocally lead to more consistent performance among operators.

2) Coordinated Movement Evaluation: In the triangle formation task, we compare three interfaces. The first presented the operator with the split-view video streams from all robots (e.g., Figure 4-c). The second combined the this split-view with the socially-attentive display previously described. The final interface consisted of a single camera (the lead robot's) and the socially-attentive display. Each of the interfaces was tried with three different obstacle courses, varying in difficulty (a total of 9 different configurations). The *simple* course consisted of an open space with no obstacles at all (Figure 8-a). The *medium* course consisted of a single obstacle that had to be by-passed (Figure 8-b). In the *difficult* course, the operator was to lead the robots between the two obstacles (Figure 8-c). To verify the relative difficulty of the path, we sampled 7 of the experiments for the number of times a robot hit an obstacle: The simple course had no such hits (as there are no obstacles). The medium course had only a single hit in all experiments. The difficult course had 2–3 hits per method.

Again, all 25 operators tried all nine different settings, in randomized order (to prevent learning effects). For each of the trials, we recorded the number of non-catastrophic formation failures, and time to complete the task. Non-catastrophic formation failures were measured as the number of times a follower robot has temporarily lost track of the lead. These are indicative of the quality of the operator's control. Too many of them result in permanent tracking failures, which lead to total breakdown of the formation. When such failures occurred, the operator would have to teleoperate the straying robot until the formation was re-established.

Figures 9,10,11 show the results of these experiments in terms of the average number of non-catastrophic failures per operator, versus the average task completion time. The horizontal axis shows the time (in milliseconds). The range of the horizontal axis in these figures is fixed at 12 seconds, though the offset is different, as the more difficult courses took longer. The vertical axis shows the average number of non-catastrophic failures that took place during each trial.

The results show that in all course difficulty settings—simple, medium and difficult—the use of the relation tool is preferable to using only individual displays. This lends support to the hypothesis that socially-attentive ecological displays (explicitly displaying coordination state) can significantly improve monitoring of robots in coordinated tasks.

In particular, both course completion time and the number of failures during execution were generally reduced using the socially-attentive display. In the simple- and medium-difficulty courses, the best monitoring approach was single camera and the socially-attentive display. It was significantly better than the split camera interface, at a 0.05 significance level. In the easy course, a one-tailed t-test (assuming unequal variances—see below) shows a significant difference these method, both in the number of failures (the probability of the null hypothesis being $p < 0.011$), and in the time ($p < 0.015$). Similarly, in the medium course, there are significant differences between these two methods, both in the number of failures ($p < 0.04$) and in task completion time ($p < 0.02$).

However, in the difficult course the best monitoring approach used both the split-view and the relation tool, in spite of the additional information displayed to the operators. The difference between this approach and the split view interface was not significant in time ($p = 0.48$), but was significantly different in the number of failures ($p < 0.014$). The difference in the number of failures between the split view interface and the interface using single camera and relation tool was only moderate ($p \approx 0.15$). We believe that this is due to the operator using the split-camera view to look at obstacles that have been bypassed by the lead (see [29], [31] for an ecological interface approach to this problem). Such obstacles were not much of a problem in the other, easier, courses. We leave further investigation of this to future work.

While the results show significant improvements in task completion time and number of failures, a question may be raised as to whether a socially-attentive ecological display qualitatively changes the way the operator interacts with the team. For instance, the experiment results above could also be

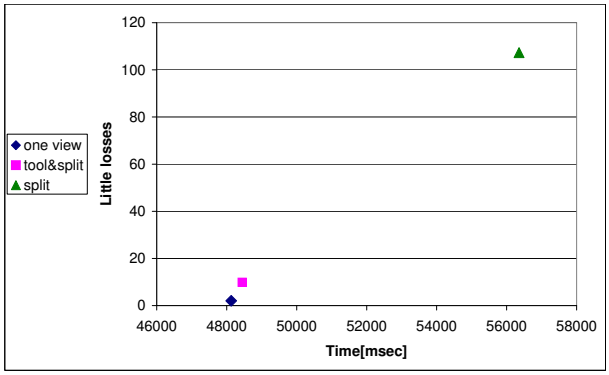


Fig. 9. Triangle formation failures in *simple* course.

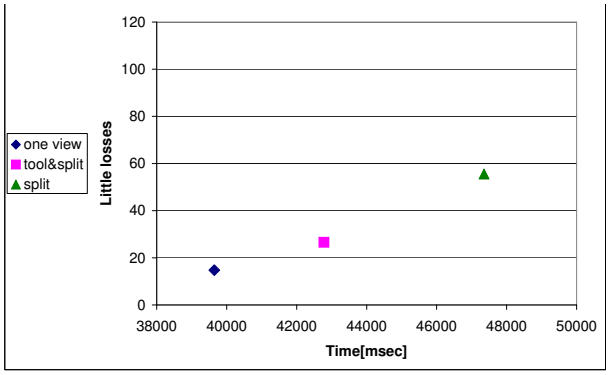


Fig. 10. Triangle formation failures in *medium* course.

indicative of the team going slower or faster, but maintaining the same number of failures per second—thus indicating that the drop in failures is due to the team moving faster, rather than to a qualitative change in operator control.

Additional results show that rather, the use of the relation tool leads to qualitative differences in the the way the operator controls the robot team. Figure 12 shows the average number of failures per second, in the different courses. Clearly, the easy course is indeed easier than the medium-difficulty course, which is easier than the difficult course. However, what we see in the results is that the use of the socially-attentive display leads to a significant reduction not just in the time and total

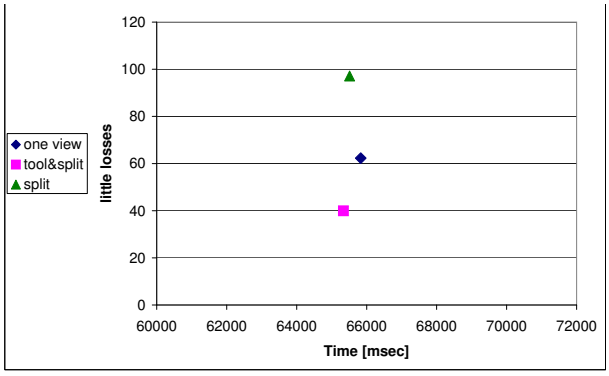


Fig. 11. Triangle formation failures in *difficult* course.

number of failures (as evident from the previous figures), but also reduces the *failure rate*.

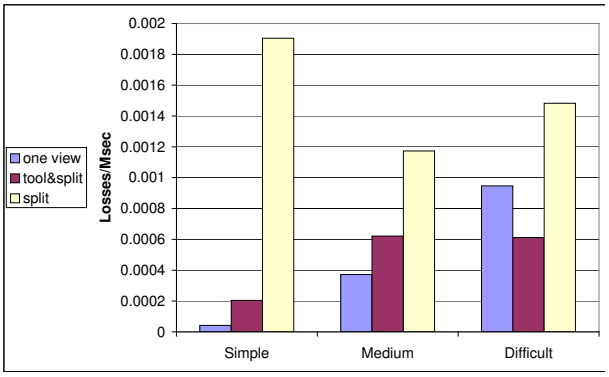


Fig. 12. Triangle formation failures per millisecond.

Additional evidence for this qualitative improvement in operator control is found when we examine the standard deviation values for the number of failures and task completion times of different operators. Table II displays the standard deviation of the number of failures, for the different courses. Table III displays the standard deviation values of the task completion time, for the different courses.

These results show that operators are more consistent in their performance when using the relation-tool, than when using the split view by itself. The standard deviation values for the methods using the relation tool are generally much smaller than for the split camera display. This indicates more consistent values, i.e., less variance between operators in terms of ability to control the robots. In the difficult path, the single camera view with the relation tool has a large standard deviation (though smaller than the one for the split camera view by itself, when looking at the number of failures), but the relation tool with the split camera view has smaller standard deviation.

Course	Split View	Split View and Relation Tool	Single View and Relation Tool
Easy	319.45	32.65	**6.59**
Medium	141.85	51.30	**50.56**
Difficult	144.97	**66.93**	138.65

TABLE II
TRIANGLE FORMATION: STANDARD DEVIATION IN NUMBER OF FAILURES.

Course	Split View	Split View and Relation Tool *(tool+split)*	Single View and Relation Tool *(one view)*
Easy	25.53	12.01	**9.17**
Medium	22.37	14.68	**10.68**
Difficult	15.88	**13.58**	19.22

TABLE III
TRIANGLE FORMATION: STANDARD DEVIATION IN TASK COMPLETION TIMES (MEASURED IN SECONDS).

D. Monitoring Formations: Summary

This section took a step towards allowing a single human operator to effectively monitor a team of robots that are tightly coordinated. The experiments we have conducted show that existing techniques do not adequately address this challenge. Their inability to explicitly display the coordination state of the team cognitively burdens the operator and reduces from her effectiveness at controlling the robots.

The socially-attentive *relation tool* display is an ecological interface display addressing this challenge. It has three principal advantages over previous work:

- First, it significantly reduces the amount of inference needed by the operator to infer the relative state of robots—and thus the state of coordination between them.
- Second, its dimensions can be used to directly provide the operator with information about failures, e.g., as in the formation case.
- Third, it can easily complement other types of displays useful for the task, such those that show the heading or distance left to the destination, power, etc.

The experiments on real robots show that the relation tool significantly reduces the total number of failures, and task completion time in two tight-coordination tasks. Furthermore, we have shown that the use of the relation tool leads to qualitative change in the capabilities of the operator: Not only do failures and completion time decrease, but the failure rate (failures per second) improves significantly as well. In addition, methods utilizing the relation tool lead to more consistent operator performance.

IV. Maintaining Robust Formations

The operator of a formation is inherently limited by the capabilities of the robots to sense their surroundings, and provide information about potential failures. A challenge arises when robots do not have sufficient sensors to both track their peers and their environment at the same time. This could be, for example, if the limited sensors are kept busy providing input to the closed-loop controller that is used to maintain the formation. This section addresses this challenge.

We first introduce differentiate sensor-based closed-loop formation maintenance from communication-based open-loop formation maintenance (Section IV-A). We then (Section IV-B) discuss the two key methods used to combine open- and closed-loop maintenance (*multiplexing* and *fusing* controllers). Finally, we report on experiments evaluating the different methods (Section IV-C).

A. Open-Loop and Closed-Loop Formation Maintenance

In the *sensor-based* formation-maintenance algorithm, each follower but the leader is to maintain a specific distance and angle to another robot (called the *anchor*). This is called Separation-Bearing formation-maintenance control, and is proven to be stable [13]. A problem arises when the robot's sensors are limited and the robot also needs to detect obstacles: If the follower does not scan for obstacles, it may fail to discover them. And if it scans for obstacles, it may lose sight of its anchor, and thus lose its place in the formation.

For instance, on the Sony AIBO ERS robots, the sensors used for formation maintenance are on a single pan-tilt component (the head). The robot cannot follow a leader (at some fixed angle) and simultaneously scan for obstacles.

One obvious alternative is to utilize open-loop control in formation maintenance, to free up robots' sensors for other uses such as obstacle avoidance. While operating in *communication-based* open-loop control, the leader of the formation broadcasts its movement vector (velocity and heading changes). Based on this communication, and their predefined ideal positions in the formation, all other robots calculate their own relative movements, without relying on sensors. However, this relies on odometry reading in both the leader and the follower. In principle, translating the movements of the leader into each follower's actions, via communications, is sufficient. In practice, accumulating odometry errors prohibit this technique from being used exclusively.

This is indeed an open-loop controller for the formation: Messages cannot in practice be sent continuously, and thus a projection is made as to the anticipated position of the leader (and by implication, the follower), using *affine transformations* [43]. Once the anticipated position is known, the follower can set it as a goal position, and use simple motion planning to generate a movement vector of its own. This movement vector is maintained until a new broadcast from the leader initiates this calculation once again.

The translation of target position to movement vector has two factors. The first is handled by the affine transformation. The second requires additional corrective actions by the follower robots. We describe these factors below.

The first factor is the effect of the leader's heading on the path chosen by its follower. Figure 13-b,c show cases where the position of the leader is identical, but its heading is different. As a result, the target location for the follower, and the path to it (both indicated by the arrow in the figures) are radically different. This also implies that the affine transformations are sensitive to errors in their inputs, as even small deviations in the heading may result in large difference in the computed movement vectors.

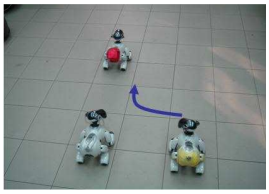
The second factor in correctly computing the movement vector is tied to the difference in the body orientations of the leader and follower robots, after the latter reach their target positions. Ideally, the orientation of the leader and followers should be equal at that point. However, depending on the path taken by the follower, the orientation of its body might be different from that of the leader (see Figure 14).

To maintain the orientation error in the followers as small as possible, we recommend explicitly tracking the difference in orientation between the leader and the follower, and correcting it in each time step. However, this approach can result in jerky movement on the part of the robots, when they attempt to correct a large error within a single time-step. To address this, the controller should limit itself to corrections that are only of a limited range, and instead apply them over multiple time-steps, if necessary.

The advantage of the communication-based controller is that it can free up some of the robot's sensors. Instead, the follower robot maintains the formation only by communica-

(a) Ideal positions of the robots.

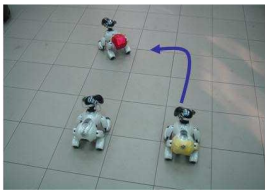
(b) Leader changed heading.



(c) Leader did not change heading.

Fig. 13. A triangle formation of three Sony AIBO robots. Figure (a) shows the ideal poses of the robots. Figures (b) and (c) illustrate the sensitivity to heading; the leader is in the same x, y location in both figures, but its heading is different, implying a radically different target position for the right follower robot.

(a) Follower orientation maintained at end of path.



(b) Follower orientation not maintained at end of path.

Fig. 14. In (a) and (b) the x, y location of the follower is the same, as the target position. However, the path taken by the right follower to the target greatly affects the final orientation of its body with respect to that of the leader.

tion. The disadvantage of this technique is that it requires perfect odometry, a requirement that cannot be fulfilled in realistic settings. If the anticipated position of the leader and the follower are computed based on imperfect, noisy odometry, the errors quickly accumulate. Moreover, as we have seen, slight differences in values of the heading can imply radically different movement vectors.

B. Combining Controllers

To allow limited-sensor robots to maintain formation while still recognizing obstacles, we propose to combine the two controllers described above, in settings where the robots' sensors are limited, but communication between robots is possible. In such settings we propose to combine two formation controllers types: A closed-loop formation-maintenance algorithm using sensors, and an open-loop algorithm using internal navigation (odometry) and communications.

We compare between two combination approaches: *multiplexing* the controllers (using one at a time), and *fusing* them (using both in parallel). The idea in combining the controllers is to offset their disadvantages, and gain from their complementary advantages.

The *multiplexing* technique works as follows. Each follower robot relies on the sensor-based algorithm until it arrives to its predefining position in the formation. We therefore explore ways to *multiplex* and/or *fuse* closed-loop and open-loop formation control. Multiplexing is done in time, giving the alternative methods different periods of time in which they control each robot. Switching between the different methods utilized the following principle: Each robot relies on visual tracking (closed-loop control) until it is within tolerance levels of its position in the formation. When this occurs, the robot switches to *communication-based* open-loop control, and uses its sensors to scan for obstacles, while communicating with the leader. To verify its position and inhibit accumulating errors, the robot switches back to visual tracking after a fixed period of time. We also explore combining controllers by fusion, by merging the output commands of each open-loop and close-loop controllers.on (within some tolerance radius, to allow for uncertainty in sensing). When this occurs, the robot switches to the communication-based formation-maintenance behavior. Now, the robot's sensors are free and the robot can search for obstacles. The follower robot moves in this mode for a fixed period of time (which we vary in the experiments, see Section IV-C). It then switches back to the sensor-based algorithm, and the cycle repeats.

In the *fusing* technique, the robots multiplex between the open- and closed- loop controllers (otherwise, they cannot hope to detect obstacles). However, during the time when both sensor-based and communication-based controllers are active, the output commands of the controllers are fused: The average of the two controllers is taken as the output.

There are competing goals in using the open-loop controller, with both combination techniques. On one hand, the more the robots rely on open-loop control, the more they can scan for obstacles, and provide improved performance. On the other hand, the longer they remain in open-loop control, the more errors in position are accumulated (in relative positions of the robots, with respect to their teammates), and thus the formation degrades.

Thus the timeout period, which limits the amount of time robots remain under communication-based open-loop control must be determined. We take an empirical approach to determining this value. We note that it might be possible to set theoretical bounds on this value, depending on expected obstacle density. We leave this direction of research to future work.

C. Combined-Control Experiments

To evaluate the contribution of these approaches, we compare the *multiplexing* and *fusing* methods with their closed-loop and open-loop components, by themselves. The experiments are carried out using physical (Sony AIBO) and simulated robots.

We carried out two separate repeated-trials experiments. The evaluation has two facets. First, in Section IV-C1, we evaluate the impact of the combination techniques on the ability of

sensor-limited AIBO robots to detect obstacles, the motivation for the techniques. Second, in Section IV-C2, we evaluate the hypothesized costs of combination, i.e., the hypothesized decrease in precision.

1) Detecting Obstacles: This section report on experiments carried out with physical Sony AIBO robots moving in formation. The goal of the experiment is to evaluate to what degree does controller combinations (e.g., multiplexing) allow robots to detect obstacles that may be otherwise be undetectable.

Here, three Sony AIBO ERS-7 robots were arranged in a triangular formation (Fig. 13-a). While operating in sensor-based separation-bearing control mode, the two followers in the rear monitor the leader using their head-mounted camera and infra-red range sensors. The robots utilize the color patch on the rear of the leader for identification, and maintain the distance and angle to it [20]. Otherwise (when using communications) they scan for obstacles and maintain the formation by communication.

The leader actively scans for obstacles. On detection, it finds a path around them that considers its own physical body, rather than the entire team (as proposed in [8]). Such a path cannot be considered safe for the followers, and indeed we intentionally place obstacles such that such a path would put them in the way of the followers. This is done so as to examine the followers' ability to detect obstacles.

We use three different obstacle courses for this experiment (see Fig. 15). In the *Right* obstacle course the robots walk in a straight line; the right follower robot needs to recognize the obstacle blocking its path. The *Left* obstacle course poses the same challenge to the left follower. Finally, in the *Diagonal* course the right follower needs to recognize the right obstacle and the left follower needs to recognize the left obstacle (the leader will try to pass between the obstacles).

In each of the obstacle courses, the formation was run five times, in both the visual sensing control mode, and the *multiplexing* mode, for a total of 30 runs (10 in each course). We did not experiment with the open-loop control in these experiments; as it essentially frees up all the robots sensors to focus only on the task of detecting obstacles, it serves as a theoretical upper limit. We therefore assumed that with pure communication-based control, all obstacles are detected. We note also that there are no separate results for the fusion method, because it is identical to the multiplexing method in terms of time available for detecting obstacles (since then only one controller is generating output). The distinction between them is explored in Section IV-C2.

Fig. 16 shows the result of the comparison between the multiplexing technique and the sensor-based formation maintenance. The X axis shows the obstacle course. The Y axis shows the fraction of the undetected obstacles over all trials, thus a lower value indicates improved performance. We can see that the multiplexing technique performs better than the sensor-based algorithm used earlier, though statistical testing shows that the difference is only moderately significant (one-tailed t-test, $p = 0.07$).

A one-tailed t-test significance test of the experiments with the robots (above) showed that multiplexing was only moderately significantly better ($p = 0.07$). We believe this is due to the relatively small number of experiments. We thus ran additional experiments with *simulated* AIBO robots, using the player/stage environment [16], where many more trials could be run. Figure 17 describes the obstacle course used in the simulated environment. Each of the techniques (multiplexing, sensor-based) was run 25 times.

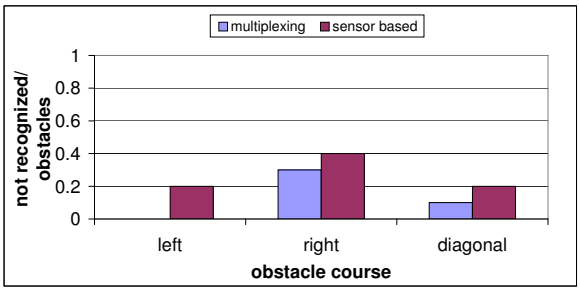


Fig. 16. Fraction of undetected obstacles over multiple runs of each technique, in different obstacle courses.

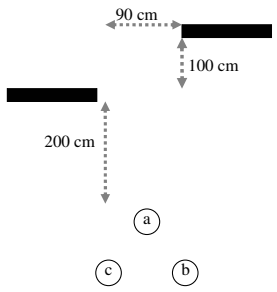


Fig. 17. Obstacle course in the simulation experiments. The leader robot moves in straight line, but its followers must detect the obstacles on its left and right.

Figure 18 shows the fraction of unrecognized obstacles (Y axis) in this experiment, for each of the techniques, over 25 runs. The Y axis shows the fraction. We again see that the multiplexing approach significantly decreases the fraction of undetected obstacles. The results are significant at a level of $p = 0.00000000164$ (one-tailed t-test).

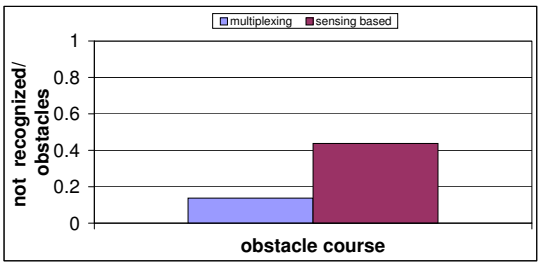
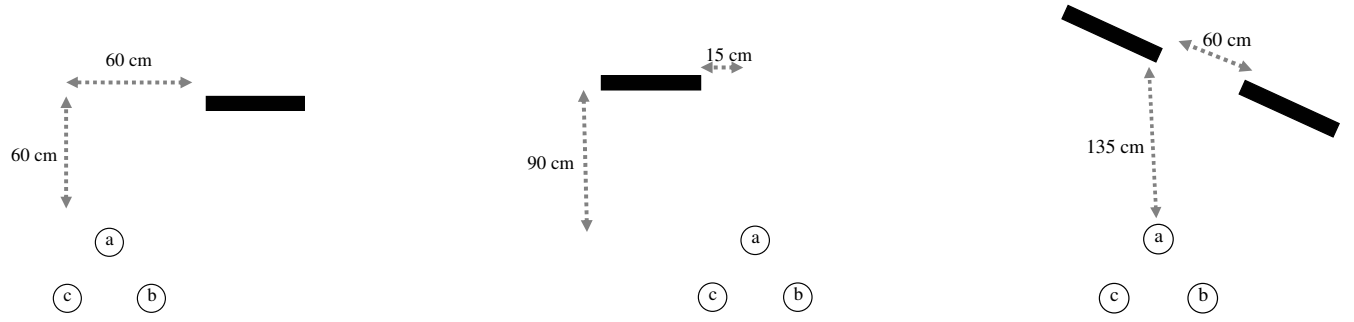


Fig. 18. Fraction of undetected obstacles over 25 runs for each technique.

Thus both in simulation and in experiments in the real world, we see that the multiplexing approach decreases the number of undetected obstacles, though it does not perform as the theoretical best (i.e., with perfect open-loop control and perfect knowledge of obstacles). This happens because the multiplexing technique, while giving more opportunity



(a) Right. (b) Left. (c) Diagonal.

Fig. 15. Three obstacle courses used in experiments with the AIBO robots.

to the followers to detect obstacles, occasionally switches back to sensor-based closed-loop control, for correcting the accumulating odometry errors. In such cases, the follower robots cannot use its sensors to detect obstacles.

2) Formation Precision: An hypothesis underlying the combination approach is that the gains it offers (as the previous section demonstrates) will come at a price of decreased precision. The reliance on open-loop control, even if only for limited periods of time, should in principle cause some degradation in the ability of robots to position themselves in the formation. It might therefore be hypothesized that selecting fusion as the combination method may lead to improved results.

This section examines this hypothesis. We compare the quality of the formation maintenance with different formation techniques, under varying conditions of noise in movement. The quality of the formation maintenance is measured as the average absolute deviation of the follower robots from their ideal location in the formation. Our expectation is that combination would fare worse than its constituent techniques, especially with increased noise.

We compare the multiplexing and fusing combination techniques, presented earlier, to the their two constituent techniques: Open-loop formation control (*communication-based maintenance*), and closed-loop formation control (*visual formation maintenance*). For the combination technique, we use a timeout of 8 seconds for the period in which the robot scans for obstacles, relying only on open-loop control. The timeout was determined empirically, but experimenting with different timeout values.

Precise positioning in formations is relatively easy when the formation moves in a straight line. It becomes more difficult to achieve in realistic settings, when formation (and robots) have to turn. We thus examine the precision resulting from each formation control technique, when the angle of the leader's turn is varied. In the following experiments, the leader robot moves in a straight line for 20 seconds and then turns in place and proceeds. We control the leader's turn angle $(0, 15, 30, 90$ degrees), and measure the resulting position errors in the followers once the turn is complete.

Given that we wanted to control the amount of uncertainty in the movements of the robots, we chose to run these experiments in simulation. We used a Gaussian to model the noise in the movements of the robots, at several qualitative levels of $0\%, 20\%$ and 40%. The percentages signify the the uncertainty in terms of standard deviation, i.e., a level of 20% Gaussian noise means that the standard deviation of target value X will be 20% of X.

Figures 19, 20 and 21 show the results of these experiments, for noise levels 0%, 20% and 40%, respectively. In these figures, the X axis shows the sharpness of the leader's turn in degrees. The Y axis represents the average absolute deviation (error) of the follower robots from their ideal position in the formation. The line marked *visual* shows the results of the closed-loop sensor-based visual maintenance. The line marked *communication* corresponds to the open-loop communication-based maintenance. The lines marked *multiplexing* and *fusing* correspond to the multiplexing and fusing (the combination approaches). Each one of the points is an average over 40 data points (20 runs, two follower robots).

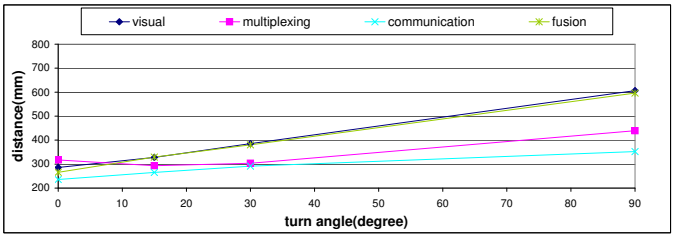


Fig. 19. Deviation from the ideal position in formation vs. the turn angle, with no uncertainty in movement/odometry.

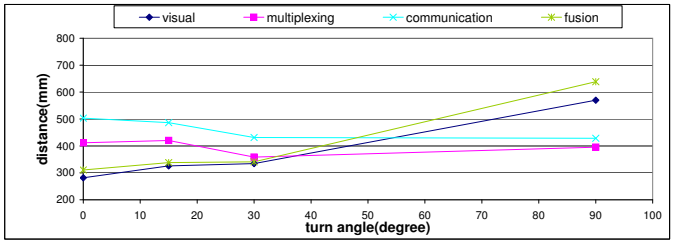


Fig. 20. Deviation from the ideal position in formation vs. the turn angle, with uncertainty levels set at 20%.

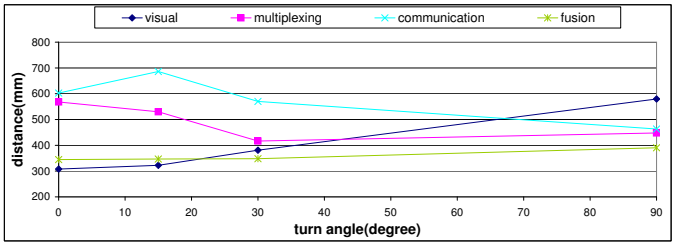


Fig. 21. Deviation from the ideal position in formation vs. the turn angle, with uncertainty levels set at 40%.

The results in Figure 19 show that the worst results are achieved by the visual formation maintenance (closed-loop control, by itself), and by the fusing technique. As the sharpness of the turn increases, use of these techniques lead to increasing errors in the positioning of the follower robots. In contrast, the multiplexing and communication-based maintenance are quite similar in most cases and they have the best results. This happens since in a world where there are fewer odometry errors, a technique that is based on mathematical calculations can calculate the exact location where the follower robot should be and with accurate odometry (i.e. lack of noises) can lead the follower robot to its ideal position in the formation.

However, as odometry noise levels increase, we can see that visual formation maintenance achieves good performance, except for the sharpest (90-degree) turn. Similarly, the fusing technique improves as well, and achieves good performances even in sharp turns. Indeed, the gap between visual maintenance and communication-based maintenance increases (see Figures 20, 21).

Thus one conclusion of these experiments is that the two constituent controllers work well, but not for the same settings. In sharp turns, open-loop control is best (even at higher noise settings). But for robustness to noise, closed-loop control is preferable.

We remind the reader that our hypothesis was that the combination variants would result in decreased precision compared to their constituents. The intuition was that as the combination methods gain the ability to detect obstacles, they sacrifice precision.

The results show that instead, the multiplexing technique emerges as a good controller when the odometry noise level decreases, and is robust in sharp turns as well (a benefit compared to the communication-based controller). It is indeed never the best performer, but it is also never the worst. In fact, this technique seems to be robust both to the noise settings (like its visual maintenance constituent) and to the turn angle sharpness (like the communication-based maintenance constituent). These results thus provide evidence that for robustness, multiplexing controllers (alternating between them) may be a good strategy.

We additionally see that the fusing technique performs well and is robust when the odometry noise level increases. Thus if the robots can recognize its odometry noise level in the environment, it can switch from fusing to multiplexing, and vice versa (depending the noise) and behave ideally. This second level of multiplexing, however, is beyond the scope of this paper.

D. Robust Formations: Conclusions

We introduce here a combination approach (involving either multiplexing or fusing of controllers) to formation maintenance. The approach combines two different formation maintenance controllers: One open-loop and one closed-loop. Our technique helps to maintain a formation and detect obstacles when the robots' sensors are limited, and therefore cannot easily detect obstacles and track their peers at the same time. In experiments with real and simulated Sony AIBO robots, we have found that the combination approach decreases the number of undetected obstacles (compared to the closed-loop visual formation maintenance controller), and maintains the precision of the formation more robustly then either of its constituent controllers by itself. We also conclude that the level of odometry errors influences the best performer between the multiplexing and fusing methods. In particular, we find that the multiplexing technique is better when the odometry error decreases, and that the fusing is preferable otherwise. Thus, we propose to switch between those two methods when the robots knows their odometry error level (e.g., use multiplexing in flat surfaces, and fusing in rocky terrains. Both techniques allow the robots to use their sensors to detect obstacles, something not possible with their constituent methods.

V. Interacting with a Disconnected Formation

Despite all improvements to the operator interface (Section III) and the formation maintenance method (Section IV), in real-world applications there will be times in which a robot fails to keep track of its teammates, and will become stuck in place, while its peers continue. Such cases require resolution by the operator. The interaction of a single human operator with multiple robots poses significant challenges. Quite literally, an operator has only two hands with which it needs to interact with possibly more than a two robots.

Section V-A introduces the challenges involved. Section V-B presents the distributed call-request approach, which utilizes the organizational knowledge of the robots to resolve call-requests in tightly-coordinated tasks. Section V-C presents the results of extensive experiments evaluating this and other approaches.

A. Call-Requests: Introduction

Robots that require the operator's assistance initiate or are issued *call-requests*, which are queued for the operator. Traditionally, the operator switches control between robots, and uses single-robot teleoperation with individual robots to resolve the call requests in some (prioritized) sequence (e.g., [2], [14]). This method works well in settings where the task of each robot is independent of its peers, and thus the resolution of a call request is independent of others. Here, the operator is used as a *centralized* resource by the robots.

Unfortunately, centralized methods face difficulties in *coordinated tasks*—tasks that require tight, continuous, coordination between the robots, i.e., robot teams where robots are highly inter-dependent. First, due to the coordinated nature of the task, robots depend on each other's execution of sub-tasks; thus a single point of failure (e.g., a stuck robot) will quickly lead to multiple call requests. Second, when the operator switches control to a robot, the other robots must wait for the resolution of the call-request, because their own decision-making depends on the results of the operator's intervention. As a result, robots wait idly while the call request is resolved. While monitoring and diagnosis techniques can help localize call-requests to the relevant robot [11], [19], minimizing the duration of call-request resolution remains a key challenge.

Operating a team of coordinated robots raises the opportunity for novel resolution methods, in which the responsibility for the resolution of the call request is *distributed*. Rather than having the operator centrally take all actions to resolve a failure, the otherwise-idle robot teammates can offer assistance, e.g., in providing useful information or in carrying out sub-tasks associated with the resolution process.

For example, in a formation-maintenance task, suppose one of the robots gets stuck, and is unable to move. A call request is issued to the operator, which must identify the failure and attempt to resolve it in some fashion. Previous approaches would have the operator attempt to teleoperate the robot in an attempt to dislodge it, while the other robots are idle [2], [14].

However, the operator could take advantage of the other robots to resolve the failure. First, the other robots could be used to provide video imagery of the stuck robot from various angles. Second, the robots may assist the operator to determine the location of the robots—since they can calculate its expected position with respect to their own position—based on its position within the formation.

This section reports on first steps towards allowing coordinating robots, in a spatial task, to use their knowledge of the coordination to autonomously assist the operator. We examine several variations of a distributed control methodology in which functioning members of the team, rather than switching to an idle mode of operation, actively seek to assist the operator in determining the failure. The key idea is that the responsibility for resolving the call-request is distributed among the team-members *in addition to* the operator.

Moreover, a distributed call-resolution necessarily requires the operator to switch from one robot to the next *while they are moving*. For instance, if the operator is moving one robot, while another requires control, switching time becomes important. Thus the control software on every robot must support quick suspension and resumption of operations, so that switching occurs as quickly as possible. We propose a simple method to enable such quick control-switching.

B. Distributed Call-Request Resolution in a Coordination Task

As previously discussed, centralized resolution of call requests, by the operator, may work well when robots' tasks are independent of each other. However, in coordinated tasks, many robots may have to stop their task execution until a call request is resolved, because their own task execution depends on that of the robot that requires the resolution. In such cases, it is critical to minimize the time it takes to resolve a call request.

We thus focus on a distributed control approach, whereby the robots who depend on the resolution of the call-request take active steps to resolve it, in collaboration with the operator. This approach takes advantage of the robot teamwork, by turning the resolution of the call-request into a distributed collaborative task for all involved. Moreover, the active robots (that do not require assistance) are involved in a coordinated effort with the robot requiring assistance, and thus may be in a better position to assist it.

The key idea behind this approach is that call-request resolution is best viewed as an instance of cooperative problem-solving. During task execution, robots collaborate to achieve the operator goal. If task execution is halted due to a failure, a new collaboration problem instance is generated (resolving the call-request), which should then be addressed by the team-members that are affected by the failure, since they have knowledge which they can bring to bear on the problem.

Concretely, we investigate distributed resolution in repairing broken formations of Sony AIBO 4-legged robots. Formation-maintenance tasks require tight, continuous coordination between robots [20]. When a robot fails and is unable to move, the formation cannot proceed until the failure is resolved in some fashion: Either the robot becomes unstuck, or it is declared dead and the formation proceeds without it. A stuck robot often cannot report on why it is stuck, due to sensory range limitations. For instance, in the AIBO robots, the camera (mounted in the head) cannot pan and tilt to cover the rear legs. Thus if one of them is caught by something, the robots own sensors cannot identify it. The robot must then issue a call-request for assistance. The operator, in turn, must use one of the other robots to locate the stuck robot and get video imagery of its state. This act of locating the other robot and getting sufficiently close to it is a key factor in the resolution of the call request in this case.

We construct two variations of distributed call-request resolution. In the first (*semi-distributed*), the robots assist the operator by autonomously beginning to search for the failing robot as soon as the call request is received. The operator views a split-screen view of their video imagery, and as soon as it identifies the stuck robot in one of the displays, can switch control to the robot associated with the display. Once a robot is taken over by the operator, the others become idle. The operator may still switch control to these other robots, but they no longer work in an autonomous fashion.

The fully-distributed scheme exploits all robots through the call resolution process. The operator may teleoperate any robot at any time, and may switch between controlled robots as needed. When not operator-controlled, the robots first head towards the expected position of their stuck peer. This position is estimated based on their knowledge of the formation (organizational knowledge), under the assumption that the robot became stuck in its previous location within the formation. If they fail to find it there, they begin a spiral search

pattern. The robots that maintain the formation have improved chances to localize themselves (and their stuck peer) with respect to the formation, than an operator which takes control of a robot in the formation, without the situational awareness of the robots. On the other hand, the operator has superior inference and vision, and may be able to better identify the stuck robot in the video imagery.

The distributed approach requires the operator to be able to switch control between robots, and for the robots to be active when the switch occurs. Traditionally, when the operator wants to switch control from robot one robot to the next, she would need to turn on (manually) the first robot's autonomous behavior (for working simultaneously, to achieve the common goal). Then she would need to turn off (manually) the second robot's autonomous controller, and take control of the robot.

When robots do not operate in parallel to the operator, as in previous methods, the effects of switching time on performance is negligible. But when using distributed resolution, switching becomes critical, as the robots that is taken control of is continuing to move and turn even while the operator is switching control over it. Any delay here may cause the robot to move away from where the operator intended to go, thus causing cascading failures.

To allow quick suspension and resumption of autonomous control, each robot maintains a suspend flag which causes the motors to ignore (temporarily, if the flag is on) the controller's commands. When an operator uses the interface to take over control of a robot A, giving up control of robot B, A's associated suspend flag is turned on, and B's flag is turned off, giving B's autonomous controller access to the motors again.

C. Experimental Evaluation of Call-Request Resolution Methods

We empirically evaluated the methods discussed in this paper in extensive experiments, with up to 25 human operators. The first set of experiments focused on comparing the distributed resolution methods presented, with alternative, traditional, methods. We simulated failure cases in a triangular formation (three robots). In each case, we disable one of the robots to simulate a catastrophic failure, not letting it move or communicate. Different call-resolution methods were then used to begin the search process. The search stops when any robot is within a predetermined distance of its failing teammate.

A potential advantage of the distributed and autonomous schemes is that they can utilize the robots' own knowledge of the coordination to locate the stuck robot. In particular, because the robots have moved in formation prior to the call-request, they may have an easier time guessing their peer's location than the operator (who needs to orient herself in space via the teleoperated camera).

We therefore examine three scenarios, in which we varied the accuracy of this knowledge. In all failures cases, the right follower robot was disabled, and color marked to allow its detection by the other robots (called *active robots*) and the operator. We varied the position of the disabled robot (Figure 22): The *easy* setup placed the disabled robot at approximately where it would be had it just stopped in its tracks prior to the team getting notification of the call request, i.e., a bit farther behind its location within the formation (Figure 22-a). The *medium* setup placed the robot behind the left follower robot (22-b). The *difficult* setup placed the robot to the left of the left follower robot, and behind it, i.e., completely out of place compared to the formation (22-c). The locations progress from a location easily predictable by the robots, to a location unpredictable to them.

(a) Easy. (b) Medium. (c) Difficult.

Fig. 22. AIBO robots in initial places for the three experimental setups.

We contrast the distributed and semi-distributed call-resolution methods with two traditional resolution schemes. The first, *teleoperated* scheme corresponds to the centralized control used in previous approaches (e.g., [2], [14]). In this scheme, the operator would switch control from one active robot to the next, as deemed necessary, and manually teleoperated controlled robots (one at a time) until the disabled robot was found. When one robot is controlled, the others remain idle. Another previously-investigated approach is the fully *autonomous* scheme, that lets the active robots (but not the operator) search for the failing robot. This scheme corresponds roughly to the method described in [28], where the robots receive general instructions (here, "search!") by the operator, but are left to translate and follow these commands autonomously, without direct manipulation.

We studied 25 human operators with each of the failure scenarios, each with all methods (22 male, 2 female; 22 of these—including the two females—were graduate or undergraduate students). All operators were novices; none had previous experience controlling multiple robots. The ordering of the scenarios was randomized between operators to prevent ordering effects.

We distinguished two phases: The first phase of the resolution involved recognition of the disabled failure from any distance. The second phase involved its localization by another robot reaching within 35 centimeters of it. Each scenario began with the simulated disabling of the robot (and issuing of the call request), and ended with its localization by at least one robot—teleoperated or autonomous.

For each of the failure scenarios and for each method, we measure the duration of the two phases. This is an objective performance measure because the initial locations of the robots are fixed, the searching speed is constant for non-teleoperated robots, and the termination condition for the search are fixed (robots within specific distance of the failing robot). Thus other than the typical robot sensor uncertainty, performance variance is introduced solely by operator intervention. The first measured duration is that of the time that it took the

operator to recognize the disabled robot in any one of the cameras (the operator uses the split-view interface in this task), i.e., the duration of the first phase. In all but the teleoperated scheme, the operator is completely passive during this interval. We then measure the time that it takes for an active robot—autonomous or teleoperated—to reach the disabled robot, i.e., the duration of the second phase. Since the motivation behind the distributed control scheme is to reduce the time spent awaiting resolution, we prefer shorter overall durations.

We begin by examining the bottom line—the total time it takes to identify the location of the disabled robot. Figure 23 shows the average total duration for the 25 operators. The vertical axis measures the time in seconds, while the horizontal axis shows the three experiment setups. In each, four bars are shown corresponding to the different resolution schemes (left-to-right: Autonomous, semi-distributed, distributed, and teleoperated).

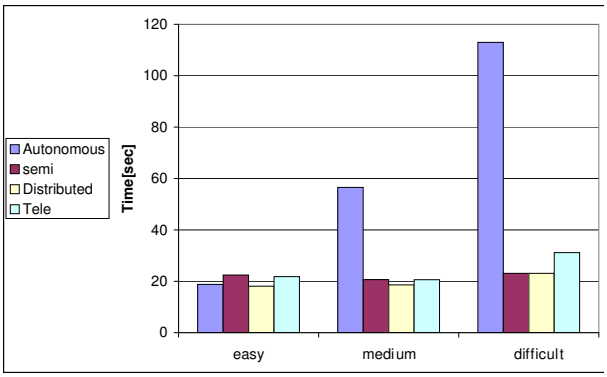


Fig. 23. Total Time to Resolution (in seconds).

The results show that in all *easy*, *medium* and *difficult* locations, the distributed approach is preferable to the both centralized teleoperation approaches, and the fully autonomous approach. Full distributed search does better than the semi-distributed approach in all locations, and better or same than the autonomous approach or same. Overall, the distributed collaboration between the operator and active robots in the distributed approach proves to be a powerful technique for significantly reducing the time to complete the task of locating the disabled robot.

The results have been tested using a one-tailed t-test assuming unequal variances. In the easy setup, the distributed scheme is not significantly different than the autonomous scheme, and only moderately different ($p < 0.12$) than the semi-distributed and teleoperated schemes. However, as we move to the medium and difficult setups, the situation changes. The total time for the distributed scheme is significantly lower than the total time for the autonomous scheme in the latter setups ($p < 0.00004$ and $p < 10^{-12}$, resp.). The distributed scheme does better than the teleoperated scheme in the difficult setup ($p < 0.02$), and is moderately better in the medium setup ($p < 0.13$).

The figure also carries other lessons. First, the ability of the robots to use organizational knowledge of the formation can be very useful in reducing the resolution time, and thus in assisting the operator. When the stuck robot was located approximately where it was predicted to be in terms of its position in the formations, the robots were able to quickly locate it, in fact beating the operator in terms of total time (see more on this below). However, the distributed scheme was superior even in these cases, because even in where the robots were not as successful, the operator (working in collaboration with the robots) was able to compensate. This is particularly evident as the difficulty of the different setups increased, and the location of the stuck robot was unpredictable to the robots.

To better understand these results, we should consider separately the results for the first phase of the search (when an remote identification of the stuck robot was made by the operator), from the second phase, in which an active robot was to approach the stuck robot to localize it. Figure 24 shows the results of the different control schemes for the first phase, averaged across operators. The figure measures the average time (in seconds) it took the operator to recognize the disabled robot from afar, in the split-view camera display. In the autonomous approach, the operator did not intervene in the operation of the robots, only indicated that the stuck robot was recognized. In the teleoperated scheme, the operator manually turned a robot around until a heading to the remote robot was recognized.

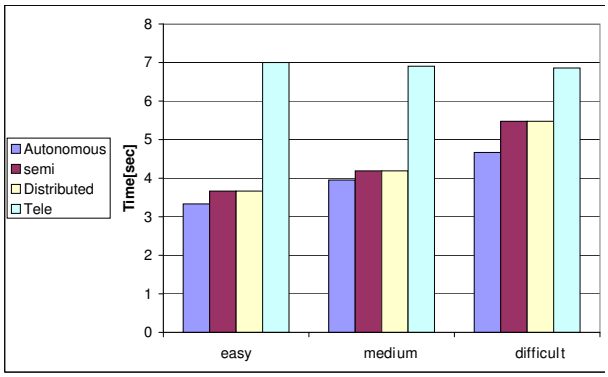


Fig. 24. Phase 1 *Time until initial (remote) identification (in seconds).*

Clearly, all approaches in which robots attempt to orient themselves towards the predicted location of the disabled are superior to a teleoperated (centralized) approach. Note that in all approaches, the operator recognizes the robot from afar. The active robots do not necessarily recognize the other robot from afar, and as we will see below, may end up searching for it in the wrong location. This significantly shorter initial recognition is a beneficial side-effect of the distributed approaches. However, the initial benefits of the robots to orient themselves towards the stuck robot is lost in more difficult settings.

Figure 24 also shows an important property of the usefulness of human operators: Human ability to recognize the robot from afar is virtually identical in all three difficulty settings. Thus humans bring to bear consistent robust (if slow) capabilities. These can be useful in real applications, where the stuck robot may be partially hidden behind obstacles or otherwise not visible at all to the robots.

An examination of the second phase of the search (once an approximate heading towards the stuck robot is determined) is also telling with respect to this issue. Figure 25 shows the results for this phase, where the task is to arrive within the proximity of the disabled robot. Despite its poor performance in phase 1, the teleoperated approach does quite well in phase 2. This is easily explained—here the disabled robot is already recognized, and the teleoperated approach simply allows the operator to now drive the teleoperated robot as quickly as possible, outrunning automatic approaches that move in constant (and typically conservative) speed. Thus again, the operator brings to bear capabilities that cannot be duplicated by the robots.

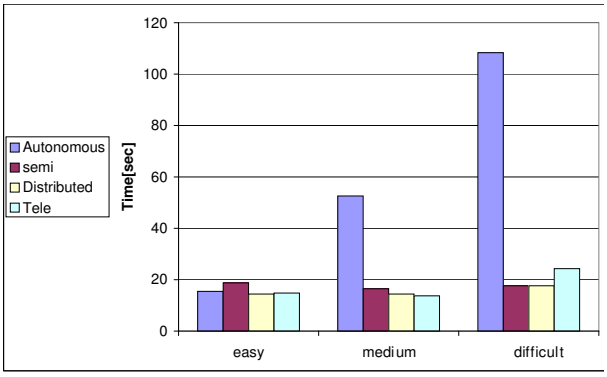


Fig. 25. Phase 2 *From initial identification to localization of the stuck robot* (in seconds).

However, the best performances was by the distributed approach, because it essentially turns this phase into a race between a teleoperated robot and an autonomous robot, as to who gets to the disabled robot first. Moreover, unlike the semi-distributed approach, where there's an overhead of a few seconds while the operator takes over control (see the results for the easy/medium location), here the transition from phase 1 to phase 2 is fairly smooth, because one active robot continues to search even while the operator is taking over control of the other. Thus there is here a composition between the *Autonomous* approach and the *Teleoperated* approach.

Indeed, contrasting the results of the *Autonomous* and *Distributed* approaches is telling. As we move from the easy location to medium to difficult, the gap between the methods is grows in favor to the *Distributed* approach. That happens as a result of the inability of the *Autonomous* approach, to locate the stuck robot in unpredictable places. The collaboration between the human operator and the robot team is superior to either, alone.

An final lesson is revealed by examination of the standard deviation of the results for total task-completion time. Table IV shows the standard deviation for the different approaches, in the three experiment setups. Each row corresponds to a different method, and each column to different setup. We can see that in the easy setup, the autonomous, semi-distributed, and distributed schemes all have essentially the same standard deviation, indicating similar performance. However, the standard deviation for the autonomous scheme in the medium setup is much higher than for the other approaches. In the hard setup, both the autonomous and teleoperated approaches have greater standard deviation in performance than the two distributed schemes. This shows an additional benefit of the distributed methods: A more consistent performance of operators in the distributed and semi-distributed cases.

	Easy	Medium	Difficult
Autonomous	11.21	34.64	23.82
Semi-Dist.	11.30	5.07	7.78
Distributed	11.29	5.16	7.90
Teleoperated	7.68	5.96	15.87

TABLE IV
STANDARD DEVIATION OF CALL-RESOLUTION TIMES (IN SECONDS).

We now turn to empirically evaluate the importance of the switching latency in the distributed resolution methods. We remind the reader that we proposed a simple mechanism (the suspend/resume flag) that enables quick switching. Otherwise, the operator would need to manually perform several preparatory actions (like, manually turning off a controller) in order to teleoperate a robot.

To test the effects of the control-switch methods, we added two control schemes: *SwitchSemi* and *SwitchDistributed*. This two control schemes are similar to the *semi-distributed* and *distributed* control schemes correspondingly except the transfer from the part of identifies the stuck robot in one of the displays to teleoperate one robot. If the operator in the *SwitchSemi-distributed* and *switchDistributed* approaches want to teleoperate a robot she needs to turn off it search behavior manually. While in the *semi-distributed* and *distributed* approaches its automatically happens and the search behavior is paused. The *semi-distributed* and *distributed* is use our quick-pause technique and the *SwitchSemi-distributed* and *switchDistributed* use the old manually technique.

We tested 21 human operators with each of the three failure scenarios (as described in the previous section), and compare there results with 9 operators using the manual-switching techniques. The results are shown in figure 26 (left-to-right: SwitchSemi-distributed, switch-distributed, semi-distributed and distributed). The figure shows the total time to complete the action (First and second phase). We compare each method to its corresponding method (i.e. the manual *SwitchSemi* with the quick *Semi-Distributed*, the manual *SwitchDistributed* with the quick *Distributed*).

We can clearly see that the manual switching methods are far worse than the quick-switching methods. We also see that this effect is of more importance with the more difficult failure cases. This supports our hypothesis that fast operator control-switching is critical to the distributed approaches. Indeed, when we compare the total resolution times of the distributed methods, using the manual switching method, to the autonomous and teleoperated approaches, we find that the advantage of the distributed methods disappears in many cases.

A closer look at the results shows that the differences become apparent in the second phase (Figure 27). The problem arises during the control switching from the first to the second phase. The switching here includes two main steps. First, the time it takes to switch from autonomous behavior to

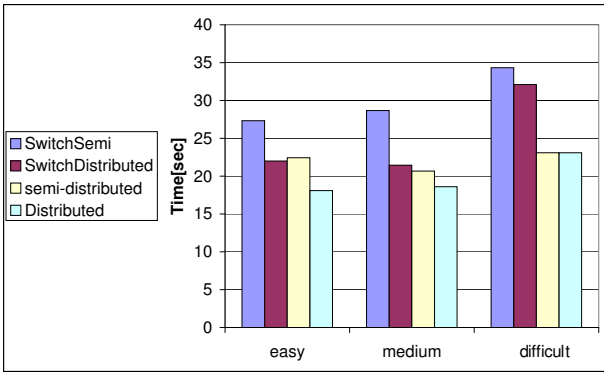


Fig. 26. Total time to resolution, with different switching methods.

teleoperated control must be considered. A secondary issue arises due to this time loss. As the robots act autonomously until the operator assumes control, robots can often take a number of steps after their initial recognition. This results in the robots moving away from the location of their first recognition. We refer to this cascading failure as a *secondary failure*.

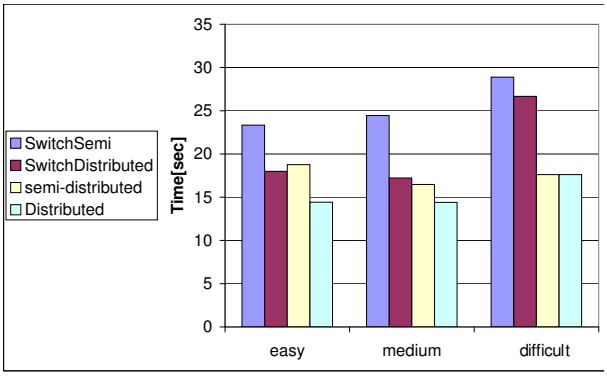


Fig. 27. A comparison of phase-2 durations for different switching methods.

We hypothesize that these secondary failures exist, i.e., that even if the switching duration itself constituted fixed latency, then the effects of this latency would cause additional failures, and thus would vary the actual latency observed. In order to support this claim, we studied the ratio between the *distributed* and *SwitchDistributed* techniques.

Table V shows the difference and ratio between the *distributed* and *SwitchDistributed* methods at easy, medium and hard courses. The results of this table attempt to isolate the delay due to the switching control between the *distributed* and *SwitchDistributed* methods. The tables demonstrates that the difference and ratio between these methods varies. Under the assumption that normal switching behaviors require a fixed amount of time, this result implies that another factor exists that accounts for this variable length. We believe that this factor is the secondary failures caused by the switching latency.

In particular, we believe these secondary failures resulted from the robot's movement from the moment the operator recognized the stuck robot in its video stream, and until the operator was able to control the robot.

	Easy	Medium	hard
difference	3.905	2.84	9.015
ratio	1.21	1.15	1.39

TABLE V
DIFFERENCE AND RATIO BETWEEN *distributed* AND *SwitchDistributed* METHODS.

The results have been tested using a one-tailed t-test assuming unequal variances. The SwitchDistributed scheme was not statistically significantly different for all courses (easy: $\rho < 0.2$, medium: $\rho < 0.3$, hard: $\rho < 0.12$) in respect to the distributed. The SwitchSemi scheme is statistically significantly different than the distributed in all courses (easy: $\rho < 0.012$, medium: $\rho < 0.006$, hard: $\rho < 0.05$).

D. Interacting with a Formation: Summary

This paper explores novel first steps towards distributed call-request resolution schemes, in which the operator and robots collaborate to resolve failures. This scheme is particularly suited to situations where robots are tightly coordinated, and thus are able to use their knowledge of the coordination to effectively assist the operator. The technique builds on a key idea, that the resolution of failures in cooperative tasks should be viewed as a cooperative task in itself. Previous techniques (teleoperation of one robot at a time, autonomous operation of the robots) were meant for tasks that do not require tight coordination between the robots.

We empirically evaluate the distributed resolution methods (and contrast them with previous approaches) in extensive experiments with up to 25 human operators, operating a team of 3 Sony AIBO robots that are moving in formation. The experiments evaluate several concrete call-request scenarios, in which a stuck robot must be located by the operator. The results show that distributed call-request resolution leads to shorter failure-recovery times. Moreover, the results show that a key factor in the success of the distributed method lies in the robots' use of organizational knowledge (i.e., their knowledge of the coordination). However, even in cases where this organizational knowledge fails, the operator is able to compensate. Thus the use of our distributed approach is always better than either the operator or the robots resolving the call request by themselves. We also report on the empirical results of using quick-switch methods (automatic suspension of autonomous activities by a robot, upon the operator switching control to it). A final promising result is that the distributed methods lead to improved operator consistency, reducing the variance in performance between operators.

VI. SUMMARY

This paper tackles key challenges in making formation-maintenance a reality in real-world applications of multi-robot teams. It provides a comprehensive set of techniques that address robustness concerns, both from the perspective of a human operator of the formation, as well as from the point

of view of maintaining greater autonomy by the robot team. Specifically, three sets of techniques are presented:

1) First, the paper presented a novel ecological coordination display, which we show improves the performance of human operators guiding a formation through obstacle courses. Performance improves in the time it takes to navigate the chosen path, the number of failures and the failure rate, and the consistency of the operator's success.
2) Then, the paper presented an approach for combining sensor-based closed-loop formation-maintenance, and communication-based open-loop formation maintenance, by either fusion or multiplexing in time. The experiments show that this allows robots to better utilize their sensors for detecting and avoiding obstacles, while still maintaining their positions in the formation.
3) Finally, the paper presented a novel distributed call-request approach to handling call-requests (operator intervention requests) in tightly-coordinated tasks. The key to this approach is that robots actively collaborate with the operator to resolve conflicts, in particular utilizing their knowledge of the coordination to assist the operator in locating robots that require assistance.

Extensive experiments using real and simulated robots, with up to 25 human operators, show significant improvements over existing methods, in all three contribution areas. The techniques clearly provide a path towards real-world applications of multi-robot formations. Videos showing actual runs in which these techniques were used, as well as videos of related techniques, are available at http://www.cs.biu.ac.il/∼maverick/Movies/ [25].

ACKNOWLEDGMENTS

We thank Avi Rosenfeld for useful comments, and Ruti Glick for help in organizing the experiments. Special thanks to K. Ushi. This work was supported in part by ISF Grant #1357/07.

REFERENCES

[1] J. Adams, A. Robertson, K. Zimmerman, and J. How. Technologies for spacecraft formation flying. In *Proceedings of ION-GPS-96*, pages 1321–1330, 1996.

[2] J. A. Adams. *Human Management of a Hierarchical System for the Control of Multiple Mobile Robots*. PhD thesis, University of Pennsylvania, 1995.

[3] K. S. Ali. *Multiagent Telerobotics: Matching systems to tasks*. PhD thesis, Georgia Institute of Technology, 1999.

[4] H. Asama, A. Matsumoto, and Y. Ishida. Design of an autonomous and distributed robot system: ACTRESS. In *Proceedings. of the 1989 IEEE/RSJ International workshop on intelligent robots and system*, pages 283–290, 1989.

[5] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[6] T. Balch and M. Hybinette. Social potentials for scalable multirobot formations. In *Proceedings of IEEE International Conference on robotics and automation (ICRA-00)*, 2000.

[7] A. Broggi, M. Bertozzi, A. Fascioli, C. G. L. Bianco, and A. Piazzi. Visual perception of obstacles and vehicles for platooning. *IEEE Transactions on Intelligent Transportation Systems*, 1(3):164–176, 2000.

[8] X. Chen and Y. Li. Smooth formation navigation of multiple mobile robots for avoiding moving obstacles. *International Journal of Control, Automation, and Systems*, 4(4):466–479, August 2006.

[9] J. P. Desai. A graph theoretic approach for modeling mobile robot team formations. *Journal of Robotic Systems*, 19(11):511–525, 2002.

[10] J. P. Desai, J. P. Ostrowski, and V. Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, 2001.

[11] R. J. Doyle. Determining the loci of anomalies using minimal causal models. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1821–1827, Montreal, Quebec, Canada, 1995.

[12] M. Fields. Modeling the human/robot interaction in OneSAF. In *Proceedings of the 23rd Army Science Conference*, 2002. (Poster).

[13] R. Fierro, A. K. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. *IEEE International Conference on Robotics and Automation*, 2001.

[14] T. Fong, C. Thorpe, and C. Baur. Multi-robot remote driving with collaborative control. *IEEE Transactions on Industrial Electronics*, 50(4):699–704, August 2003.

[15] J. Fredslund and M. J. Mataric. A general algorithm for robot formations using local sensing and minimal communications. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 2002.

[16] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, Jul 2003.

[17] M. A. Goodrich and A. C. Schultz. Human-robot interaction: A survey. *Foundations and Trends in Human-Computer Interaction*, 1(3):203–275, 2007.

[18] C. A. Johnson, J. A. Adams, and K. Kawamura. Evaluation of an enhanced human-robot interface,. In *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, Washington, DC, 2003.

[19] M. Kalech, G. A. Kaminka, A. Meisels, and Y. Elmaliach. Diagnosis of multi-robot coordination failures using distributed CSP algorithms. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[20] G. A. Kaminka, R. Glick, and V. Sadov. Using sensor morphology for multi-robot formations. *IEEE Transactions on Robotics*, 2008. To appear.

[21] G. A. Kaminka and M. Tambe. Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.

[22] H. K. Keskinpala and J. A. Adams. Usability analysis of a PDA-based interface for a mobile robot,. *Human-Computer Interaction*, 2004.

[23] H. K. Keskinpala, J. A. Adams, and K. Kawamura. PDA-based human-robotic interface. In *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, Washington, DC, 2003.

[24] M. Lemay, F. Michaud, D. Létourneau, and J.-M. Valin. Autonomous initialization of robot formations. *IEEE International Conference on Robotics and Automation*, pages 3018–3023, 2004.

[25] The MAVERICK Group movies page, Computer Science department, Bar Ilan University. http://www.cs.biu.ac.il/∼maverick/Movies/, Last checked: Feb 24, 2008.

[26] F. Michaud, D. Létourneau, M. Gilbert, and J.-M. Valin. Dynamic robot formations using directional visual perception. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[27] A. I. Mourikis and S. I. Roumeliotis. Optimal sensor scheduling for resource constrained localization of mobile robot formations. *IEEE Transactions on Robotics*, 22(5):917–931, October 2006.

[28] K. L. Myers and D. N. Morely. Human directability of agents. In *Proceedings of the First International Conference on Knowledge Capture, K-CAP 2001*, Canada, 2001.

[29] C. W. Nielsen, M. A. Goodrich, and R. W. Ricks. Ecological interfaces for improving mobile robot teleoperation. *IEEE Transactions on Robotics and Automation*, 23(5):927–941, 2007.

[30] P. Ogren and N. E. Leonard. Obstacle avoidance in formation. In *Proceedings. of the IEEE Int. Conference. on Robotics and Automation (ICRA)*, Taipei, Taiwan, 2003.

[31] R. W. Ricks, C. W. Nielsen, and M. A. Goodrich. Ecological displays for robot interaction: A new perspective. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-04)*, 2004.

[32] P. E. Rybski, I. Burt, T. Dahlin, M. Gini, D. F. Hougen, D. G. Krantz, F. Nageotte, N. Papanikolopoulos, and S. A. Stoeter. System architecture for versatile autonomous and teleoperated control of multiple miniature robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2001.

[33] P. Scerri, L. Johnson, D. Pynadath, P. Rosenbloom, M. Si, N. Schurr, and M. Tambe. A prototype infrastructure for distributed robot, agent, person teams. In *AAMAS-03*, 2003.

[34] M. Skubic, D. Anderson, S. Blisard, D. Perzanowski, and A. Schultz. Using a qualitative sketch to control a team of robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06)*, 2006.

[35] T. Suzuki, K. Yokota, H. Asama, H. Kaetsu, and I. Endo. Cooperation between the human operator and the multi-agent robotic system: Evaluation of agent monitoring methods for the human interface system. In *Proceedings. of the 1995 IEEE/RSJ International conference on intelligent robots and systems*, pages 206–211, 1995.

[36] The Tekkotsu Homepage. www.tekkotsu.org, 2002.

[37] A. D. Tews, M. J. Mataric, and G. S. Sukhatme. A scalable approach to human-robot interaction. In *ICRA-03*, 2003.

[38] S. Venkataramanan and A. Dogan. Nonlinear control for reconfiguration of UAV formation. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2003.

[39] K. A. Vicente. Ecological interface design: progress and challenges. *Human Factors*, 44(1):62–78, 2002.

[40] Z. Wang, Y. Hirata, and K. Kosuge. Control a rigid caging formation for cooperative object transportation by multiple mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-04)*, pages 1580–1585, 2004.

[41] H. A. Yanco, J. L. Drury, and J. Scholtz. Beyond usability evaluation: Analysis of human-robot interaction at a major robotics competition. *Journal of Human-Computer Interaction*, 19(1 and 2):117 – 149, 2004.

[42] K. Yokota, T. Suzuki, H. Asama, A. Masumoto, and I. Endo. A human interface system for the multi-agent robotic system. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-94)*, pages 1039–1044, 1994.

[43] D. Zwillinger, editor. *CRC Standard Mathematical Tables and Formulae*, chapter 4.3, pages 312–314. CRC press, 30th edition, 1995.

Linear-Time Temporal Logic Control of Discrete Event Models of Cooperative Robots

Bruno Lacerda, Pedro Lima
Institute for Systems and Robotics
Instituto Superior Técnico
Lisbon, Portugal
{blacerda, pal}@isr.ist.utl.pt

***Abstract*—A Discrete Event System (DES) is a discrete state space dynamic system that evolves in accordance with the instantaneous occurrence, at possibly unkown times, of physical events. Given a DES, its behavior (the sequence of displayed events) may not satisfy a set of logical performance objectives. The purpose of Supervisory Control is to restrict that behavior in order to achieve those objectives. Linear-Time Temporal Logic (LTL) is an extension of Propositional Logic which allows reasoning over an infinite sequence of states. We will use this logical formalism as a way to specify our performance objectives for a given DES and build a supervisor that restricts the DES' behavior to those objectives by construction. Several simulated application examples illustrate the developed method.**

***Index Terms*—Discrete Event Systems, Supervisory Control, Linear-Time Temporal Logic**

I. Introduction

In recent years there has been a considerable interest in Discrete Event Systems (DES), whose discrete states change in response to the occurrence of events from a predefined event set. Examples of such systems can be found in communication networks, computer programs, operating systems, manufacturing processes and robotics. One of the main fields of study is Supervisory Control, introduced in [9] and further developed in [2], which focuses on the restriction of a DES behavior in order to satisfy a set of performance objectives. This restriction is, in many cases, performed in an *ad-hoc* manner, but with the continuing growth of this type of systems, a more generalized framework is needed. In this work, we present a framework to restrict a DES behavior specifying its performance objectives with Linear-Time Temporal Logic (LTL). Using this approach, we guarantee that the required behavior is achieved by construction. Furthermore, in many cases, the specification of the performance objectives using LTL is almost immediate, allowing the supervision of more complex systems. A great deal of work has been done recently in a slightly different context: controlling continuous state space time-driven linear systems with LTL specifications ([1], [7], [10]). In this context, a discretization of the linear system is needed before the LTL specification can be enforced, obtaining a discrete space system. The system is then refined to a hybrid system. This approach is mainly used to perform robot motion planning (enforcing a robot to go to certain places and avoid certain obstacles). Our approach is different because we will be concerned with, given a team of robots where we assume that each one can perform a number of tasks individually, coordinating their behavior so that they reach a given objective. For this purpose DES models are more suitable and reduce the involved complexity by comparison to hybrid systems models. LTL enables the formulation of complex sentences by compact logical sentences.

The work in divided in three main Sections: In Section 2 we introduce the notions of Discrete Event System and Supervisory Control, explaining how one can see a Finite State Automaton as a DES. In Section 3 we define Linear-Time Temporal Logic and mention a method to build a Büchi automaton that accepts exactly the ω-language of the infinite sequences that satisfy a given formula φ. Finally, in Section 4 we congregate all the theory defined throughout this work to present our method of supervisory control and give some operational examples of applications of the presented method. The developed approach is illustrated with simulation examples that are deployed along the paper.

II. Discrete Event Systems

A. Preliminaries

Definition 1 (Discrete Event System): A Discrete Event System is composed of a discrete set X of possible states and a finite set $E = \{e_1, ..., e_m\}$ of possible events.

At a given time $t \geq 0$, the DES is in a given state $x \in X$, which is all the information needed to characterize the system at that time instant. The state of a DES can only be changed by the occurence of an event $e \in E$ and these events occur both instanteneously and asynchronously.

The set X is called the *state-space* of the DES and the set E is called the *event-space* of the DES. Both these sets must be discrete and E must be finite. We can interpret the state as the task the system if performing at a given moment, such as a robot moving forward, a machine being idle or a computer running a program. The events are interpreted as physical phenomenons, such as a robot's sensor detecting something, a new job arriving to a machine or a program crashing.

Example 1 (Transporting robots): Consider two robots, each one holding one end of a bar. Their objective is to transport the bar to another place. To simplify, assume that the robots can only move a constant distance forward or stop.

This situation can be modeled as a DES with $X = \{$Both robots stopped, Robot$_1$ moving and Robot$_2$ stopped, Robot$_1$ stopped and Robot$_2$ moving, Both robots moving$\}$ and $E = \{Move_1,\ Move_2,\ Stop_1,\ Stop_2\}$.
A sequence of events in this DES can be $((Move_1,\ t_1)$, $(Stop_1,t_2)$, $(Move_1,t_3)$, $(Move_2,\ t_4)$, $(Stop_1,t_5)$, $(Stop_2,\ t_6))$, $t_1 < t_2 < ... < t_6$.

In this example, one of the robots can move forward to a position where it is too far from the other one, making the bar fall.

B. Modeling Logical DES

There are three levels of abstraction usually considered in the study of DES, *Untimed (or logical) DES models*, *Deterministic Timed DES Models* and *Stochastic Timed DES Models*.

The theory of Supervisory Control is defined over Logical DES Models, so in this work we will introduce Finite State Automata as our modeling framework.

Definition 2 (Finite State Automaton): A Finite State Automaton (FSA) is a six-tuple $G = (X, E, f, \Gamma, X_0, X_m)$ where:

- X is the finite set of states
- E is the finite state of events
- $f : X \times E \to X$ (deterministic) or $f : X \times E \to 2^X$ (nondeterministic) is the (possibly partial) transition function
- $\Gamma : X \to 2^E$ is the active event function
- $X_0 \subseteq X$ is the initial state (a singleton for deterministic FSA)
- $X_m \subseteq X$ is the set of marked states

Deterministic FSA (DFA) and Nondeterministic FSA (NFA) are equivalent, as proven in [5]. The following definitions will be made for DFA, but the generalization for NFA is straightforward. $f(x, e) = y$ means that there is a transition labeled by event e from state x to state y. $\Gamma(x)$ is the set of all events e for which $f(x, e)$ is defined. Note that Γ is uniquely defined by f, it was included in the definition for convenience. We also extend f from domain $X \times E$ to domain $X \times E^*$ in the following recursive manner:

- $f(x, \epsilon) = x$
- $f(x, se) = f(f(x, s), e),\ s \in E^*,\ e \in E$

Now, we are in conditions to define the languages generated and marked by a DFA. As we will see, the objective of Supervisory Control is to restrict these languages to the strings we consider "legal" for our system.

Definition 3 (Generated and Marked Languages): Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a DFA. We define

- $L(G) = \{s \in E^* : f(x_0, s)\ is\ defined\}$, the language generated by G
- $L_m(G) = \{s \in L(G) : f(x_0, s) \in X_m\}$, the language marked by G

The notion of marked language is used to model "complete tasks" of a DES. We will now introduce three operations over DFA that are very useful in DES modeling and necessary to perform supervision.

Definition 4 (Acessible Part): Let $G = (X, E, f, \Gamma, x_0, X_m)$ be a DFA. The accessible part of G is the DFA $Ac(G) = (X_{ac}, E, f_{ac}, \Gamma_{ac}, x_0, X_{ac,m})$ where

- $X_{ac} = \{x \in X : \exists s \in E^* f(x_0, s) = x\}$
- $X_{ac,m} = X_m \cap X_{ac}$
- $f_{ac} = f|_{X_{ac} \times E \to X_{ac}}$
- $\Gamma_{ac} = \Gamma|_{X_{ac} \to 2^E}$

The accessible part of a DFA is simply its restriction to the states that can be reached from the initial state. f_{ac} is the restriction of f to domain $X_{ac} \times E$ and Γ_{ac} is the restriction of Γ to domain X_{ac}. It is clear that $L(G) = L(Ac(G))$ and $L_m(G) = L_m(Ac(G))$.

Definition 5 (Product Composition): Let $G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1})$ and $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$ be two DFA. The product composition of G_1 and G_2 is the DFA $G_1 \times G_2 = Ac(X_1 \times X_2, E_1 \cap E_2, f, \Gamma_{1\times2}, (x_{01}, x_{02}), (X_{m1} \times X_{m2}))$ where

$$f((x_1, x_2), e) = \begin{cases} (f_1(x_1), f_2(x_2)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

and thus $\Gamma_{1\times2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$

The product composition is also called the *completely synchronous composition*. In this composition, the transitions of the two DFA must always be synchronized on a common event $e \in E_1 \cap E_2$. This means that an event occurs in $G_1 \times G_2$ if and only if it occurs in both DFA. Thus, it is easily verified that $L(G_1 \times G_2) = L(G_1) \cap L(G_2)$ and $L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$.

Definition 6 (Parallel Composition): Let $G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1})$ and $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$ be two DFA. The parallel composition of G_1 and G_2 is the DFA $G_1 \parallel G_2 = Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_{1\|2}, (x_{01}, x_{02}), (X_{m1} \times X_{m2}))$ where

$$f((x_1, x_2), e) = \begin{cases} (f_1(x_1), f_2(x_2)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1), x_2) & \text{if } e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1, f_2(x_2)) & \text{if } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

and thus $\Gamma_{1\|2}(x_1, x_2) = [\Gamma(x_1) \cap \Gamma(x_2)] \cup [\Gamma(x_1) \setminus E_2] \cup [\Gamma(x_2) \setminus E_1]$

The parallel composition is also called the *synchronous composition*. In this composition, an event in $E_1 \cap E_2$ (common event) can only be executed if the two DFA both execute it sinultaneously. An event in $(E_2 \setminus E_1) \cup (E_1 \setminus E_2)$ (private event) can be executed whenever possible. If $E_1 = E_2$, then the parallel composition reduces to the product, since all transitions must be synchronized and if $E_1 \cap E_2 = \emptyset$, then there are no synchronized transitions and $G_1 \parallel G_2$ models the concurrent behavior of G_1 and G_2 (in this case we call $G_1 \parallel G_2$ the *shuffle* of G_1 and G_2).

Example 2 (Transporting Robots): The DES of Example 1 can be modeled by the FSA shown in Figure 1.

Another way of modeling this system is using parallel composition, which is very useful when our system has several components operating concurrently. It allows us to model each component separately and then get the FSA that

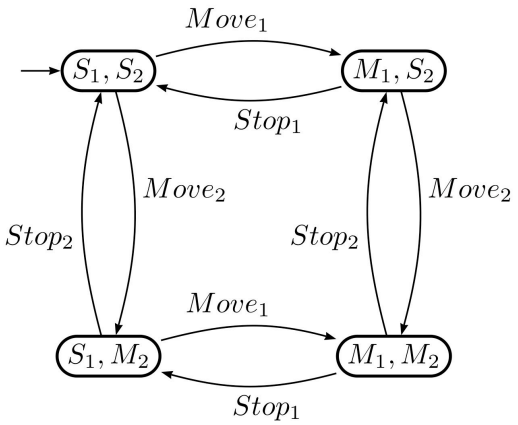


Fig. 1. FSA model of Transporting Robots

models the whole system by applying it. Hence, if we model each robot separately, we obtain the FSA $G_i, i = 1, 2$, seen in Figure 2.

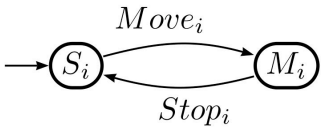


Fig. 2. FSA model of Transporting Robot i

It is easy to see that $G_1 \parallel G_2$ is the FSA represented in Figure 1.

Example 3 (Robotic Soccer): Consider a team of n robots playing a soccer game. The objective is to reach a situation in which one of the robots is close enough to the goal to shoot and score. When a robot does not have the ball in its possession, it has two options:

- Move to the ball until it is close enough to take its possession, or
- Get ready to receive a pass from a teammate.

When a robot has the possession of the ball, it can:

- Shoot the ball (if it is close enough to the goal), or
- Take the ball to the goal, if there is no opponent blocking its path, or
- Choose a teammate to pass the ball and, when it is ready. to receive, pass it

For simplicity, we assume that, when a robot shoots the ball, the team loses its possession (we do not differentiate the situation where the robot scores from the situation where the robot does not score since the team will lose the ball's possession in both) and that the opponents do not steal the ball (they are only able to block paths, at which point our robot will try to pass to a teammate). Figure 3 depicts a possible FSA R_i model for robot i. An FSA model for the whole team is given by $T = R_1 \parallel R_2 \parallel ... \parallel R_n$. Note that the $pass(i, j)$ event must be synchronized between robot i (the passing robot) and robot j (the receiving robot).

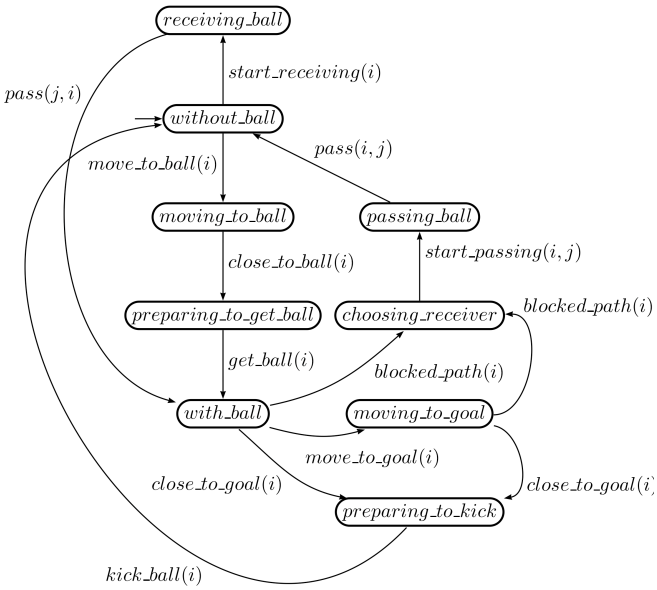


Fig. 3. FSA for Robot R_i

Note that, when we write $start\_passing(i, j)$, $pass(i, j)$ and $pass(j, i)$ in a transition, we are representing $n-1$ events, since $j = 1, ..., n$, $j \neq i$.

III. Supervisory Control

As we have seen in previous examples, sometimes our DES model has some behaviors that are not satisfactory. Let's assume we have a DES modeled by FSA G. G models the "uncontrolled behavior" of the DES and is called the *plant*. Our objective is to modify the plant's behavior, i.e., restrict its behavior to an admissible language $L_a \subseteq L(G)$, using control.

To do this, we start by partitioning the event set E in two disjoint subsets $E = E_c \cup E_{uc}$.

E_c is the set of *controllable events*, i. e., the events that can be prevented from happening and E_{uc} is the set of *uncontrollable events* , i.e., the events that cannot be prevented from happening. This partition is due to the fact that, in general, there are events that make a DES change its state that are not of the "responsibility" of the DES itself.

Example 4: We list the set of controlled and uncontrolled events in previous examples.

- In Example 2, we assume that the robots can only move a constant distance forward. Hence, after a robot starts moving, the decision to stop is not its responsibility, it always stops after it moves the predefined distance.
  - $E_c = \{Move_1,\ Move_2\}$
  - $E_{uc} = \{Stop_1,\ Stop_2\}$
- In Example 3 the events $close\_to\_ball$, $close\_to\_goal$ and $blocked\_path$ are caused by changes in the environment around the robots and not by the robots themselves.

Therefore, they are considered uncontrollable events. The controllable events correspond to the actions available to each robot.

- $E_c = \{move\_to\_ball(i),\ get\_ball(i),\ kick\_ball(i),\ move\_to\_goal(i),\ \ start\_passing(i,j),\ start\_receiving(i),\ pass(i,j) : i,j = 1,...,n,\ j \neq i\}$
- $E_{uc} = \{close\_to\_ball(i),\ blocked\_path(i),\ close\_to\_goal(i) : i = 1,...,n\}$

Next, we introduce the notion of a DES $G = (X, E = E_c \cup E_{uc}, f, \Gamma, X_0, X_m)$ controlled by a supervisor S. Formally, a supervisor is a function $S : L(G) \rightarrow 2^E$ that, given $s \in L(G)$ outputs the set of events G can execute next (*enabled events*). We only allow supervisors S such that, when event $e \in E_{uc}$ is active in the plant G, it is also enabled by S. That is, a supervisor must always allow the plant to execute its uncontrollable events.

Definition 7 (Admissible Supervisor): Let $G = (X, E = E_c \cup E_{uc}, f, \Gamma, x_0, X_m)$ be a DES and $S : L(G) \rightarrow 2^E$. S is an admissible supervisor for G if for all $s \in L(G)$ $E_{uc} \cap \Gamma(f(x_0, s)) \subseteq S(s)$.

We will check the admissibility of our supervisors S in a case-by-case basis.

Definition 8 (Controlled DES): Let $G = (X, E = E_c \cup E_{uc}, f, \Gamma, x_0, X_m)$ be a DES and $S : L(G) \rightarrow 2^E$. The controlled DES (CDES) S/G (S controlling G) is a DES that constrains G in such a way that, after generating a string $s \in L(G)$, the set of events that S/G can execute next (enabled events) is $S(s) \cap \Gamma(f(x_0, s))$.

The way S/G operates is represented in Figure 4 and is as follws: s is the string of all events executed so far by G, which is observed by S. S uses s to determine what events should be enabled, that is, which events can occur after the generation of s.

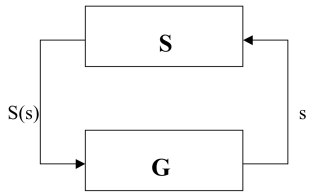


Fig. 4. The feedback loop of supervisory control

Definition 9 (Generated and Marked Languages by a CDES): Let S/G be a CDES and e one of its events. The language generated by S/G, $L(S/G)$, is defined as follows:

- $\epsilon \in L(S/G)$;
- if $s \in L(G)$ and $se \in L(G)$ and $e \in S(s)$ then $se \in L(S/G)$.

and the language marked by S/G, $L_m(S/G)$, is

- $L_m(S/G) = L(S/G) \cap L_m(G)$.

Thus, given a plant G and an admissible language $L_a \subseteq L(G)$, we want to find a supervisor S such that $L(S/G) = L_a$ (in this work we will be focused on generated languages and will not be concerned with marked languages).

In this framework, the supervisor is usually implemented by an FSA R, such that $L(R) = L_a$. R is refered to as the *standard realization* of S. The most common method to build R is to start by building a simple FSA H_{spec} that captures the essence of the natural language specification and then combine it with G, using either product or parallel composition. We choose parallel composition if the events that appear in G but not in H_{spec} are irrelevant to the specification that H_{spec} implements or product composition when, on the other hand, the events that appear in G but not in H_{spec} should not happen in the admissible behavior L_a.

Having the FSA $G = (X_G, E_G, f_G, \Gamma_G, x_{G,0}, X_{G,m})$ and $R = (X_R, E_R, f_R, \Gamma_R, x_{R,0}, X_{R,m})$ that represent the plant and the standard realization of S respectively (note that $E_R \subseteq E_G$), the feedback loop of supervisory control is implemented as follows: Let G be in state x and R be in state y following the execution of string $s \in L(S/G)$. G executes an event e that is currently enabled, i.e., $e \in \Gamma_G(x) \cap \Gamma_R(y)$. R also executes the event, as a passive observer of G. Let $x' = f_G(x, e)$ and $y' = f_R(y, e)$ be the new states of G and R respectively, after the execution of e. The set of enabled events of G after string se is now given by $\Gamma_G(x') \cap \Gamma_R(y')$. It is common to make $X_{R,m} = X_R$, so that $R \times G$ represents the closed-loop system S/G:

- $L(R \times G) = L(R) \cap L(G) = L_a \cap L(G) = L_a = L(S/G)$
- $L_m(R \times G) = L_m(R) \cap L_m(G) = L_a \cap L_m(G) = L(S/G) \cap L_m(G) = L_m(S/G)$

So, from now on, we will refer to a supervisor S and its standard realization R interchangeably.

Next, we address modular supervision, a mean of reducing the complexity of the controlled DES model.

Definition 10 (Modular Supervision): Let $S_1, ..., S_n,\ n \in \mathbb{N}$ be admissible supervisors for DES $G = (X, E = E_c \cup E_{uc}, f, \Gamma, x_0, X_m)$ and $s \in L(G)$. We define the (admissible) modular supervisor as

- $S_{mod12...n}(s) = S_1(s) \cap S_2(s) \cap ... \cap S_n(s)$

It is obvious, by definition 7 that $S_{mod12...n}$ is admissible for G. In Figure 5 we represent modular supervision with 2 supervisors. In modular control, an event is enabled by $S_{mod12...n}$ if and only if it is enabled for all $S_i,\ i = 1, ..., n$.

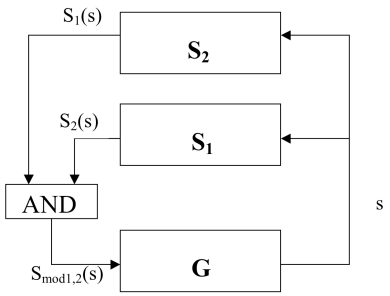


Fig. 5. The feedback loop of modular supervisory control

Remark 1 (Multiple Specifications): When our admissible behavior is composed of multiple specifications, that is, when $L_a = L_{a,1} \cap ... \cap L_{a,n}$, where $L_{a,i}$ represents a given specification we want our plant G to satisfy, we will build n supervisors $S_i,\ i = 1, ..., n$ such that $L(S_i/G) = L_{a,i}$ and use modular control to implement a supervisor $S_{mod1...n}$ such that $L(S_{mod1...n}/G) = L_a$.

Example 5 (Transporting Robots): As we have mentioned, it is possible for one robot to move forward to a position where it is too far from the other, making the bar fall. One way to avoid this is to impose alternation between the robots' motion: one robot moves forward while the other is stopped, holding the bar. Then the other robot moves forward while the one that moved before is stopped, holding the bar, etc. So, we have 4 specifications:

- **Spec** 1 **-** Robot 1 cannot start moving while Robot 2 is moving;
- **Spec** 2 **-** Robot 2 cannot start moving while Robot 1 is moving;
- **Spec** 3 **-** After Robot 1 moves, it will only start moving again after Robot 2 has moved;
- **Spec** 4 **-** After Robot 2 moves, it will only start moving again after Robot 1 has moved.

Example 6 (Robotic Soccer): Regarding Example 3, one may define the following specifications, which are useful to improve the team's performance in a soccer game for each Robot i:

- **Spec** $1, i$ **-** If another teammate goes to the ball, robot i will not go to the ball until it is kicked by some robot in the team;
- **Spec** $2, i$ **-** Robot i will not get ready to receive a pass, unless one of its teammates decides to pass it the ball and, in this case, it will be ready to receive the pass as soon as possible.

Spec $1, i$ guarantees that only one robot moves to the ball at a time and that, when the team has the ball, no robot moves to it and Spec $2, i$ guarantees that no robot will be ready to receive a pass when none of its teammates wants it to receive a pass and that when a robot wants to pass the ball, another one will get ready to receive it as soon as possible .

IV. Linear-Time Temporal Logic and Büchi Automata

In this Section we introduce Linear-Time Temporal Logic (LTL). We start by defining the syntax and semantics of LTL and then refer the translation from LTL formulas to Büchi Automata.

A. Linear-Time Temporal Logic

LTL is an extension of Propositional Logic which allows reasoning over an infinite sequence of states. LTL is widely used for verification of properties of several concurrent systems (for example, *safety* and *liveness*), especially software systems. In the following, Π is a set of propositional symbols.

Definition 11 (Syntax): The set $L_{LTL}(\Pi)$ of LTL formulas over Π is defined inductively as follows:

- $true,\ false \in L_{LTL}(\Pi)$;
- If $p \in \Pi$ then $p \in L_{LTL}(\Pi)$;
- If $\varphi, \psi \in L_{LTL}(\Pi)$ then $(\neg\varphi), (\varphi \vee \psi), (\varphi \wedge \psi) \in L_{LTL}(\Pi)$;
- If $\varphi \in L_{LTL}(\Pi)$ then $(X\varphi) \in L_{LTL}(\Pi)$;
- If $\varphi, \psi \in L_{LTL}(\Pi)$ then $(\varphi U \psi) \in L_{LTL}(\Pi)$;
- If $\varphi, \psi \in L_{LTL}(\Pi)$ then $(\varphi R \psi) \in L_{LTL}(\Pi)$.

In Definitions 12 and 13, we define the LTL semantics.

Definition 12 (Local Satisfaction): Let $\sigma : \mathbb{N} \to 2^{\Pi}$, $t \in \mathbb{N}, p \in \Pi$ and $\varphi, \psi \in L_{LTL}(\Pi)$. The notion of satisfaction ($\Vdash$) is defined as follows:

- $\sigma(t) \Vdash true$ and $\sigma(t) \nVdash false$;
- $\sigma(t) \Vdash p$ if and only if $p \in \sigma(t)$;
- $\sigma(t) \Vdash (\neg\varphi)$ if and only if $\sigma(t) \nVdash \varphi$;
- $\sigma(t) \Vdash (\varphi \vee \psi)$ if and only if $\sigma(t) \Vdash \varphi$ or $\sigma(t) \Vdash \psi$;
- $\sigma(t) \Vdash (\varphi \wedge \psi)$ if and only if $\sigma(t) \Vdash \varphi$ and $\sigma(t) \Vdash \psi$;
- $\sigma(t) \Vdash (X\varphi)$ if and only if $\sigma(t+1) \Vdash \varphi$;
- $\sigma(t) \Vdash (\varphi U \psi)$ if and only if exists $t' \geq t$ such that $\sigma(t') \Vdash \psi$ and for all $t'' \in [t, t'[\ \sigma(t'') \Vdash \varphi$;
- $\sigma(t) \Vdash (\varphi R \psi)$ if and only if for all $t' \geq t$ such that $\sigma(t') \nVdash \psi$ exists $t'' \in [t, t'[$ such that $\sigma(t'') \Vdash \varphi$.

Definition 13 (Global Satisfaction): Let $\sigma : \mathbb{N} \to 2^{\Pi}$ and $\varphi \in L_{LTL}(\Pi)$. The notion of global satisfaction is defined as follows:

- $\sigma \Vdash \varphi$ if and only if $\sigma(0) \Vdash \varphi$.

Now, we give a brief explanation of each operator defined:

- The X operator is read "next", meaning that the formula it precedes will be true in the next state;
- The operator U is read "until", meaning that its first argument will be true until its second argument becomes true (and the second argument must become true in some state, i.e., an ω-string where φ is always satisfied but ψ is never satisfied does not satisfy $\varphi U \psi$);
- The operator R, which is the dual of U, is read "releases", meaning that its second argument must always be true until its first argument becomes true (in this case, an ω-string where ψ is always satisfied satisfies $\varphi R \psi$, because the definition does not require the existence of t').

There are two other commonly used temporal operators, F and G, usually defined by abbreviation.

Definition 14 (Abbreviations): Let $p \in \Pi$ and $\varphi, \psi \in L_{LTL}(\Pi)$. We define the following abbreviations:

- $(\varphi \Rightarrow \psi) \equiv_{abv} ((\neg\varphi) \vee \psi)$;
- $(\varphi \Leftrightarrow \psi) \equiv_{abv} ((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$;
- $(F\varphi) \equiv_{abv} (trueU\varphi)$;
- $(G\varphi) \equiv_{abv} (falseR\varphi)$.
- The F operator is read "eventually", meaning that the formula it precedes will be true in a future state;
- The G operator is read "always", meaning the formula it precedes will be true in all future states.

B. Büchi Automata

Büchi Automata are used to describe ω-languages, i.e., languages of infinite strings[1] (ω-strings). Büchi automata have

[1]One should notice that a function $\sigma : \mathbb{N} \to 2^{\Pi}$ is in fact an ω-string whose i-th element is given by $\sigma(i-1)$.

the same structure as FSA, The characteristic that sets them apart is their semantics, since for Büchi Automata one defines generated and marked ω-languages instead of generated and marked languages.

To define the generated and marked ω-languages by a Büchi Automaton, we need to introduce the notion of valid state labeling.

Definition 15 (Valid State Labeling): Let $B = (X, E, f, \Gamma, X_0, X_m)$ be a Büchi automaton and $\sigma \in E^\omega$ an ω-string. A valid state labeling for B and σ is a function $\rho : \mathbb{N} \to X$ such that:

- $\rho(0) \in X_0$
- $\rho(i+1) \in f(\rho(i), \sigma(i)),\ for\ all\ i \in \mathbb{N}$

We denote $P(B, \sigma)$ as the set of all possible valid state labelings for B and σ.

A valid state labeling for B and σ is an ω-string over the state set of B, where $\rho(i)$ is one of the possible states B can be in (in the deterministic case, the state where B is), while applying its transition function to σ_i. If, for some $i \in \mathbb{N}$, event $\sigma(i+1)$ is not active for any of the possible states B can be in, that is,

$$\sigma(i+1) \notin \bigcup_{x_0 \in X_0} \Big(\bigcup_{x \in f(x_0, \sigma_i)} \Gamma(x) \Big)$$

no such function exists.

Definition 16 (Generated ω-Language by Büchi Automata): Let $B = (X, E, f, \Gamma, X_0, X_m)$ be a Büchi automaton. We define the ω-language generated by B as

- $L(B) = \{\sigma \in E^\omega : P(B, \sigma) \neq \emptyset\}$

The generated ω-strings by B are the ones for which there exists a valid state labeling.

Definition 17 (Marked ω-Language by Büchi Automata): Let $B = (X, E, f, \Gamma, X_0, X_m)$ be a Büchi automaton. We define the ω-language marked by B as

- $L_m(B) = \{\sigma \in L(B) : exists\ \rho \in P(B, \sigma)\ such\ that\ inf(\rho) \cap X_m \neq \emptyset\}$

where, for $\chi \in X^\omega$, $inf(\chi) \subseteq X$ is the set of all $x \in X$ that appear infinite times in χ.
The marked ω-strings by B are the ones generated by "runs" of B that visit at least one of the marked states infinite times.

Now, we state the Theorem that allows us to perform Supervisory Control over a DES given a set of LTL formulas stating our performance objectives. The proof of this theorem is constructive and yields a method to construct the Büchi Automaton that marks the sequences that satisfy a given formula φ. [11] presents the most immediate proof of the theorem and [4] describes a most efficient method for the translation, which is used to calculate the examples we will present later.

Theorem 1: Let $\varphi \in L_{LTL}(\Pi)$. Then there exists a (non-deterministic) Büchi automaton B_φ such that

$$\sigma \Vdash \varphi \quad \text{if and only if} \quad \sigma \in L_m(B_\varphi)$$

V. Supervisor Synthesis

In this Section, we explain how to define the LTL - based supervisor for a plant G and a set of LTL formulas $\varphi_1, ..., \varphi_n,\ n \in \mathbb{N}$. As we have seen, the first step in building a standard realization of a supervisor S, such that $L(S/G) = L_a$ is to construct an FSA H_{spec} that captures the essence of our natural language specification. The construction of H_{spec} can be very error - prone and, in general, not obvious. On the other hand, translating natural language to LTL formulas is, in most cases, straightforward. Thus, we can define our performance objectives in LTL and use the Büchi Automaton referred in Theorem 1 to solve our problem in a much more user - friendly way.

Note that, in order to restrict $L(G)$ to L_a, we will be constructing LTL formulas over the set of propositional symbols E (G's event set), i.e., we will be interested in formulas $\varphi \in L_{LTL}(E)$. Since we assume the occurence of events in a DES to be asynchronous, at each state exactly one event can occur. This allows us to assume $\sigma : \mathbb{N} \to E$ in Definition 12 and substitute condition $\sigma(t) \Vdash p$ if and only if $p \in \sigma(t)$ by $\sigma(t) \Vdash e$ if and only if $\sigma(t) = e$, for $t \in \mathbb{N}$ and $e \in E$. Thus, given a Büchi automaton B_φ, we can delete all events that are not singletons in B_φ's event set and redefine B_φ's transition function accordingly.

Since a Büchi automaton's structure is the same as an NFA, we consider B_φ as an NFA. Next, we need to find the equivalent DFA, H_φ, of B_φ. This must be done because, if we build a supervisor from B_φ, it will disable some events that should not be disabled, due to the nondeterministic choices that are made when an event occurs at a given state and there is more than one state we can go to, e.g., if $f(x, e) = \{y, z\}$ we want the enabled events in state $f(x, e)$ to be $\Gamma(y) \cup \Gamma(z)$ but if we nondeterministically jump to state y we will not be enabling the events in $\Gamma(z) \setminus \Gamma(y)$. This problem is solved by using the equivalent DFA, thus keeping track of all the states B_φ can be in and enabling all the events that are active in at least one of those states. As seen in [5], finding the equivalent DFA of an NFA is an exponential operation, but, in general, the LTL formulas that are relevant to perform supervision yield small Büchi automata. Despite that, the complexity issue is a major one when applying this theory, as we will see in the next Section. Then, we obtain the supervisor $S_\varphi = G \parallel H_\varphi$ or $S_\varphi = G \times H_\varphi$, depending on our supervision problem. Using this method, we guarantee that for all $s \in L(S_\varphi/G)$, there exists $\sigma \in E^\omega$ such that $s\sigma \Vdash \varphi$, i.e., the generated language of the CDES S/G is always in conformity with the specification given by φ. Since the generated language by a CDES is a set of finite strings, this is the best we can have in this framework. We can now describe the method we will use for supervision. Given a plant G and a set of formulas $\{\varphi_1, ..., \varphi_n\},\ n \in \mathbb{N}$ representing the specifications we want G to fulfill, we build the supervisors $S_{\varphi_1}, ..., S_{\varphi_n}$, as explained above, and perform modular supervision, as explained in Section III. The use of modular supervision gives us a gain in efficiency ([9]) and, in addition, allows us to translate the formulas $\varphi_1, ..., \varphi_n$ to Büchi automata one by one, which also allows a significant improvement in the efficiency of the method: If $r_1, ..., r_n$ is the size (number of operators) of $\varphi_1, ..., \varphi_n$ respectively, then

- If we had not opted for modular control, to enforce all

the specifications given by $\varphi_1, ..., \varphi_n$ we would need to build a Büchi automaton B_φ for formula
  - $\varphi = \left(\bigwedge_{i=1}^{n} \varphi_i \right)$

  It is easy to see that φ has, at most, size
  - $r = (\sum_{i=1}^{n} r_i) + n - 1$

  where the $n-1$ factor is due to the $n-1$ "and" ($\wedge$) operators we added to φ. Hence, B_φ would have, at most, the following number of states (we have seen that the translation from an LTL formula to a Büchi automaton yields an automaton whose number of states is exponential in the size of the formula):
  - $|B_\varphi| = 2^r$
- Using modular supervision, we need to build n Büchi automata $B_{\varphi_1}, ..., B_{\varphi_n}$, which, altogether, have at most the following total number of states:
  - $\sum_{i=1}^{n} |B_{\varphi_i}| = \sum_{i=1}^{n} 2^{r_i}$

  which is clearly better than the previous option's worst case scenario.

VI. Examples

In this section, we present some applications of the framework defined throughout this work. We will build supervisors for the DES in Examples 2 and 3 that enforce the specifications we gave in natural language in Examples 5 and 6. To build these examples, some functions were implemented in *Matlab*. These functions can be found in *http://islab.isr.ist.utl.pt/ltldes_examples.zip*:

- A function that receives a NFA and outputs its equivalent DFA;
- A function that receives two FSA and outputs their product composition;
- A function that receives two FSA and outputs their parallel composition;
- A function that receives a set of LTL formulas and translates them to Büchi automata (this function uses the implementation described in [4] to build the Büchi automaton, which is written in *C* and adapts a *Matlab* function written for the implementation described in [7] to take the output of the *C* function and turn it into a viable *Matlab* structure);
- A function that, given a plant and n supervisors, simulates the feedback loop of modular control;
- A function that congregates all of the above. This function receives a plant and n LTL formulas, creates the supervisors and simulates the feedback loop of modular control.

Example 7 (Transporting Robots): Let's return to the transporting robots example and let G be the FSA represented in Example 2. In Example 5 we defined 4 specifications that prevent the robots from moving to a position where they are too far from the other, making the bar fall. Spec i can be translated to LTL by formula φ_i, where

- $\varphi_1 = (G(Move_2 \Rightarrow (X((\neg Move_1) U Stop_2))))$
- $\varphi_2 = (G(Move_1 \Rightarrow (X((\neg Move_2) U Stop_1))))$
- $\varphi_3 = (G(Move_1 \Rightarrow (X((\neg Move_1) U Stop_2))))$
- $\varphi_4 = (G(Move_2 \Rightarrow (X((\neg Move_2) U Stop_1))))$

Looking at these formulas, one can see that the events that can be disabled are $Move_1$ and $Move_2$. Hence, an admissible supervisor will be obtained. We construct the DFA $H_{\varphi_i},\ i = 1, 2, 3, 4$ from the Büchi automata, as explained before. In Figure 6 we represent the Büchi automaton obtained from φ_2.

Next, we obtain the 4 supervisors $S_i = G \parallel H_{\varphi_i}$. In Figure 7 we represent the supervisor S_2. Note that the states reached after an event $Move_1$ happens do not have the event $Move_2$ in their active event set. The modular supervisor $S_{mod1234}$ implements the robot alternation. The controlled system only allows 2 types of strings, $Move_1 - Stop_1 - Move_2 - Stop_2 - Move_1 - Stop_1 - Move_2 - Stop_2 - ...$ or $Move_2 - Stop_2 - Move_1 - Stop_1 - Move_2 - Stop_2 - Move_1 - Stop_1 - ....$ In Figure 8, we represent the automaton $G \times S_1 \times S_2 \times S_3 \times S_4$ which, as we have seen, represents the controlled system. One should notice that our controlled system is not minimum, i.e., there is a 5 states DFA that implements the robot alternation. This is one drawback of this method: in general the controlled system is not the smallest it could be.

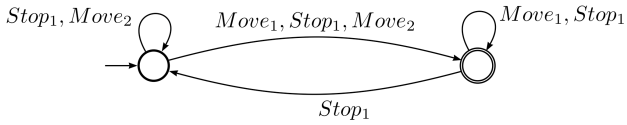


Fig. 6. Büchi automaton marking the ω - strings that satisfy φ_2

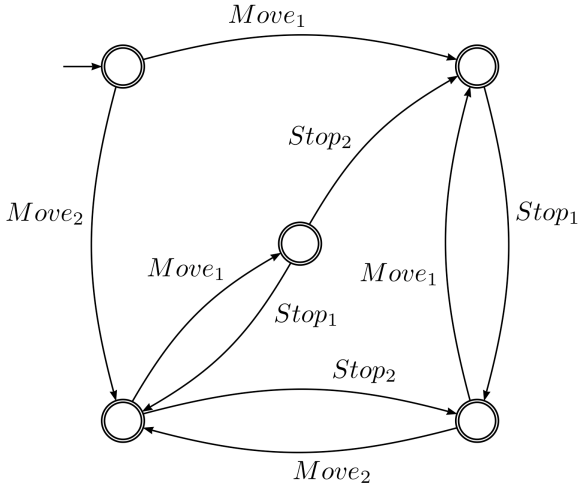


Fig. 7. The supervisor S_2, obtained by formula φ_2

Example 8 (Robotic Soccer): Regarding Example 6, it is easier to represent Spec $1, i,\ i = 1, ..., n$ by only one formula

- $\varphi_1 = (G[(\bigvee_i move\_to\_ball(i)) \Rightarrow (X[(\neg(\bigvee_i move\_to\_ball(i)))U(\bigvee_i kick\_ball(i))])])$

Formula φ_1 enforces that, after one robot moves to the ball (which means the team does not have the ball in its

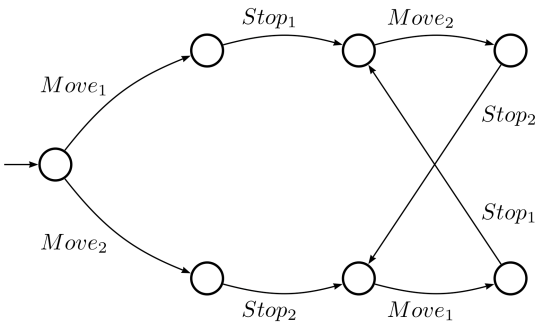


Fig. 8. Automaton representation of the controlled system, with the robot alternation implemented

possession), all the robots will not move to the ball until one of them shoots it (which means that the team lost the ball possession).

Spec 2, i is represented by formulas $\varphi_{2,i},\ i = 1, ..., n$, where

- $\varphi_{2,i} = ((\neg start\_receiving(i))$
  $\wedge(G[(\bigvee_{j\neq i} start\_passing(j,i))$
  $\Leftrightarrow (X start\_receiving(i))])$

Formula $\varphi_{2,i}$ enforces that a robot's first action cannot be getting ready to receive a pass and that, only when one of its teammates chooses it as a receiver, it gets ready to receive the ball and it gets ready as soon as possible.

These formulas do not refer to uncontrollable events, so checking that an admissible supervisor is yield is immediate.

The controlled system was tested for 3 robots. The plant has 729 states, the supervisor obtained by φ_1 has 100 states (the great reduction in the number of states is due to the fact that the plant allows more than one robot to have the ball in its possession and it is φ_1 that disallows this kind of situation) and the supervisors obtained by $\varphi_{2,i},\ i = 1, 2, 3$ have 1458 states each. Next, we give two examples of output of the simulation. One should notice that when when one robot is chosen by a teammate to receive a pass, it gets ready to receive it immediately and that robots only go to the ball when it is not in the team's possession (i.e. when it is kicked) and only go one at a time. In Simulations 1 and 2 we emphasize these situations respectively. In Simulation 3 we show the uncontrolled behavior of the system. The lack of restrictions imposed for this system allows it to regularly evolve to a *deadlock* situation.

Simulation 1 - move_to_ball(3) - close_to_ball(3) - get_ball(3) - move_to_goal(3) - close_to_goal(3) - kick_ball(3) - move_to_ball(1) - close_to_ball(1) - get_ball(1) - blocked_path(1) - *start_passing(1,2)* - *start_receiving(2)* - *pass(1,2)* - blocked_path(2) - *start_passing(2,1)* - *start_receiving(1)* - *pass(2,1)* - move_to_goal(1) - blocked_path(1) - *start_passing(1,3)* - *start_receiving(3)* - *pass(1,3)* - move_to_goal(3) - close_to_goal(3) - kick_ball(3) - move_to_ball(3) - close_to_ball(3) - get_ball(3) - move_to_goal(3) - blocked_path(3) - *start_passing(3,1)* - *start_receiving(1)* - *pass(3,1)* - close_to_goal(1) - kick_ball(1) - move_to_ball(1) - close_to_ball(1) - get_ball(1) - blocked_path(1) - *start_passing(1,3)* - *start_receiving(3)* - *pass(1,3)* - blocked_path(3) - *start_passing(3,1)* - *start_receiving(1)* - *pass(3,1)* - move_to_goal(1) - blocked_path(1) - *start_passing(1,2)* - *start_receiving(2)* - *pass(1,2)* - close_to_goal(2) - kick_ball(2) - move_to_ball(3) - close_to_ball(3) - get_ball(3)

Simulation 2 - *move_to_ball(1)* - *close_to_ball(1)* - *get_ball(1)* - blocked_path(1) - start_passing(1,3) - start_receiving(3) - pass(1,3) - move_to_goal(3) - blocked_path(3) - start_passing(3,1) - start_receiving(1) - pass(3,1) - blocked_path(1) - start_passing(1,3) - start_receiving(3) - pass(1,3) - move_to_goal(3) - close_to_goal(3) - *kick_ball(3)* - *move_to_ball(2)* - *close_to_ball(2)* - *get_ball(2)* - blocked_path(2) - start_passing(2,3) - start_receiving(3) - pass(2,3) - blocked_path(3) - start_passing(3,2) - start_receiving(2) - pass(3,2) - close_to_goal(2) - *kick_ball(2)* - *move_to_ball(1)* - *close_to_ball(1)* - *get_ball(1)* - close_to_goal(1) - *kick_ball(1)* - *move_to_ball(2)* - *close_to_ball(2)* - *get_ball(2)* - close_to_goal(2) - kick_ball(2) - *move_to_ball(3)* - *close_to_ball(3)* - *get_ball(3)* - move_to_goal(3) - blocked_path(3) - start_passing(3,1) - start_receiving(1) - pass(3,1) - close_to_goal(1) - kick_ball(1)

Simulation 3 - start_receiving(3) - move_to_ball(1) - move_to_ball(2) - close_to_ball(2) - close_to_ball(1) - get_ball(1) - blocked_path(1) - start_passing(1,2) - get_ball(2) - blocked_path(2) - pass(1,3) - move_to_ball(1) - move_to_goal(3) - start_passing(2,1) - close_to_goal(3) - close_to_ball(1) - get_ball(1) - kick_ball(3) - close_to_goal(1) - kick_ball(1) - move_to_ball(3) - close_to_ball(3) - move_to_ball(1) - get_ball(3) - blocked_path(3) - start_passing(3,2) - close_to_ball(1) - get_ball(1) - move_to_goal(1) - blocked_path(1) - start_passing(1,2)

VII. Conclusion

In this work, we defined a method to perform supervisory control of Discrete Event Systems using Linear-Time Temporal Logic. We introduced all the necessary theory to understand how the method works and gave some examples of application. Analyzing the examples, one can conclude that, with this method, the specification of supervisors for systems with an arbitrary number of components that must coordinate themselves is almost straightforward: all the formulas are written for an arbitrary $n \in \mathbb{N}$. Unfortunately, this advantage is somewhat shadowed by the high complexity of the method: despite writing the formulas for an arbitrary number of components, when performing the simulations we witnessed the great increase of the number of states, both in the plant and in the supervisors, which only allows the application of the method for systems with a relatively small number of components.

There are several paths one can follow to improve the method we just presented. The most obvious one is to try

to reduce its complexity. Another improvement is to increase the method's expressive power, for example by using CTL (a temporal logic that is incomparable with LTL) or CTL* (a temporal logic that contains both LTL and CTL) [6] as a way to specify the supervisors or by identifying each state of the DES model with a set of propositions that are satisfied in that state and build our LTL specification over those propositions, instead of building it over the DES' event set. One major advantage of this option is that it allows for more than one proposition to be satisfied at each state of the DES, unlike the method we presented, where only one is satisfied. One can also model the DES itself as a set of LTL formulas, as seen in [8], avoiding the construction of any automaton by hand (which can be very error-prone). Another option is to define a similar logic to LTL, but with its semantics defined over finite string, avoiding the need to use Büchi Automata. A final suggestion is to develop this theory in order to cover other aspects of Supervisory Control. For example, being concerned with marked languages and deal with blocking issues or introduce the notion of unobservable events [2].

REFERENCES

[1] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins and George J. Pappas, Symbolic Planning and Control of Robot Motion. In IEEE Robotics & Automation Magazine:61-70, March 2007

[2] Christos G. Cassandras and StŐphane Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.

[3] E. Allen Emerson, Temporal and Modal Logic. In Handbook of theoretical computer science (vol. B): formal models and semantics, MIT Press:995 - 1072 1991

[4] Paul Gastin and Denis Oddoux, Fast LTL to Büchi Automata Translation, LIAFA, UniversitŐ Paris 7

[5] John E. Hopcroft, Motwani, Rajeev Ullman, Jeffrey D., *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison-Wesley, 2001

[6] Shengbing Jiang and Ratnesh Kumar, Supervisory Control Of Discrete Event Systems with CTL* Temporal Logic Specifications. In Proceeding of the 40th IEEE, Conference on Decision and Control:4122-4127, December 2001

[7] Marius Kloetzer and Calin Belta, A Fully Automated Framework for Control of Linear Systems From LTL Specifications. In Lecture Notes in Computer Science , J. Hespanha and A. Tiwari, Eds. Berlin, Germany: Springer-Verlag, vol 3927:333-347, 2006

[8] Jing-Yue Ling and Dan Ionescu, A Reachability Synthesis Procedure for Discrete Event Systems in a Temporal Logic Framework. In IEEE Transactions on Systems, Man, and Cybernetics, VOL. 24, No. 9:1397-1406, September 1994

[9] Peter J. G. Ramadge and W. Murray Wonham, The Control of Discrete Event Systems. In *Proceedings of the IEEE*, Vol. 77, No. 1:81-98, 1989.

[10] Paulo Tabuada and George J. Pappas, Linear Time Logic Control of Discrete-Time Linear Systems. In *IEEE Transactions on Automatic Control*, Vol. 51, No. 12:1862-1877, 2006.

[11] Pierre Wolper, Constructing Automata from Temporal Logic Formulas: A Tutorial. In *Lectures on Formal Methods in Performance Analysis*, Vol. 2090:261-277 of *Lecture Notes in Computer Science*, Springer-Verlag, 2001