Need a Fleet? Use the Force!

VINCENZO GERVASI
Università di Pisa, Italy

GIUSEPPE PRENCIPE
Università di Pisa, Italy

Keywords

Fun, Experimental Algorithms, Mobile Robots, Flocking.

1 Prologue

The theme of this paper, and the main motivation for writing it, was provided to us by Professor Chew Becca, from the Tattooine University, who was visiting our Department a few months ago. He presented us with a problem that seemed to be — to him — of the greatest practical importance. His stay at our Department was too brief for us to completely solve the problem, but we are reporting here on the progress that we have made after his visit.

Prof. Becca's problem was apparently simple. As is, by now, well known, the Rebels had infiltrated an important Imperial base with one of their best agents — call him Lucky Skywalker. Due to the chronic shortage of vessels that afflicted the Rebel's fleet at the time, Lucky's mission was to "appropriate" the greatest possible number of Imperial ships and send them, without them being intercepted, through a large extension of Empire-controlled space. The ultimate destination of the ships was to be a secret Rebel base, where the ships were to be reworked and were to become part of the Rebel fleet.

However, Lucky's mission was placed under a few other constraints. The ships would have to escape unnoticed by the Imperials, always proceeding in standard fleet formation, and they could have no crew on board, since no other agents were available. Furthermore, the ships could not communicate with each other, to avoid the risk of their radio communication being intercepted. Worst of all, in order to further strengthen his grip on the entire universe, the Dark Emperor himself had had his latest operating system, Windows 3000, installed on the navigational computers of all the ships in the Imperial fleet. This left no available memory on the computers to use as storage for an auto-pilot algorithm. Finally, the Rebels did not want to provide the coordinates of their secret base to the auto-pilots, as the Empire then could have traced them and attacked the base in force.

It was thus decided to simulate a chase: the stolen ships, in autopilot-driven fleet formation (with no memory and no communication) would follow a small but very fast Rebel ship, the Millennium Bug Falcon — piloted by the famous rebel leader Son Solo — as if they were to intercept her. The Rebels hoped that by simulating a chase, the stolen fleet would attract little attention, and no interference, from other Imperial forces. In reality, the Millennium Bug Falcon would lead the ships to the Rebel base, and Lucky Skywalker could continue working behind the lines on his next mission.

In this paper, we provide a solution to Prof. Becca's problem. In the next section, we present an overview of related works. In Section 3 we present the model under which we will study the flocking problem. In Section 4 we present an algorithm solving the problem, and in Section 5 some experimental results are discussed. Some conclusions, and an account of what happened to Prof. Becca's friends, conclude the paper.

Publication of an extended version of this work in encrypted e-book form is being sought through the People's Revolutionary Committee for Science and Culture. Permission pending.

2 Overview

Control and coordination of a set of autonomous vehicles that can freely move on a plane[1] is a widely studied topic in robotics. The focus on this kind of problem has grown in recent years because of the increased interest in studying systems populated by many, simple units, instead of few, powerful ones. In particular, these units simply observe the environment by using their sensors, and react following simple rules: the reaction to the environmental stimuli is called the *behavior* of the unit. Despite the simplicity of the units, it has been shown that these groups of vehicles can exploit rather complex group behaviors [15]. Moreover, such a system is preferable to one made up of just one large and powerful unit for many reasons: low costs in changing faulty units; ability to solve tasks otherwise unsolvable by a single unit [7, 17] (e.g., robots that have to move big objects [16]); certain problems can be solved quicker [3] (e.g., robots that are asked to clean a room [13]); for fault tolerance considerations; the decreased cost through simpler individual unit design. An extensive survey can be found in [5].

One major question that arises is: How is it possible to properly coordinate these group of mobile robots, such that they can together accomplish what they are asked to do? And another question is: How simple can these units be [8]?

In this paper we study the *flocking problem*: a set of mobile units are required to follow a leader unit while keeping a predetermined formation (i.e., they are required to move in flock, like a group of soldiers). Moreover, the units in the

[1]Or an approximation of it, e.g., the surface of a suitably large planet.

flock do not know beforehand the path the leader will take: their task is just to follow him wherever he goes, and to keep the formation while moving. This is a problem often studied in robotics, with several applications (e.g. military [4], or in factories, where robots can be asked to move heavy loads). The approach usually adopted to study this and similar problems, is to design solutions based on heuristics and tailored on the capabilities of the robots employed, and then test them by computer simulations, or on real robots.

For instance, in [15], experiments are conducted on a team of simple mobile units in order to produce *complex* behaviors, by compounding *basic* ones (such as safe-wondering, i.e. the ability to avoid collisions while moving; dispersion, i.e. the ability of the robots to spread out over an area; aggregation, i.e. the ability to gather; and homing, i.e. the ability to reach a predetermined destination). In particular, the author points out that flocking can be obtained combining safe-wandering, aggregation, dispersion, and homing. Hence, in her experiments, all the units have a common destination to reach.

T. Balch and C. Arkin studied formation and navigation problems in multi-robot teams. In particular in [2] the problem of specifying the behavior for the navigation of a mobile unit is analyzed, and results of both computer simulation and real experimentation are reported. In [4] the approach is extended to multi-robot teams that navigate the environment maintaining particular formations: in particular the case of a line, column, diamond and wedge are examined. In their study, the authors assumed that the path along which the group of robots has to move is known in advance to every unit. This same assumption is made in [6], where the robots are asked to move in a matrix shape along a path represented by a straight line followed by a right turn and then a straight line again. In contrast, in this paper we do not assume any knowledge by the followers of the path that the leader will follow. The followers have only a common description of the formation they have to keep while moving.

A similar problem is studied in [18], where the author derives equations describing navigational strategies for robots moving in formation, and following the movement described by one (ore more) leader. In the studied framework, the robots have identities, hence their positions in the formation are fixed. Moreover, in order for the i-th robot to compute its position at time t, it has to know the position of either the $(i-1)$-th robot or the leader at time t. Hence, some degree of synchrony has to be introduced in order to implement these strategies.

In this paper, we analyze the flocking problem by using very simple units and by dropping the assumption that all the ships in the flock know the path, or that they can derive it (e.g. by observing the orientation of the leader's prow, or by deriving it by observing the leader in different positions). Following the motivations that prompted previous studies ([9, 12]), we adopt simple units to study the problem: the ships are completely anonymous, identical (no identities are used during the computation), asynchronous, memoryless, and with no means of direct communication. We describe an algorithm (the same for all the ships) that allows the followers to keep a formation given to them as input, while following a path determined at run-time by the leader, that acts completely independently and does not behave according to the followers' algorithm. Moreover, we present results of computer simulations that show the effectiveness of the proposed solution.

3 The Model

We consider[2] a system of autonomous mobile units (ships). There are two kinds of ships in the environment: the *leader* and the *followers*. The leader acts independently from the others, and we can assume that it is driven by a human pilot. In the following we will discuss only about the followers.

Each ship is capable of observing its surrounding, computing a destination based on what it observed, and moving towards the computed destination; hence it performs an (endless) cycle of observing, computing, and moving.

Each ship has its own *local view* of the world. This view includes a local Cartesian coordinate system having an origin (that without losing generality we can assume to be the position of the ship), a unit of length, and the *directions* of two coordinate axes (which we will refer to as the x and y axes), together with their *orientations*, identified as the positive and negative sides of the axes. In general, there is no agreement among the followers on the chirality of the local coordinate systems (i.e., the ships do not share the same concept of where North, East, South, and West are).

The ships are modeled as units with computational capabilities, which are able to freely move in the plane.[3] They are equipped with sensors that let each ship observe the positions of the others with respect to their local coordinate system. Each ship is viewed as a point, and can see all the other ships in the flock (and the leader).

The ships act totally independently and *asynchronously* from each other, and do not rely on any centralized directives, nor on any common notion of time. Furthermore, they are *oblivious*, meaning that they do not (need to) remember any previous observation nor computations performed in the previous steps. Note that this feature gives the algorithms designed in this model the nice property of self-stabilization [10]: in fact, every decision taken by a follower can not depend on what happened in the system previously, and hence is not based on corrupted data stored in its local memory. They have as input, however, the same pattern $\mathcal{F}$ representing the flock to be kept. $\mathcal{F}$ is described as a set of coordinates in the plane, relative to a point representing the leader.

The ships in the flock are *anonymous*, meaning that they are a priori indis-

[2]The model we adopt is based on one introduced in [11, 9]. It has been adapted, however, to the particular problem under study.

[3]Following illustrious precedents [1], we only consider movement in a two dimensional plane here, as "flat" fleet formations are standard procedure in the Imperial Fleet.

tinguishable by their appearances, and they do not (need to) have any kind of identifiers that can be used during the computation. They can only distinguish if a ship is the leader or a fellow follower. Moreover, there are no explicit direct means of communication; hence the only way they have to acquire information from the fellow ships is by observing them[4].

They execute the same algorithm, which takes as input the observed positions of the ships, and returns a destination point towards which the executing ship moves. A ship, asynchronously and independently from the other ships, (1) observes the environment (*Look*), by taking a snapshot of the positions of all other ships with respect to its local coordinate system. Each ship is viewed as a point, and therefore its position in the plane is given by its coordinates. The observation returns the positions of all the other ships in the plane (leader and followers). (2) It computes a destination point p according to its oblivious algorithm (*Compute*); the local computation is based only on the current (i.e., at the time of the previous Look) locations of the observed ships. (3) Finally, the ship moves an unpredictable amount of space towards p (*Move*), which is however assumed to be neither infinite, nor infinitesimally small (see Assumption A2 below), and goes back to the Look state. Hence, there is no assumption on the maximum distance a ship can travel before observing again (apart from the bound given from the destination point that has to be reached). The life of a follower consists in repeating an endless cycle of *states* (1)–(3). Moreover, the only assumptions made in the model are the following [11]: **(A1)** The time for a ship to complete a Look-Compute-Move cycle is neither infinite nor infinitesimally small (i.e., is finite and bounded from below). **(A2)** For each follower f, there exists an arbitrary (small) constant $\delta_f > 0$, representing the minimum distance it travels in the Move state; if the computed destination point is closer than δ_f, f will reach it. **(A3)** Since we need to model ships that "continuously" move, we assume that the time spent in looking and computing is negligible compared to the time spent in moving.

Summarizing, each ship moves totally independently and asynchronously from the others, not having any bound on the time it needs to perform a *Move*, hence a cycle (it has to be, however, finite by Assumption A1); therefore, a ship can be seen *while* it is moving; in addition, they are oblivious, and anonymous. Moreover, no one of the followers knows in advance the path that the leader will follow, nor can it derive it at run-time. Their only task is to observe where the leader and the other followers are, reach an agreement — without communicating — on how and where to form the pattern in the plane, and move to positions such that the flock is formed and maintained.

Adopting such "simple" units aims at understanding what kind of complex tasks can be achieved, and under which conditions (for a detailed discussion on this model and its motivations, refer to [11, 9, 10, 12]).

[4]The obliviousness of the ships also renders the observations weaker. In fact, nothing observed in the past can be remembered, hence used in order to let the ships organize themselves to accomplish their task.

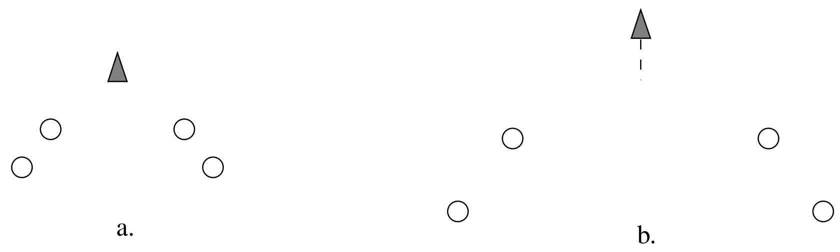

Figure 1: The triangle represents the leader, and the circles the positions of $\mathcal{F}$. If the ships did not have a common unit distance, then they could scale $\mathcal{F}$, and the flock would not follow L while it moves.

4 An Algorithm for the Flocking Problem

In this section we present an algorithm to solve the flocking problem that works in the general setting where no kind of common knowledge on the direction and the chirality of the followers' local coordinate system is assumed. Every follower f_i is given in input a formation $\mathcal{F}$ described as a set $p_1, \dots p_{|\mathcal{F}|}$ of points, relatives to the leader ship, L; we clearly assume to have $|\mathcal{F}|$ followers arbitrarily placed at the beginning (all the ships have distinct positions), where $|\mathcal{F}|$ denotes the total number of points in the formation.

We note that, since we want the ships to form a specific formation and *to keep that* formation while moving, it is necessary for the followers to agree on a common unit measure. Otherwise (i.e. if the input pattern can be scaled), the formation would be formed even if, instead of following the leader, the followers would simply "scale" the input formation (that is, we would have a situation where the ships would form a pattern that becomes bigger and bigger as the leader goes farther away, see Figure 1). Hence,

Observation 1 *In order for the problem to be significant, the followers must agree on the unit measure.*

The intuition behind Algorithm 1 is described in the following (see also Figure 2). First, the generic follower f gets the positions of the leader L and of the other followers (with respect to its local coordinate system), by calling `Look()` in Step 1. Then it computes the baricenter B of the followers' positions (Step 2), by executing `Baricenter(`*Followers*`)`, and a shared vertical axis Y given by the line passing through L and B, and oriented according to $\overrightarrow{LB}$ can be derived: this is accomplished by `Get_Y_axis(`L,B`)`, that returns the axis Y that all the followers will use to agree on orienting themselves in the plane.[5] Then S_0, S_1 and

[5]If $L = B$, the followers can simply wait for the leader to move away from B.

Algorithm 1 The Flocking Algorithm.

Input: The formation $\mathcal{F}$ to be kept, relative to the leader.

$(Followers, L) :=$ `Look()`;
$B :=$ `Baricenter(`$Followers$`)`;
$Y :=$ `Get_Y_axis(`L, B`)`;
$S_0 := \{$Ships on $Y\}$;
$S_1 := \{$Ships on the left of $Y\}$;
$S_2 := \{$Ships on the right of $Y\}$;
$F :=$ `Final_Positions(`$\mathcal{F}, L, Y$`)`;
$B_F :=$ `Baricenter(`F`)`;
$F_0 := \{$Final Positions on $Y\}$;
$F_1 := \{$Final Positions on the Left of $Y\}$;
$F_2 := \{$Final Positions on the Right of $Y\}$;
For All $j = 0, 1, 2$ **Do**
  `Sort(`F_j, L, B_F`)`;
  `Sort(`S_j, L, B`)`;
End For
Case *me* in
  $\bullet S_1$
    $k :=$ `Rank(`me, S_1`)`;
    If $k \leq |F_1|$ **Then**
      `Move(`k-th position in F_1`)`.
    Else If $k \leq |F_1| + |F_0| - |S_0|$ **Then**
      $H := \{$Ships in S_1 whose rank $> |F_1|\} \bigcup$
        $\{$Ships in S_2 whose rank $> |F_2|\}$;
      `Sort(`H, L, B`)`;
      $k' :=$ `Rank(`me, H`)`;
      If I am the only ship in H with rank k' **Then**
        $p := (k' + |S_0|)$-th *slot* in F_0;
      Else
        $p :=$ random *slot* between the $(k' + |S_0|)$-th
          and the $(k' + |S_0| + 1)$-th in F_0;
      `Move(`p`)`.
    Else
      `Move(`$(|F_2| - (k - |F_1| - |F_0| + |S_0|) + 1)$-th position in F_2`)`.
  $\bullet S_2$
    /* This case is simmetric to the previous one */
  $\bullet S_0$
    $k :=$ `Rank(`me, S_0`)`;
    If $k \leq |F_0|$ **Then**
      `Move(`k-th position in F_0`)`.
    Else If $|S_1| \leq |S_2|$ **Then**
      `Move(`$(|F_1| - |S_1| + (k - |F_0|))$-th position in F_1`)`.
    Else
      `Move(`$(|F_2| - |S_2| + (|k - |F_0|))$-th position in F_2`)`.

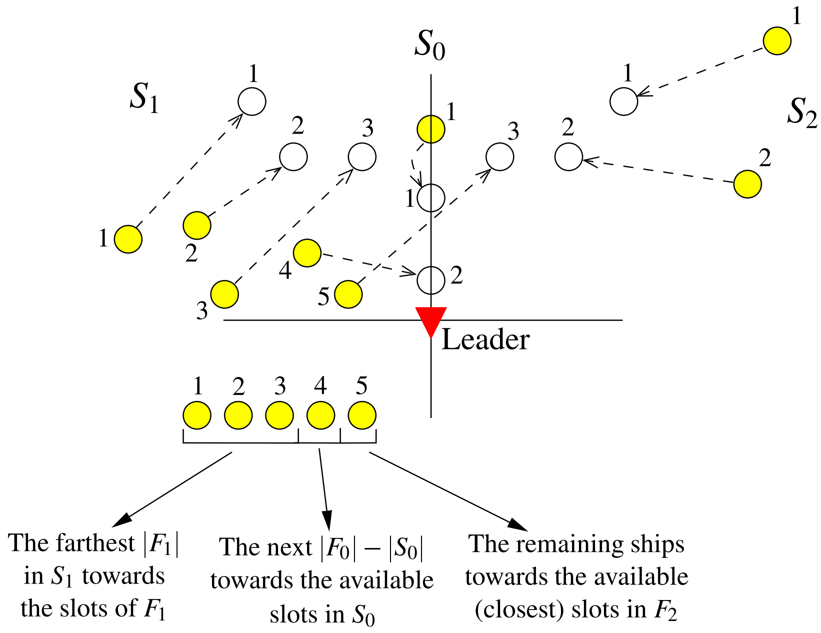

Figure 2: Some cases of Algorithm 1. The filled triangle is the leader, the filled circles are the followers, and the empty circles the *slots* the followers want to reach.

S_2, containing respectively ships whose positions are exactly on Y, to its left, and to its right, are computed (according to the local concept of *left/right of* Y).

At this point, f executes `Final_Positions` $(\mathcal{F}, L, Y)$ (Step 7), that rotates the points in $\mathcal{F}$, assuming that the leader is moving according to the direction and orientation of Y, and translates them into the observed leader's position. We will call the positions returned by this routine, the *slots* of $\mathcal{F}$ (these are the positions that the followers will try to reach). After having computed the baricenter B_F of the *slots* in Step 8, these positions are partitioned in three subsets: those exactly on Y (F_0, Step 9), to the left of Y (F_1, Step 10), and to its right (F_2, Step 11). Then, F_j, $j = 0,1,2$, are sorted in decreasing order with respect to the distances from L and B_F (Step 13), and S_j, $j = 0,1,2$, are sorted in decreasing order with respect to the distances from L and B (Step 14). These sorting operations are done by `Sort` (A, l, b), where A is the array (set of points) to be sorted. In particular, after the sorting, it is guaranteed that

$$\forall i, j,\ i < j \Rightarrow (d(l, p_i) > d(l, p_j)) \vee (d(l, p_i) = d(l, p_j) \wedge d(b, p_i) > d(b, p_j)),$$

where p_i and p_j are points in A, and $d(q,r)$ denotes the distance between points q and r. Next, the rank k of f in the subset it belongs to is computed, by `Rank` $(me,\cdot)$.

Now, if f is the k-th follower in S_1, and $k \leq |S_1|$, then it moves towards the k-th position in F_1 (Step 20; a similar argument applies if f is in S_2, see Step 35). Otherwise, if there are positions *available* in S_0 (i.e., $|F_0| > |S_0|$ and $k - |F_1| \leq |F_0| - |S_0|$), f is directed towards S_0. In particular, Step 22 computes the set H containing the ships in S_1 and S_2 whose rank is respectively bigger than $|F_1|$ and $|F_2|$; H is then sorted, and the rank k' of f in H is computed in Step 25. If f is the only ship with rank k', then it is directed towards the $(k' + |S_0|)$-th *slot* in F_0 (Step 27), that is towards a *slot* in F_0 that is not a target of ships in S_0 (refer to Step 35 to see how ships in S_0 choose their target). If there is another ship in H with rank k' (i.e., f is at the same distance from L and B as a ship in S_2), then f randomly chooses as a destination a *slot* between the $(k' + |S_0|)$-th and the $(k' + |S_0| + 1)$-th in F_0 (Step 29). This is the only[6] case in which f does a random computation. If no position in S_0 is available, f moves towards the $(|F_2| - (k - |F_1| - |F_0| + |S_0|) + 1)$-th position in F_2, that is towards one of the *slots* in F_2 that are not a destination point of either a ship in S_1 whose rank is smaller than k, or of a ship in S_2 (Step 33).

If f is in S_0, and its rank k is smaller than $|F_0|$, then it simply moves towards the k-th *slot* in F_0 (Step 39). Otherwise, it chooses to move towards the side that has fewer ships (note that if $|S_1| = |S_2|$, then it chooses to move towards a slot in F_1). In Figure 2, an example of how the followers chose their *slots* is depicted.

Finally, `Move` (p) moves the executing ship towards p, and terminates the current cycle. As already pointed out in Section 3, in general the ship does not reach p in one *Move* (the distance it travels is finite, but unpredictable). Clearly, since the ships can not remember p in the next cycle (obliviousness), this implies that it is possible that f changes its destination point in the next cycle, because its ranking can change.[7]

Since the ships are memoryless, they can not be sure of the direction of movement of L. They only assume that the leader is going away from B (i.e. according to $\overrightarrow{BL}$). Furthermore, the followers assume that the direction of movement of L is given by the axis passing through B and L, and oriented from B towards L, hence they can reach an agreement on Y. They can not, however, reach in general a similar agreement on X, that is on an axis hortogonal to Y that would let them agree on the concept of *left* and *right*. Hence,

[6] We are assuming here that ships are inpenetrable, i.e., two ships cannot occupy exactly the same point in space.

[7] This can be avoided, e.g. to have guaranteed (slower) convergence rather than (faster) statistical convergence. To this end, each ship can compute for each follower f_i a circle centered in L and having $\overline{Lf_i}$ as radius. This circle represents the rank of follower f_i: all the ships that are outside this circle have higher rank than f_i; all the ones inside it, have a lower rank. By choosing a course towards its destination point that avoids intersecting any *rank's circle*, f can avoid changes in ranking while moving.

Observation 2 *Algorithm 1 applies only to formations that are symmetric with respect to the direction of movement of L.*

Let us denote with v_L and v_f respectively the maximum linear velocity of the leader and of the generic follower f, and with ω_L the maximum angular velocity of L. Clearly, if we want the followers to be able to follow L, the leader must be slower than the slowest follower, that is $v_L < \min_f v_f$. Furthermore, when the leader turns, all the *slots* will turn with a velocity related to ω_L. In particular, the maximum linear velocity of a slot is $v_L + \omega_L \cdot r$, where $r = \max_{p \in \mathcal{F}} d(L, p)$, i.e. the maximum distance between L and the points in $\mathcal{F}$. Therefore, in order for the leader to not be "too fast", it must be $\omega_L \cdot r + v_L < \min_f v_f$. Hence, we can state the following

Theorem 1 *In order for the formation to be kept in movement, it is necessary for the leader to have* $\omega_L \cdot r + v_L < \min_f v_f$.

5 Analysis of Experiments

Let $f_i(t)$ be the *absolute position* of follower f_i at time t (i.e., its position w.r.t. an inertial reference frame), $L(t)$ the absolute position of L at time t, $B(t)$ the absolute position of B at time t, and $Y(t)$ the axis passing through $L(t)$ and $B(t)$. Moreover, let Π be the set of all possible permutations of $1, \ldots, |\mathcal{F}|$.

Definition 1 *We define the distance $\mathcal{D}$ of the ships' positions from a formation $\mathcal{Z}$ at time t, as follows: $\mathcal{D}(\mathcal{Z})^t = \min_{\pi \in \Pi} \sum_{i=1}^{|\mathcal{F}|} d(f_i(t), z_{\pi(i)})$, where z_j is a point in $\mathcal{Z}$.*

Let $\mathcal{F}(t)_{Y(t)}$ be the set of positions in $\mathcal{F}$ translated w.r.t. $L(t)$, and rotated w.r.t the orientation of $Y(t)$, and $\mathcal{F}(t)_{L(t)}$ the set of positions in $\mathcal{F}$ translated w.r.t. $L(t)$, and rotated w.r.t the direction of movement of the leader at time t. To measure how far the ships are at time t from the aimed formation, we use the following functions: $\Delta_r(t) = \mathcal{D}(\mathcal{F}(t)_{L(t)})^t$ and $\Delta_e(t) = \mathcal{D}(\mathcal{F}(t)_{Y(t)})^t$. In particular, $\Delta_r(t)$ represents how far the followers are from the *real* formation, that is the formation rotated w.r.t. the actual heading direction of L at time t. $\Delta_e(t)$ represents how far the followers are from the *estimated* formation, obtained from the position of the baricenter at time t.

In Figure 6, some plots of Δ_e and Δ_r are reported, relative to computer simulations run with six followers trying to keep a wedge shaped formation, four in a line, ten in a spread formation (all shown in Figure 3), and with random formations.

In all cases, the simulations started with all the ships (leader and followers) randomly placed in a square area with a side of 512 units. Each ship also had a random direction and orientation of the axes for its local coordinate system. Each

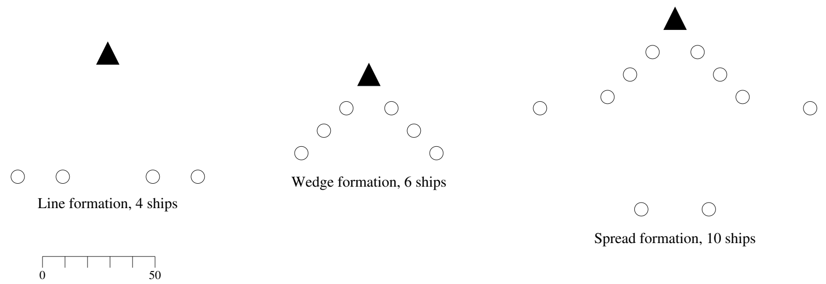

Figure 3: Fleet formations used in the simulations.

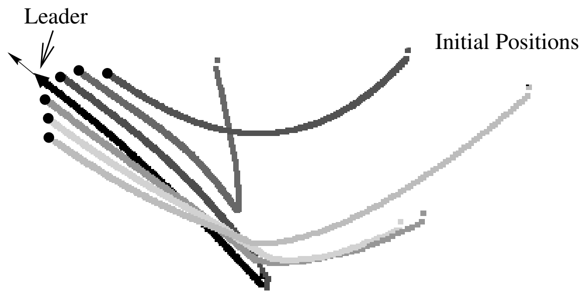

Figure 4: Trace of the ships while forming and keeping a wedge shaped formation.

follower had a random velocity between 0.5 and 5.0 units/move, while the leader's speed was determined in accordance with the limitations of Theorem 1.

The leader's course was determined as follows: at all times, the leader would move forward according to its velocity. At each move, with a probability of 1/20, the leader could start turning to its left or right with an angular speed limited again by Theorem 1. If already turning, with the same probability the leader could stop and continue its course as a straight line. Figures 4 and 5 show the courses of the ships in two simulation runs.

In Figure 6, it can be observed how convergence to the estimated formation (Δ_e) is obtained usually in less than half the time needed to reach the real formation. Figure 6 also reveals other interesting phenomena. In the Δ_r graph for the wedge formation, we can observe a "plateau" caused by the ships forming the pattern in exactly the wrong direction, i.e. in front of the leader rather than behind it. This correctly formed, but incorrectly-headed pattern, is kept until the leader starts changing its direction, at which point all the followers rapidly reach their proper position behind the leader. Related effects can also be observed in the Δ_r graphs for the other formation, in which instabilities in the distance are caused by

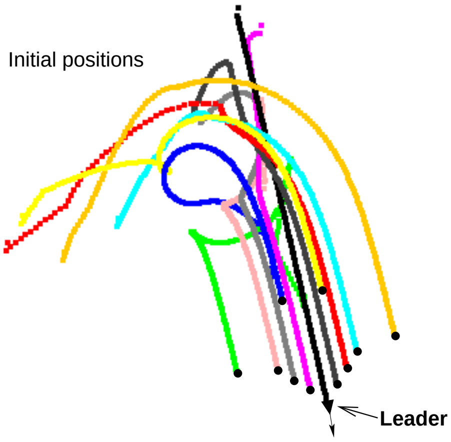

Figure 5: Trace of the ships while forming and keeping the spread formation with ten ships. Note the circular trajectory of the ships at the beginning, while trying to align the formation with the course of the leader.

the followers trying to catch up with change of directions of the leader.

All our experiments demonstrated that the algorithm is properly behaved, and in all cases the followers were able to assume the desired formation and to maintain it while following the leader ship along her route. Indeed, while Theorem 1 provides conditions under which the ability of the ships to follow the leader is guaranteed, even when those conditions were relaxed the followers were usually able to compensate for sudden turns or accelerations of the leader (as long as the effective speed of the leader remained, at least on average, lower than that of the slowest follower). As a concluding remark, we note that the obliviousness of the algorithm contributes to this result, since the followers do not base their computation on past leader's positions.

6 Epilogue

In this paper we have presented a novel algorithm for misappropriating space ships or, more specifically, for coordinating a set of non-communicating, asynchronous and memoryless vehicles into following a leader while keeping a fixed formation. The algorithm only assumes the vehicles share a common unit of distance, but no common sense of direction (i.e., a common coordinate system) is needed, nor any previous knowledge of the path the leader will follow. Moreover,

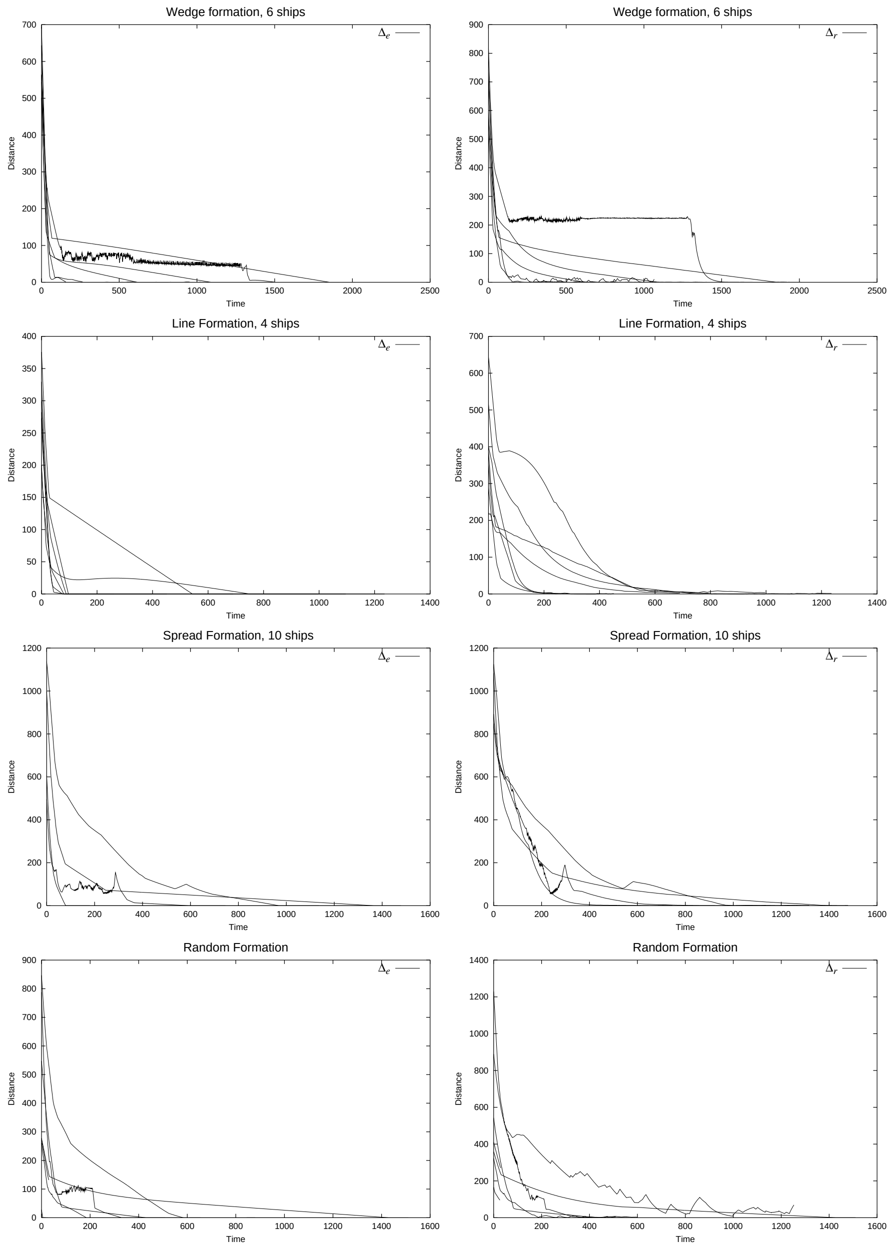

Figure 6: Some plot of the Δ_e and Δ_r, while forming a *wedge* of six ships, a *line* of four ships, a spread formation with ten ships, and random formations (with a number of ships variable between two and eight).

the followers do not need to have an identity, and are not distinguishable in any way one from the other.

Indeed, the algorithm we propose exhibits remarkable robustness, and numeric simulations indicate that in most cases the formation is reached in a relatively short time and kept after that, as Professor Becca wanted.

In fact, it has been reported to us [14] that Lucky Skywalker indeed succeeded in seizing the ships, and that the autopilot software based on an early version of our algorithm worked flawlessly, so that the stolen ships formed and maintained a perfect formation during their escape.

Unfortunately, the valiant attempt by the rebels took place on February 8, 3106 EC. While our algorithm is totally independent of time (and indeed, follower ships are not required to have any notion of time, nor of their own speed), the on-board software of the Millennium Bug Falcon was not. Captain Solo could only blame himself for not having anticipated that exactly on that date the 32-bit counter used to keep track of the date and time (counting the number of seconds since the midnight of January 1, 2970) would have rolled over. This caused a chain of failures in various navigation subsystems, that left his ship constantly turning on a close circle. The followers followed in perfect formation. Imperial forces were puzzled at the strange show of a chase being conducted in such circumstances, and in the end intervened and arrested Captain Solo.

Had not the Dark Emperor been slain by a flock of ferocius penguins while he was trying to install his latest Operating System on their brains, that gross failure could well have led to a definitive defeat for the Rebels. But Destiny had other plans, and after the not necessarily premature demise of the Emperor, the entire Empire collapsed. The Rebellion in the end triumphed, and we all can only be glad that Freedom, Democracy, and Abundant University Funding have at last spread to the entire known universe.

References

[1] A. Square (Edwin A. Abbot). *Flatland – A Romance of Many Dimensions*. 1884.

[2] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *Int. J. of Robot. Res.*, 8(4):92–112, 1989.

[3] R. C. Arkin and T. Balch. *Cooperative Multiagent Robotic Systems*, chapter in *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, D. Kortenkamp and R. P. Bonasso and R. Murphy edition, 1998.

[4] T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Trans. on Robot. and Autom.*, 14(6), 1998.

[5] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng. Cooperative Mobile Robotics: Antecedents and Directions. In *IEEE/TSJ Int. Conf. on Intell. Robots and Sys.*, pages 226–234, 1995.

[6] Q. Chen and J. Y. S. Luh. Coordination and Control of a Group of Small Mobile Robots. In *Proc. of IEEE Int. Conf. on Robot. and Autom.*, pages 2315–2320, 1994.

[7] B. R. Donald, J. Jennings, and D. Rus. Analyzing Teams of Cooperating Mobile Robots. Technical Report TR 94-1429, Dept. of Comp. Sci., Cornell University.

[8] E. H. Durfee. Blissful Ignorance: Knowing Just Enough to Coordinate Well. In *International Conference on MultiAgent Systems (ICMAS)*, pages 406–413, 1995.

[9] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In *10th International Symposium on Algorithm and Computation (ISAAC '99)*, pages 93–102, 1999.

[10] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed Coordination of a Set of Autonomous Mobile Robots. In *IEEE Intelligent Vehicles Symposium (IVS2000)*, pages 480–485, 2000.

[11] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Asynchronous Mobile Robots with Limited Visibility. In *18th International Symposium on Theoretical Aspects of Computed Science (STACS 2001)*, 2001.

[12] G.Prencipe. A New Distributed Model to Control and Coordinate a Set of Autonomous Mobile Robots: The CORDA Model. Technical Report TR-00-10, Univ. di Pisa, 2000.

[13] D. Jung, G. Cheng, and A. Zelinsky. Experiments in Realising Cooperation between Autonomous Mobile Robots. In *5th International Symposium on Experimental Robotics (ISER)*, June 1997. Barcelona, Catalonia.

[14] Leia D. Princess. Private communication.

[15] M. J Matarić. *Interaction and Intelligent Behavior*. PhD thesis, MIT, May 1994.

[16] F. R. Noreils. Toward a Robot Architecture Integrating Cooperation between Mobile Robots: Application to Indoor Environment. *The Int. J. of Robot. Res.*, pages 79–98, 1993.

[17] L. E. Parker. Adaptive Action Selection for Cooperative Agent Teams. In *Proc. Second Int'l. Conf. on Simulation of Adaptive Behavior*, pages 442–450, 1992.

[18] P. K. C. Wang. Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation. *J. of Robot. Sys.*, 8(2):177–195, 1991.

Vincenzo Gervasi is a Research Fellow at the Computer Science Department of the University of Pisa. E-mail: `gervasi@di.unipi.it`

Giuseppe Prencipe is a Ph.D. student at the Computer Science Department of the University of Pisa. E-mail: `prencipe@di.unipi.it`