Cooperative Leader Following in a Distributed Multi-Robot System

Stefano Carpin and Lynne E. Parker
Center for Engineering Science Advanced Research
Computer Science and Mathematics Division
Oak Ridge National Laboratory (ORNL)
Oak Ridge, TN 37831-6355, U.S.A.
E-mail: shamano@dei.unipd.it, parkerle@ornl.gov

Abstract— *The cooperative leader following task for multi-robot teams is introduced and discussed. We describe the design and implementation of a distributed technique to coordinate team level and robot level behaviors for this task, as well as a multi-threaded framework for the implementation of a heterogeneous multi-robot system. This approach enables robots to remain in formation as they deal with other obstacles that may appear within the formation. We describe how single robot behaviors are realized and scheduled. The proposed approach has been run and validated on a team of robots performing both in indoor and outdoor environments.*

I. Introduction

In this paper we address the problem of *leader following* in the case of a heterogeneous multi-robot team. This task requires the robots to move in a *linear* pattern, each following the previous robot, with the first robot either following a human operator, being teleoperated, or going through a predetermined path. The task can be seen as a *service task* for bringing the robots to a work area. Of course, the team is required to perform in a *robust* way, so that the team is able to correctly operate even if some external undesired or unforeseen event occurs. Examples of unexpected events include an obstacle in the way or one or more of the robots failing.

We propose a distributed policy based on explicit communication that allows this goal to be achieved at the team level. At the single robot level we introduce a multi-threaded structure that can be employed on different robots, and we also give the details about the tracking techniques. This framework allows us to abstract the coordinated tracking process from the low-level sensor details.

This paper is organized as follows: related work is discussed in Section II, while our approach at the team level is introduced in Section III, and at the robot level in Section IV. Details about the software architecture are provided in Section V, and in Section VI we briefly outline how the various sensors are used for leader following and obstacle avoidance. Finally in Section VII we present additional implementation results, with conclusions offered in Section VIII.

This work was done while S. Carpin visited ORNL. He is currently with the Dept. of Electronics and Informatics, The University of Padova, Italy shamano@dei.unipd.it

II. Related work

The task of pattern formation and formation marching has gained a lot of attention in previous years, with the former being a preliminary step for the latter. Indeed there are some tasks that intrinsically require the movement of a multi-robot system in a specified pattern, such as in military applications, demining or surveillance. However, much of this previous research has been done in simulation only and just a few projects have been implemented and tested on real robot systems. We now briefly overview the most significant work in relation to our research. In [1] an implementation on a physical team of robots is discussed. A class of reactive behaviors that implement formations is introduced and tested on a team of military unmanned ground vehicles performing in an outdoor terrain. While this work is similar to ours in terms of the behavior-based approach, the main differences are that we do not use a global positioning system (e.g., GPS) and we also deal with a heterogeneous system in which robots are equipped with different sets of sensors. Potential field approaches are widely used for formation keeping (see for example [10]). Robots move being attracted to their position in the pattern and being repulsed from obstacles and other robots. In [3] the idea of moving a team by means of one robot that leads the rest of the robots is discussed and some simulation results are shown. The issue of designing control laws for this kind of problem is discussed in [8] and [9]. In contrast to much of this previous research, our research explicitly addresses issues of maintaining formation in a leader-following application while also avoiding obstacles that may unexpectedly appear within the formation.

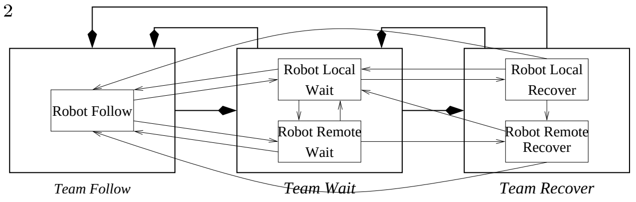

Fig. 1. Team-level (bold lines and italics font) and robot-level (thin lines and roman font) behavior transitions.

III. Team Level Behaviors

We address the following problem: given a set of possibly heterogeneous robots arranged in a linear pattern, design a strategy so that if the first moves in an unknown environment, the other follows while at the same time avoiding obstacles that may appear within the formation. Of course the line pattern can be modified, for example, to avoid an obstacle suddenly appearing in the proximity of a robot. The invariant, however, is that each robot (except the first) should follow the previous robot. We call this task *Cooperative Leader Following*. Indeed, the proposed task is cooperative since to be reached at the team level, every robot has to reach its own goal at the local level. Clearly, if one robot loses the tracking of its leader with all the following robots behind it, the team fails. As usual in multi-robot team design, it is necessary to match group goals with individual goals. We utilize a behavior-based approach ([2]) and distinguish between *team-level* behaviors and *robot-level* behaviors.

At the group level we use a situated automaton ([6]) approach, with the team seen as a finite state automaton whose inputs come from the environment. The transitions between the three group-level behaviors are shown in Figure 1.

At the team level we introduce three behaviors:

• **Team-Follow**: if the team is executing this behavior, every robot (except the first) will follow its local leader.
• **Team-Wait**: this behavior is executed when the team is waiting for some event to happen. This is the case when, for example, a moving object approaches a robot, so that it is not safe to keep moving. In this case all robots stop to avoid breaking the formation.
• **Team-Recover**: this behavior is executed when the team is trying to recover from a *wait* condition. Since not all robots are involved in a collision danger situation, when the team is performing this behavior single robots will execute different behaviors, with some robots trying to go around obstacles, and others simply performing regular following at a reduced speed.

The introduction of the *Team-Wait* behavior is motivated by the assumption that the team will not operate in an interference-free environment, but rather in an unknown, possibly unstructured, environment shared with other moving entities, such as humans or other robots. In this scenario, it could happen that a moving object approaches the robot, so that it is necessary for it to stop to avoid a collision. The robot first *waits* for a certain amount of time for the obstacle to go away. If this time is exceeded, then the robot attempts to circumnavigate the obstacle – i.e. to *recover* from the situation and to resume the *follow* behavior. The wait stage is introduced because recovering is a difficult task, and it is preferred to execute it only when there is no alternative. Of course, when a robot is waiting, other robots should wait too, to keep the formation together. From the above discussion it is evident that some sort of *explicit* communication is necessary to gain the desired team level behavior, especially for large team sizes.

IV. Robot Level Behaviors

At the single robot level we designed a set of five behaviors that, locally executed, give the team level behaviors previously described (see Figure 1). They are:

• **Robot-Follow**: the robot is following its leader.
• **Robot-Local-Wait**: the robot is waiting because an obstacle does not let it move safely.
• **Robot-Remote-Wait**: the robot is waiting because one or more other robots are in *Robot-Local-Wait*.
• **Robot-Local-Recover**: the robot is trying to recover from a *Robot-Local-Wait* situation. This means that it is trying to overtake an obstacle while keeping track of its leader.
• **Robot-Remote-Recover**: the robot is following its leader but at a reduced speed, so that if its follower is doing a *Robot-Local-Recover* behavior, it will be easier for the robot to keep tracking while overtaking the obstacle.

Figure 1 outlines the relationship between team-level behaviors and robot-level behaviors. So, the team is doing a *Team-Follow* behavior if all the single robots are doing a *Robot-Follow*. The team is in a *Team-Wait* behavior if all the robots are doing either *Robot-Local-Wait* or *Robot-Remote-Wait*. Finally, the team is doing a *Team-Recover* behavior if all the robots are doing either *Robot-Local-Recover* or *Robot-Remote-Recover*. So, at the team level the switching between different behaviors is triggered by explicit communication, while at the single robot level the triggering comes both from sensors and from communication.

At the single robot level the switching between different behaviors is triggered from inputs which come from sensors and communication. To make this structure independent of the number of robots in the team, each robot is modeled as a *Situated Counter Machine*. A counter machine ([4]) is a computational model which lies between finite state automatons and push-down automatas. Informally, a counter machine is a finite state automata augmented with a counter that can count arbitrary numbers, so that the next state choice is based not only on the current state and input (from sensors and communcation), but also on the counter value, which can be updated during the transition. This choice has been made so that the same algorithm will work independently of the size of the team, as described below. Indeed, a finite state automata is not capable of this, since it cannot count arbitrary numbers, due to the fixed *finite* number of different states.

Each robot perceives its environment through its sensors, which in addition to tracking data (such as the actual position of the robot being tracked or of the obstacles to be avoided) identify two high level conditions – namely *Local Warn Begin* (LWB) and *Local Warn End* (LWE). A LWB condition means that an obstacle has approached the robot, so that it is no longer possible to move in a safe manner, while a LWE condition means that the problem which caused the LWB condition is no longer present. Clearly, a LWE condition occurs if and only if a LWB condition has previously ocurred.

Additionally, the communcation system provides two high-level messages to other robots, which are *Remote Warn Begin* (RWB) and *Remote Warn End* (RWE). RWB and RWE mean, respectively, that another robot is in the LWB or LWE condition. When a robot is in a LWB or a LWE condition, it communicates this to the team, so that other members can receive a RWB or RWE, respectively.

Finally, it is necessary to deal with the switching from *Wait* behaviors to *Recover* behaviors. When a robot is in a LWB condition, it starts a timer, and after a fixed amount of time expires, T_{timer}, it goes from *Robot-Local-Wait* to *Robot-Local-Recover*, communicating this to the other robots, so that they can switch from *Robot-Remote-Wait* to *Robot-Remote-Recover*. All of this communcation is performed in a *broadcast* manner, so that the robot that communicates can ignore how many otherrobots are listening. Similarly, the robot that receives a message is not required to know who sent it.

It is possible that while a robot is performing a Robot-Remote-Recover behavior, it enters the LWB condition, so that it has to start its local timer. Moreover, there could be more than one robot waiting to switch to the *recover* behavior. It is for this reason that *counters* are introduced. There are two counters – *Warn* (W_n) and *Timer* (T_n). Warn counts the number of robots that are in a local warn condition, while Timer counts the number of robots which are waiting for the timer to expire. So when both *Warn* and *Timer* are 0, the team is performing *Team-Follow*, when both are positive the team is performing *Team-Wait*, and when *Warn* is positive and *Timer* is 0 the team is in *Team-Recover*. Table I describes how a robot locally changes its state (and then its behavior), based on the values of the two counters.

The use of counters and timers is somewhat similar to those in [7], which were shown to be effective. In the physical robot implementation, each robot has its own copies of the counters, and the broadcast messages are used to keep all these copies in a consistent state. However, we recognize that the communication system and/or robots will not be perfect, and that at times messages will be lost or never sent. For instance, a robot could fail after entering the LWB condition, perceived as a RWB from the others, causing it to not be able to send the corresponding RWE or timeout message. To avoid the team being stopped from this event, all the robots start an internal timer when they receive a RWB message. If after a reasonable amount of time the corresponding RWE or timeout message does not come, the first who recognizes this situation issues the missing message, so that the team can resume.

V. Dealing with a Heterogeneous Team

To test the proposed framework, we implemented this approach on a heterogeneous team of five Emperor robots, shown in Figure 2. The robots are heterogeneous in that they are equipped with different sensor suites and have different mobile platforms. We developed a multi-threaded software architecture in which each sensor is handled by a separate thread that uses its data to get information about tracking and navigation. Each thread then sends its output (e.g., the polar coordinates of the point to track) to a *Decisor* thread which, on the basis of the desired behavior and sensory input, drives the robot. The Decisor first merges the provided points to track and then decides how to move on the basis of a set of *fuzzy rules*. A separate thread handles explicit communcation. Each robot is equipped with sonar sensors that provide distance readings within 4 meters. Three of the robots are equipped with a SICK laser range finder. Also, three of them have a Sony camera mounted on a pan-tilt unit. Each robot is equipped with wireless Ethernet, which allows them to communicate over the standard TCP/IP protocol. Finally, we note that even though these robots also have a Differential Global Positioning System (DGPS), we did not use it in this research because of its frequent unavailability (e.g., due to indoor

B_n	T_n	W_n	I_n	B_{n+1}	T_{n+1}	W_{n+1}
Follow	0	0	LWB	Local Wait	1	1
Follow	1	1	-	Remote Wait	1	1
Local Wait	n	1	LWE	Remote Recover	n-1	0
Local Wait	1	1	LWE	Follow	0	0
Local Wait	n	m	LWE	Remote Wait	n-1	m-1
Local Wait	n	m	LTO	Timer Elapsed	n-1	m
Timer Elapsed	0	m	-	Local Recover	0	m
Timer Elapsed	n	m	LWE	Remote Wait	n	m-1
Local Recover	0	m	LWE	Remote Recover	0	m-1
Local Recover	0	1	LWE	Follow	0	0
Local Recover	1	m	-	Timer Elapsed	1	m
Remote Wait	n	m	LWB	Local Wait	n+1	m+1
Remote Wait	0	m	-	Remote Recover	0	m
Remote Wait	0	0	-	Follow	0	0
Remote Recover	n	m	LWB	Local Wait	n+1	m+1
Remote Recover	n	m	-	Remote Wait	n	m
Remote Recover	0	0	-	Follow	0	0

TABLE I

Counter machine transition function. In this table, B_n is the current behavior, T_n is the *Timer* counter, W_n iw the *Warn* counter, and I_n is the current condition.

Fig. 2. Heterogeneous Emperor robots used in our experiments. Four of the robots are ATRV-minis, and one robot is a Transit robot.

operations, satellite obstructions, etc.).

VI. Tracking the Leader

A. Laser PLS Sensor

The SICK PLS (Proximity Laser Scanner) sensor provides readings with a scan angle of 180 degrees and an angular resolution of 0.5 degrees. Distances returned under 15 meters are considered reliable. Starting from this sensor data, the tracking routine removes spikes and averages across samples to obtain a smoother sequence (see Figure 3).

From the smoothed sequence, the routine then extracts all the local minima in the sequence and then

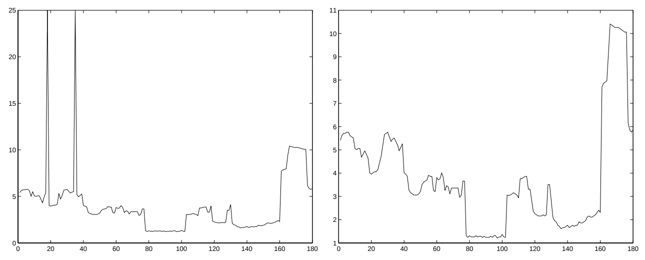
Fig. 3. Raw readings from the PLS sensor and a smoothed sequence (note the difference in the two scales).

tries to match each one with the previous tracked minima[1]. The one which is closest is considered to be the new local minima to track and will be tracked in the next step. The routine returns the polar coordinates (distance, ρ, and angle, θ) of the minima to track. This is the information that the thread dealing with the laser will provide to the Decisor module. The chosen local minima is the one which minimizes the quantity:

$$d = \sqrt{(\rho_c \cos\theta_c - \rho_t \cos\theta_t)^2 + (\rho_c \sin\theta_c - \rho_t \sin\theta_t)^2}$$

where (ρ_c, θ_c) are the polar coordinates of the candidate local minima and (ρ_t, θ_t) are the coordinates of the minima currently being tracked. This formula gives the distance on the cartesian plane between the

[1] It is assumed that at the beginning all the robots are arranged in a line pattern, so that at the first scan cycle the current position of the leader is roughly known.

candiate local minima and the currently tracked minima. In addition, during the local minima search, the routine verifies that no obstacle is too close to the robot. If this is not the case, the LWB condition is raised and the corresponding LWE condition is issued when the obstacle moves away.

B. CCD Camera

As previously mentioned, three of the robots are equipped with a CCD camera mounted on a pan-tilt unit. The associated framegrabber returns a color image in the RGB coordinate space of 120×160 pixels. Robots equipped with a camera have to detect and follow in real time a similar robot in their field of view. The designed approach for detecting the position of the leader robot is as follows:

- *Color segmentation* is accomplished by defining a range of colors that are accepted as possibly the red color of the ATRV robots, discarding all other pixels.
- *Averaging (Smoothing)* is achieved using a neighborhood averaging technique in which pixel color is updated by the eight surrounding pixels. If four or more are red it is set to red, otherwise it is set to white (white pixels are ignored in the following steps).
- *Blob detection* is done by checking the boundaries of 25 pixels. When a red pixel along the boundary of a region is found, the region is flagged as a possible hit.
- *Object assignment* gives a different label to each connected component, using the iterative algorithm by Haralick ([5]).
- *Object selection* decides which of the objects should be tracked. This is done by comparing the center of mass of every distinct object with the position in the image plane of the previous object being tracked (in a similar way to what is done in the laser routine, as outlined in VI-A).
- *Proximity estimation* gives a rough estimation of the distance to the robot, based on the dimension of the blob being tracked, and of its position on the image plane.

Thus, as for the laser, the camera-handling thread gives a position to track in polar coordinates (i.e., distance and direction). Figure 4 shows examples of the detection of robot position using this approach.

C. Sonar

Every robot is equipped with sonar sensors. In our experiments they have been used for obstacle avoidance only, and not for tracking, because from a number of trials it became evident that crosstalk and environmental conditions were such that tracking could be obtained only within a small distance range. Instead, they proved to be highly effective for obstacle avoidance. The thread which deals with sonars iden-

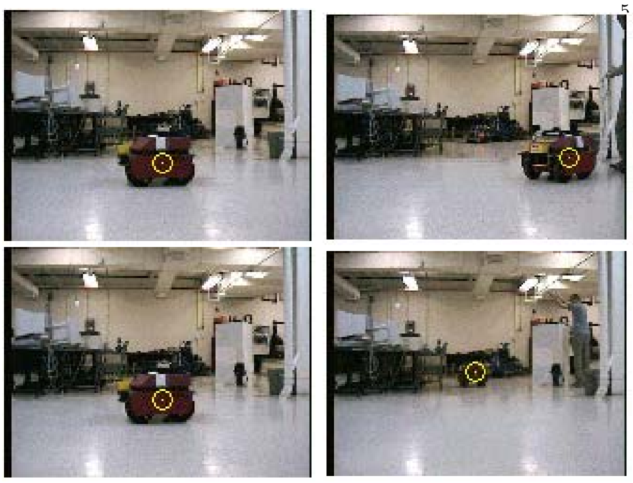

Fig. 4. Results of visual tracking approach. Small white circles indicate locations of robots detected by the visual tracking algorithm.

tifies the LWB and LWE conditions, as for the laser, and in addition it produces a vector whose direction and intensity indicate the direction and distance to obstacles. Since readings from sonars come in polar coordinates (ρ_i, θ_i), the cartesian components[2] x_r and y_r of the resulting vector are calculated as:

$$x_r = \sum_{i=1}^{n} f(\rho_i) \cos\theta_i \qquad y_r = \sum_{i=1}^{n} f(\rho_i) \sin\theta_i$$

where n is the number of samples returned by the sonar sensor and f is the function plotted in Figure 5.

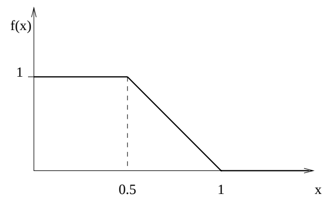

Fig. 5. Sonar weight function, where x is in meters.

VII. Robot Team Experiments

The proposed framework has been implemented and tested both in indoor and outdoor environments using teams of three to five robots. Figure 6 shows the robots performing these behaviors in an outdoor grassy environment, while figure7 shows the robots performing these behaviors in an outdoor gravel environment.

Figure 8 gives an example of the robot state changes that occur to maintain formations when the robots encounter obstacles within their formation. In this figure, all three robots are initially in the *Robot-Follow*

[2]Here, the X axis is directed along the heading of the robot and the Y axis is perpendicular and positive to the left.

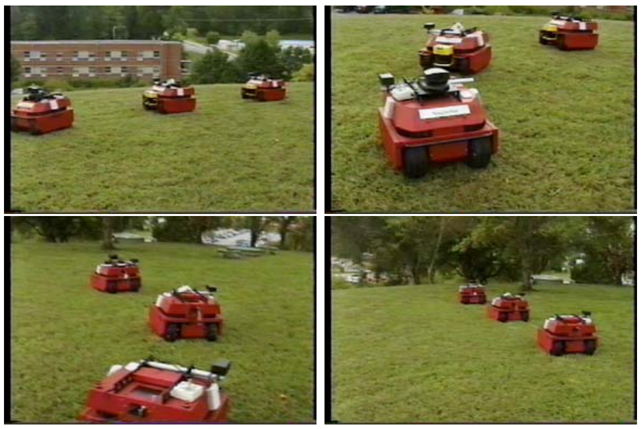

Fig. 6. Results of approach implemented on robots operating in an outdoor grassy environment.

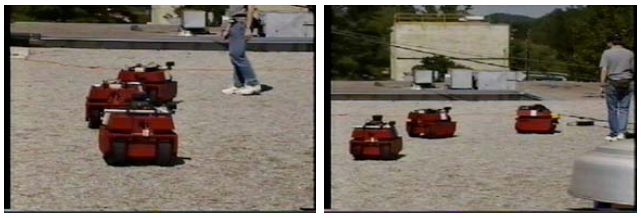

Fig. 7. Results of approach implemented on robots operating in an outdoor gravel environment.

behavior. Then, at time $T0$, Robot 1 encounters an obstacle that puts it into the *Robot-Local-Wait* behavior, causing the other two robots to enter the *Robot-Remote-Wait* behavior. At time $T1$, after waiting a period of time for the obstacle to leave but with the obstacle still in the way, Robot 1 enters the *Robot-Remote-Wait* behavior, causing the other two robots to enter the *Robot-Remote-Recover* behavior. At time $T2$, Robot 3 itself then encounters an obstacle, causing it to go into the *Robot-Local-Wait* behavior. When that obstacle does not move, Robot 3 enters the *Robot-Local-Recover* behavior at time $T3$. However, since Robot 1 had not yet completed moving around its obstacle, it also enters the *Robot-Local-Recover* behavior at this time. Robot 2 enters the *Robot-Remote-Recover* behavior. Then, at time $T4$, Robot 1 successfully passes its obstacle moves to the *Robot-Remote-Recover* behavior to wait on Robot 3 to complete the bypass around its obstacle. At time $T5$, Robot 3 completes its obstacle bypass, and all robots return to the *Robot-Follow* behavior.

The introduction of the *wait* behavior proved to be effective in reducing the number of recover stages, where it is more difficult to both go around an obstacle and to keep track of the leader.

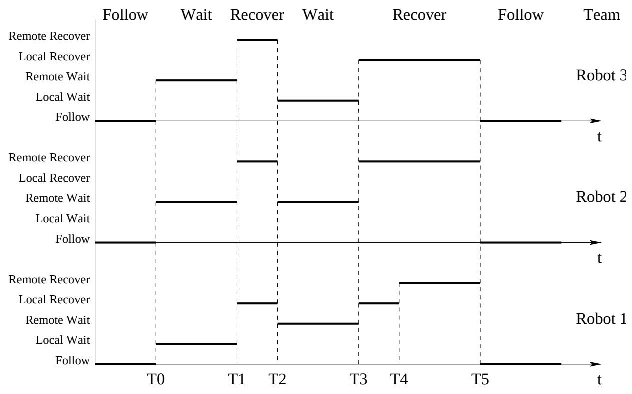

Fig. 8. An example of local behavior scheduling in the case of a three robot team. In this figure, the behaviors in the left-most column refer to robot behaviors, while the behaviors across the top refer to the team behaviors.

VIII. Conclusions

A distributed technique for the coordinated motion in a linear pattern of a multi-robot team has been illustrated. The framework, based on explicit anonymous broadcast communcation, is fully scalable with the size of the team and deals with communcation failures. The proposed approach has been implemented and validated over a heterogeneous multi-robot team performing both in indoor and outdoor environments.

Acknowledgments

The authors would like to thank Hunter Brown for his work in developing the visual tracking algorithm.

References

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

[2] R. Brooks. A Robust Layered Control System For A Mobile Robot. it IEEE Journal of Robotics and Automation, 2(1):14-23, 1986.

[3] J.P. Desai, V. Kumar, J.P. Ostrowski. Control of changes in formation for a team of mobile robots. In *Proc. ICRA 1999*, pp. 1556-1561.

[4] P.C. Fischer, A.R. Meyer, A.L. Rosenberg. Counter Machines and Counter Languages. *Math. Systems Theory*, 2(3):265-283, 1968.

[5] R.M. Haralick. Some Neighborhood Operators. *Real-Time Parallel Computing Image Analysis*. M. Onoe, K. Preston, A. Rosenfeld (Eds.), Plenum, New York, 1981.

[6] L. Kalebling, S. Rosenschein. Action and Planning in Embedded Agents. In *Designing Autonomous Agents*, ed. P. Maes, MIT Press, Cambridge, MA, 1990, 35-48.

[7] L.E. Parker. *Heterogeneous Multi-Robot Cooperation*, Ph.D. Dissertation, MIT, 1994.

[8] L.E. Parker. Designing Control Laws for Cooperative Agent Teams. In *Proc. of ICRA 1993*, Vol.3, pp. 582-587.

[9] L.E. Parker. Cooperative Motion Control for Multi-Target Observation. In *Proc. of IROS 1997*, pp. 1591-1598.

[10] J.H. Reif, H. Wang. Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots. K. Goldberg (Ed.) *Algorithmic Foundations of Robotics*, AK Peters, 1995, pp. 331-336.